ggplot2

Jiayue LIU, Soukaina ELGHALDY

15 novembre 2020

Contents

Ce Tuto décrit comment créer un nuage de point avec le package ggplot2

Préparer les données

Le jeu de données mtcars est utilisé dans les exemples ci-dessous

```
# convertir la colonne cyl envariable de type facteur
mtcars$cyl <- as.factor(mtcars$cyl)
head(mtcars)
```

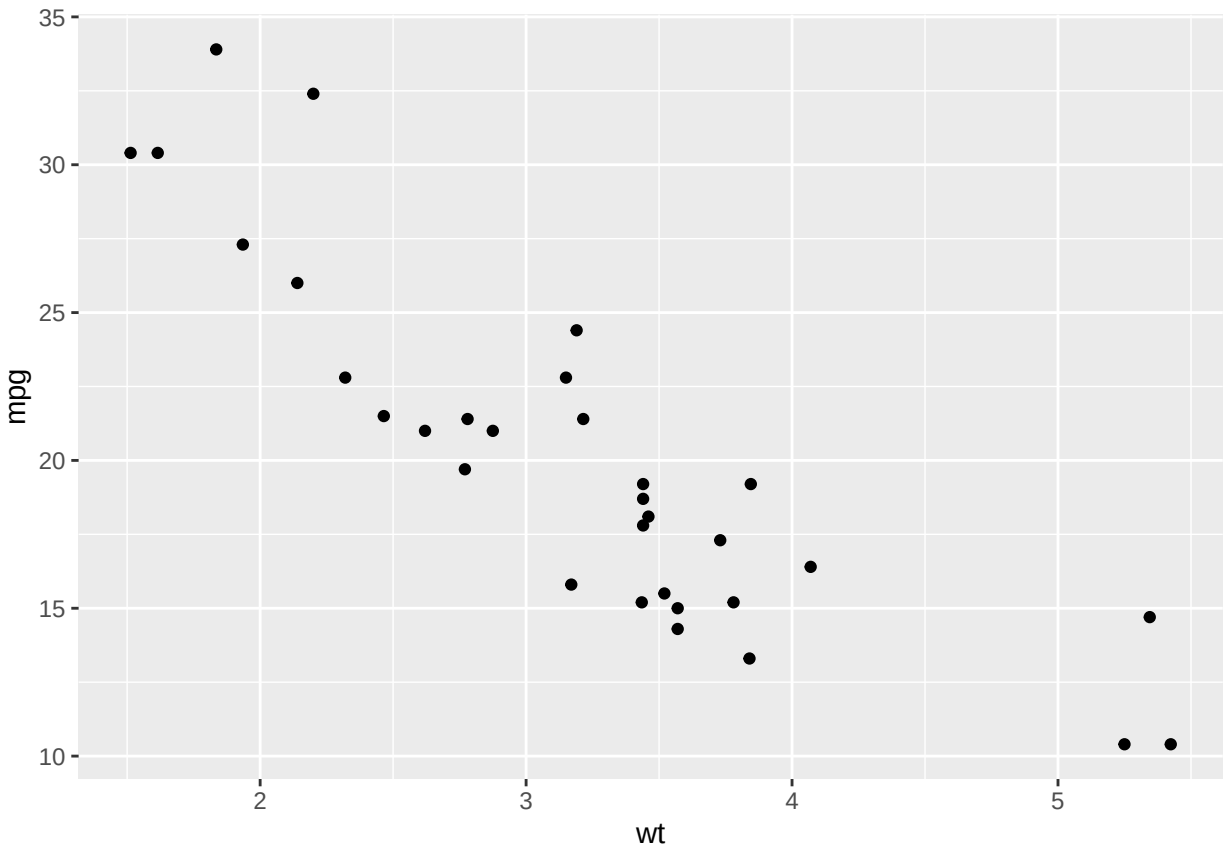
```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
```

```
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

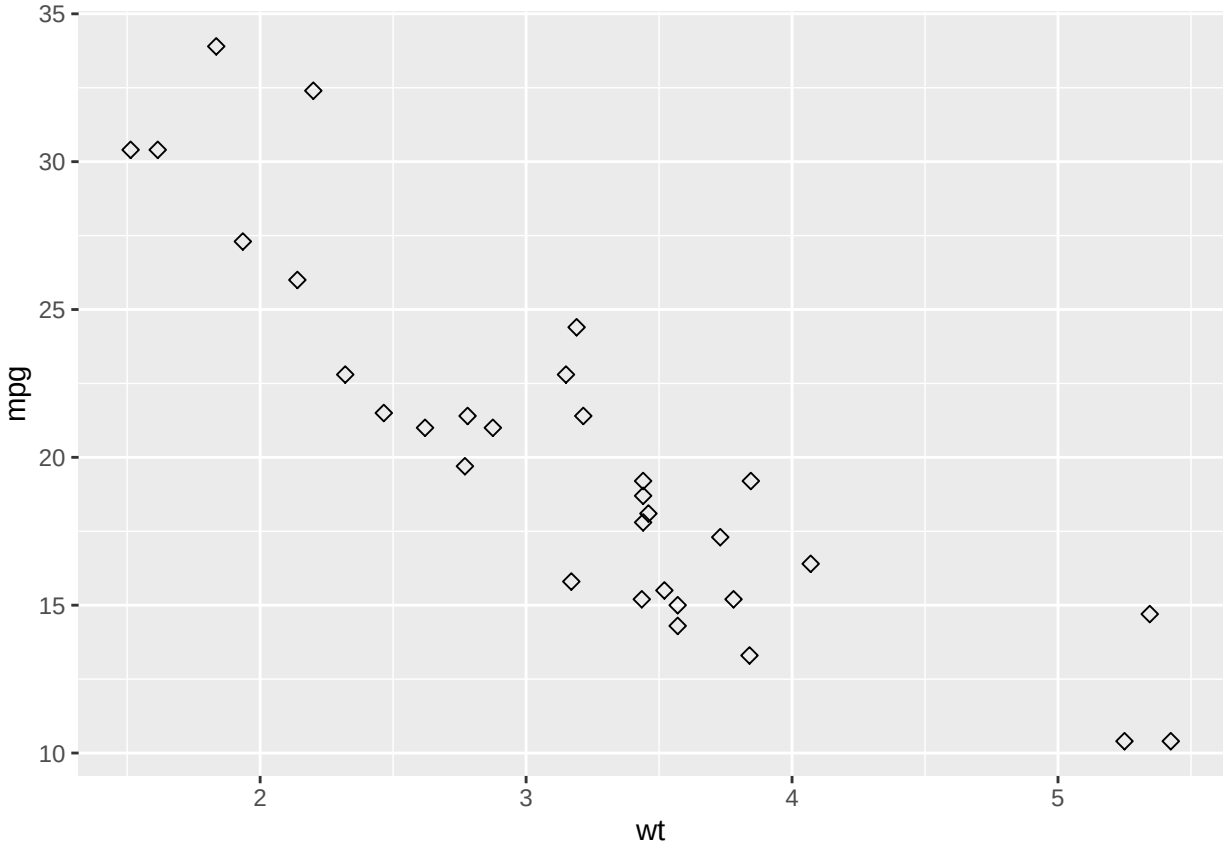
Nuage de points simples

Des nuages de points simples sont créés en utilisant le code de R ci-dessous. La couleur, la taille et la forme des points peuvent être modifiées en utilisant la fonction `geom_point()` comme suit: `geom_point(size, color, shape)`

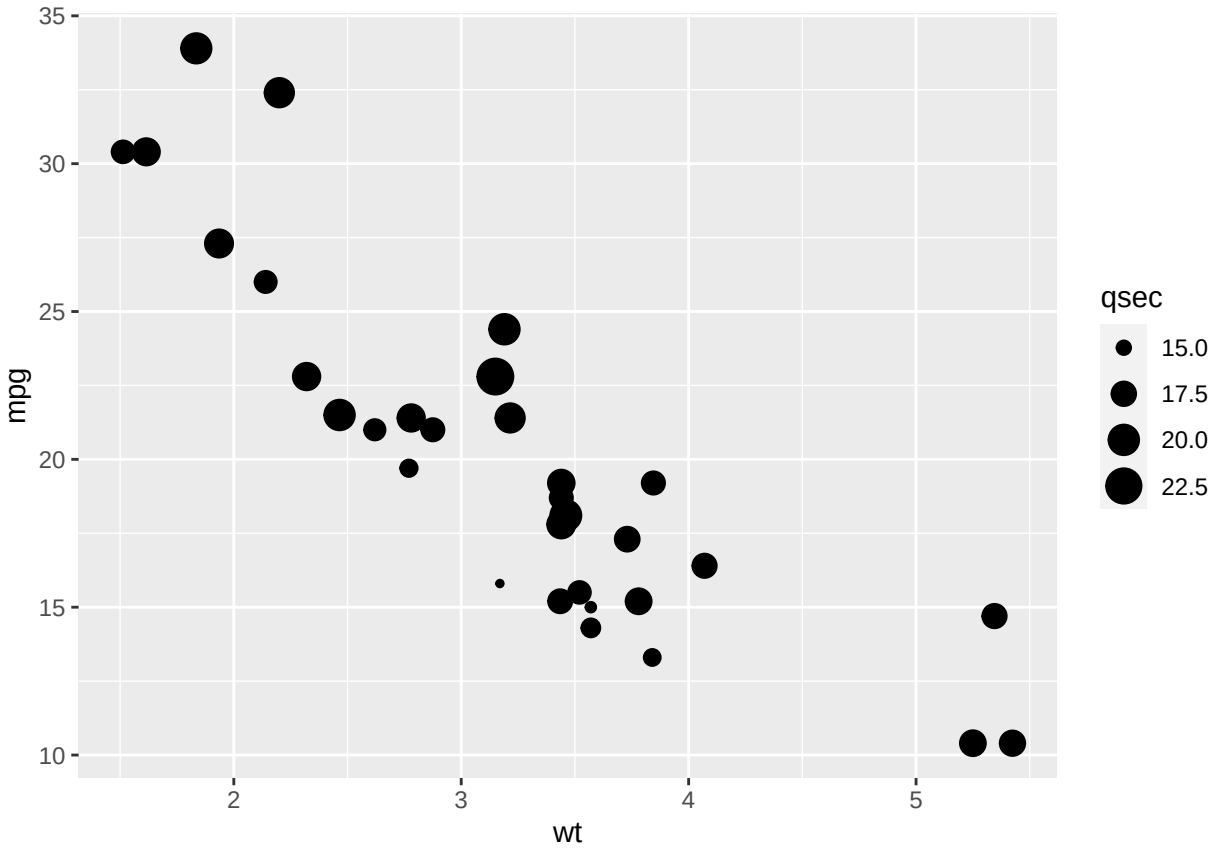
```
#Nuage de points simple
ggplot(mtcars, aes(x=wt, y=mpg)) + geom_point()
```




```
# Changer la taille et la forme
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point(size=2, shape=23)
```



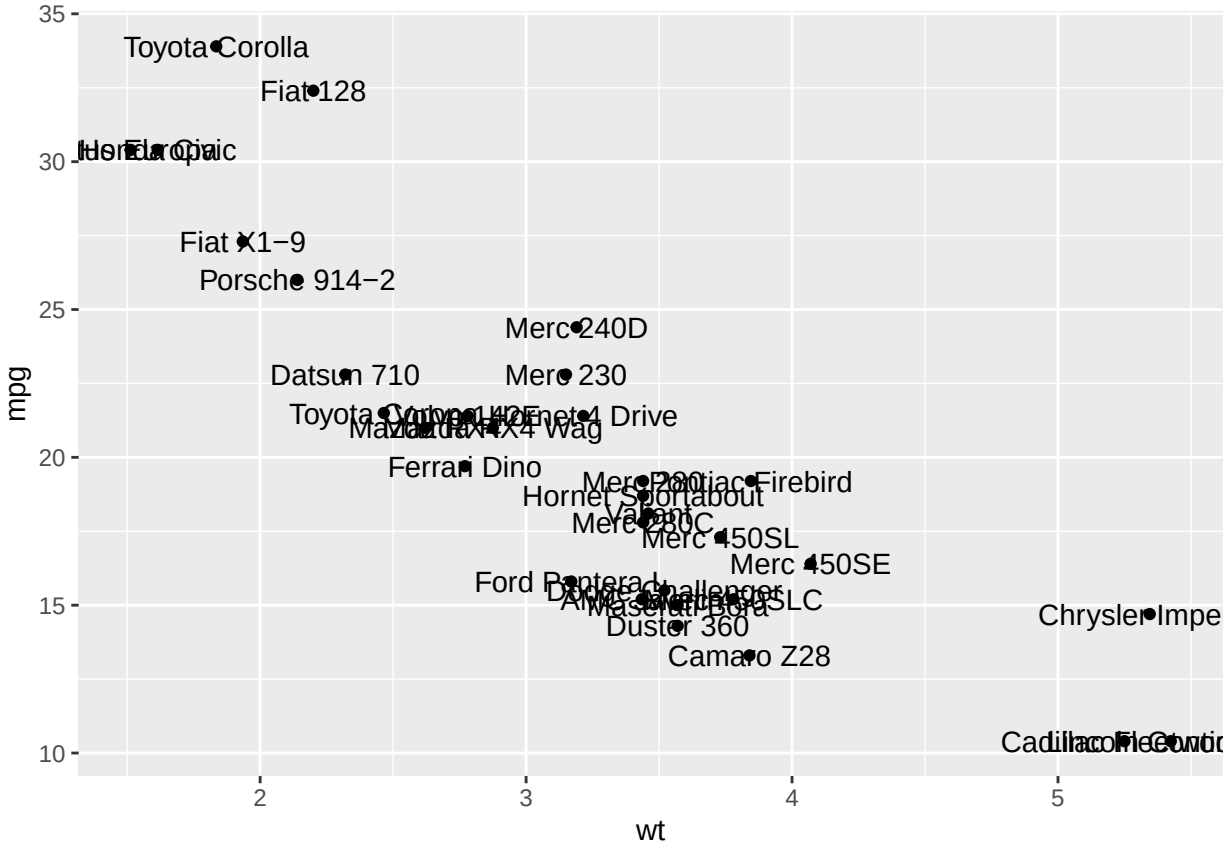

```
# Changer la taille des points
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point(aes(size=qsec))
```



Annoter les points

La fonction `geom_text()` peut être utilisée :

```
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point() +
  geom_text(label=rownames(mtcars))
```



Ajouter des lignes de regression

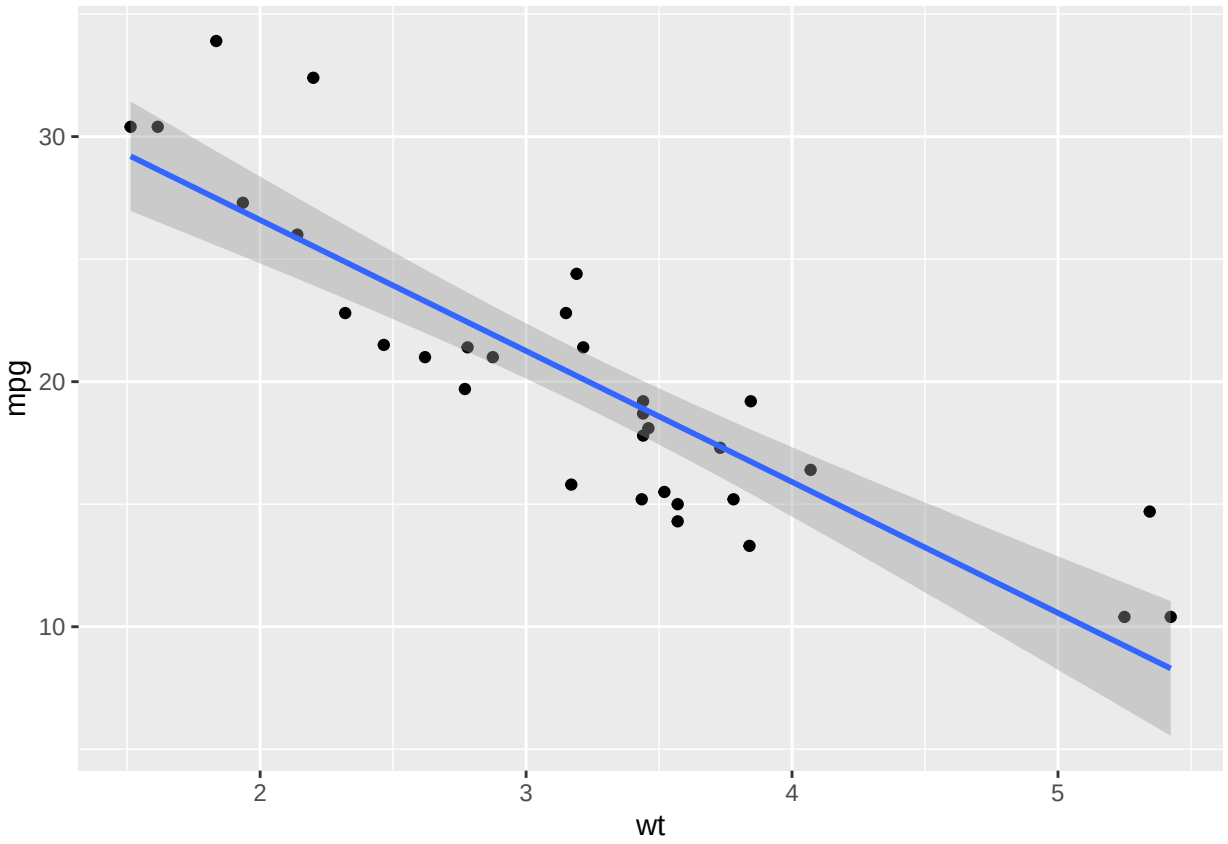
Les fonctions ci-dessous peuvent être utilisées pour ajouter des droites de régression à un nuage de points: * `geom_smooth()` et `stat_smooth` * `geom_abline()` Seule la fonction `geom_smooth()` est couverte dans cette section.

Un format simplifié est: `geom_smooth(method="auto", se=TRUE, fullrange=FALSE, level=0.95)`

- `method` : méthode à utiliser pour estimer la tendance moyenne. Les valeurs possibles sont lm, glm, gam, loess, rlm.
- `se` : valeur logique. Si TRUE, l'intervalle de confiance est affichée autour de la moyenne.
- `fullrange` : valeur logique. Si TRUE, la courbe moyenne couvre le graphique en entier.
- `level` : niveau de l'intervalle de confiance à utiliser. La valeur par défaut est de 0,95.

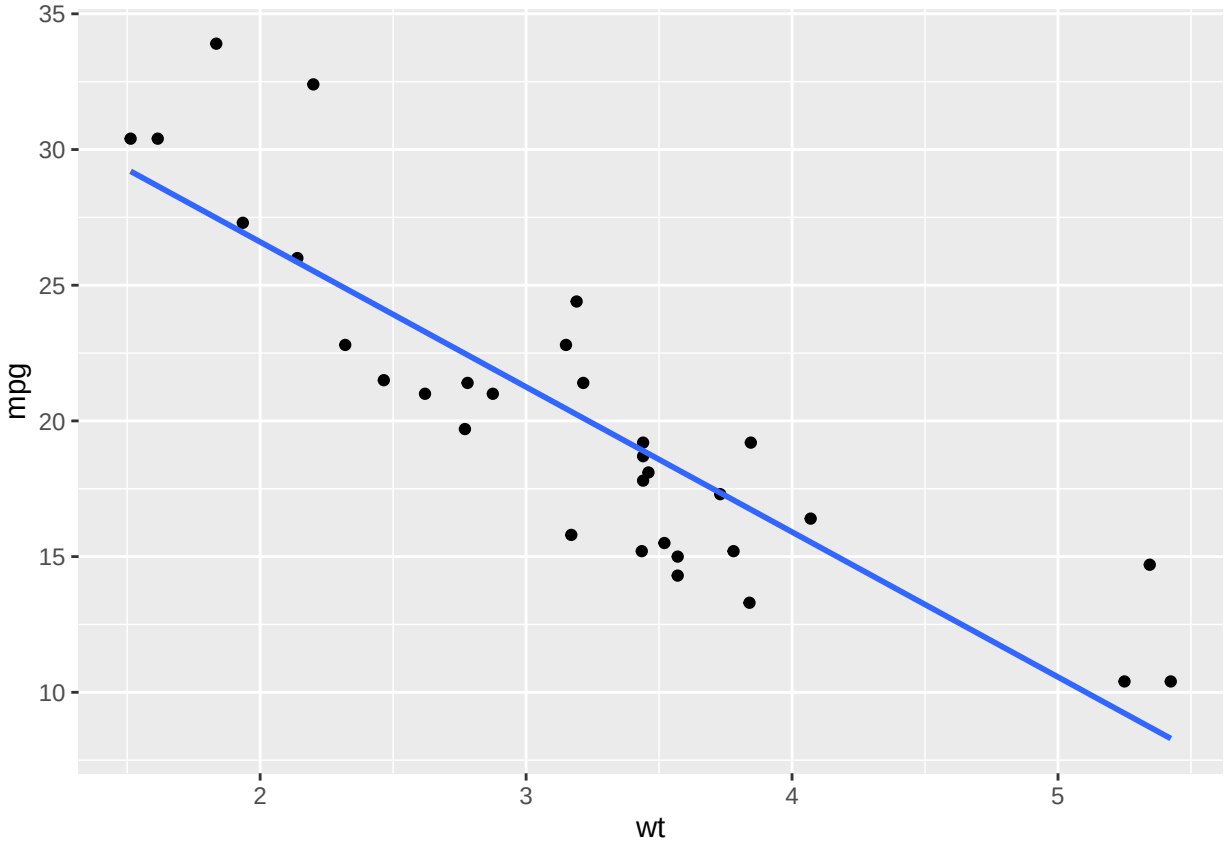
```
# Ajouter la droite de regression
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point()+
  geom_smooth(method=lm)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



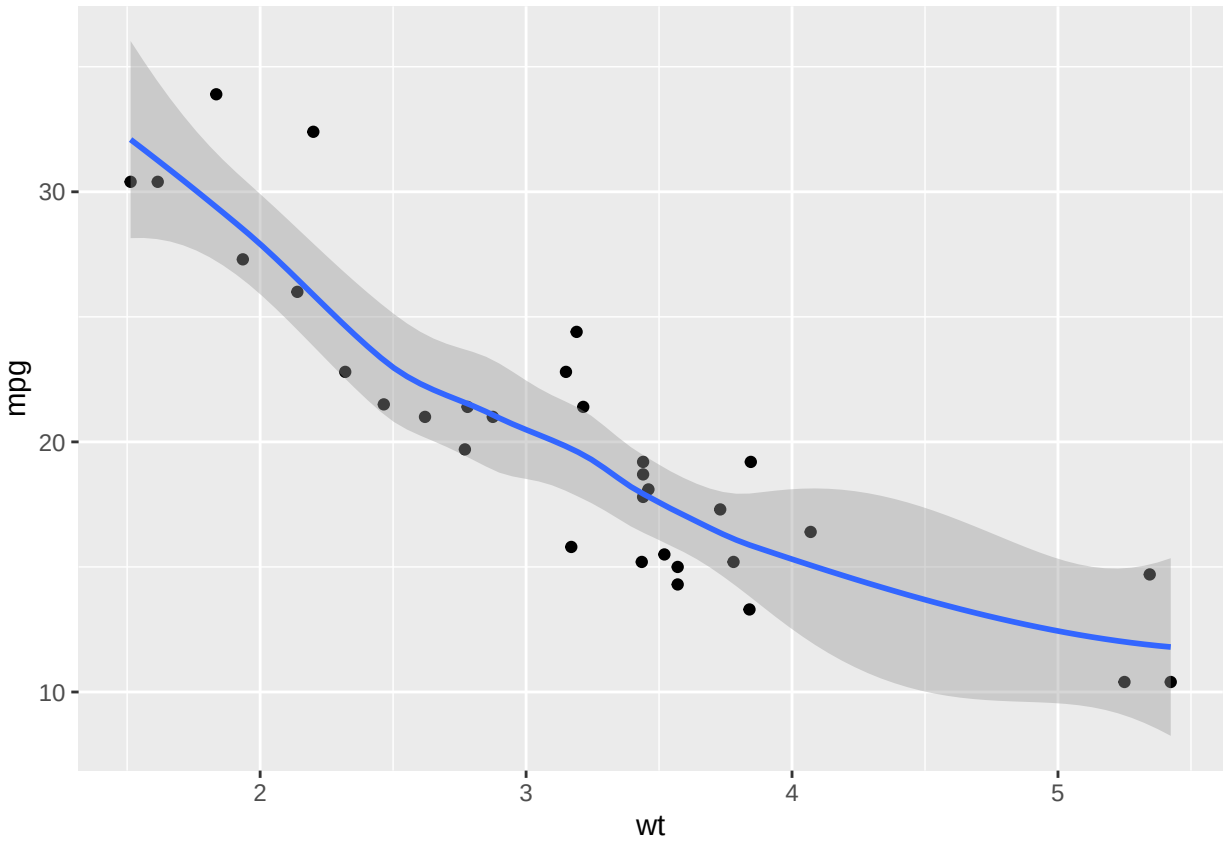

```
# Supprimer l'intervalle de confiance
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point()+
  geom_smooth(method=lm, se=FALSE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
# La méthode "Loess"
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point()+
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

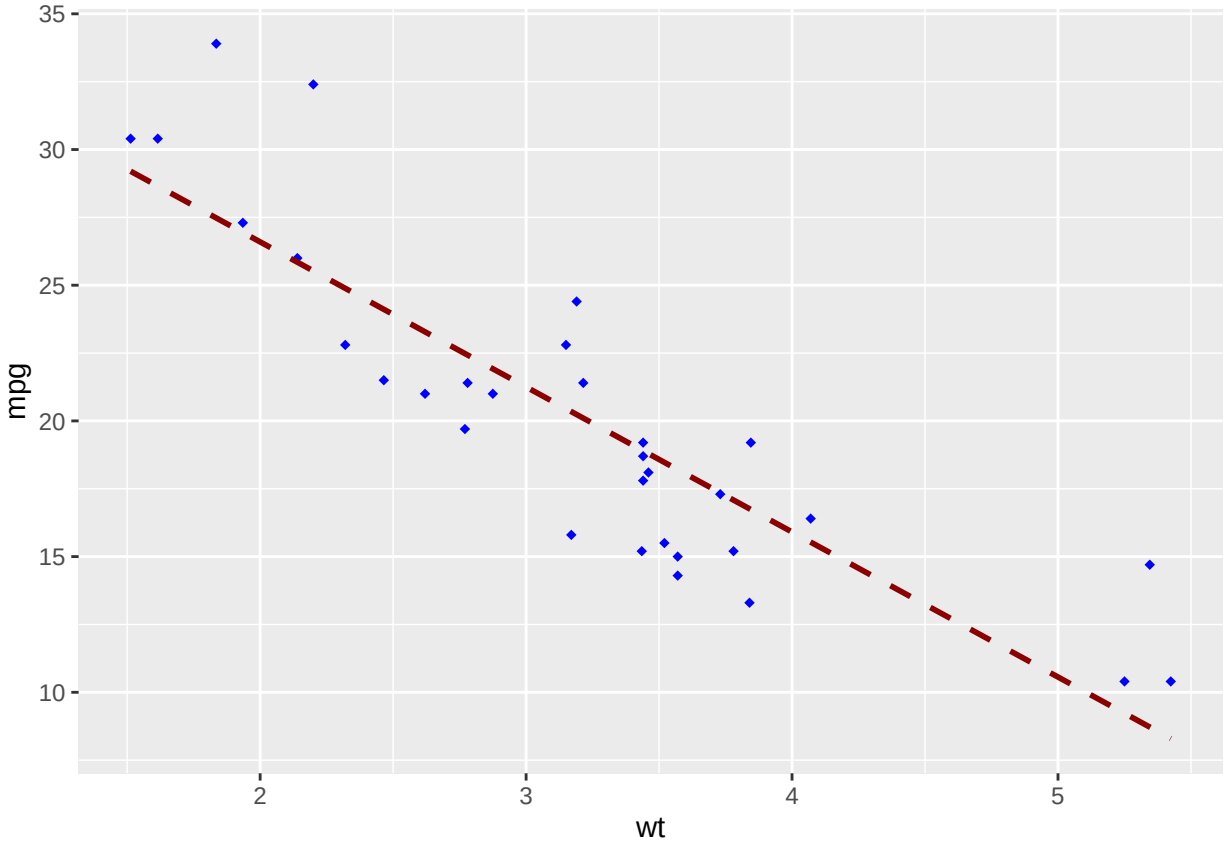


Changer l'apparence des points et des traits

Cette section décrit comment modifier: *la couleur et la forme des points* le type de trait et la couleur de la droite de régression *la couleur de remplissage de l'intervalle de confiance

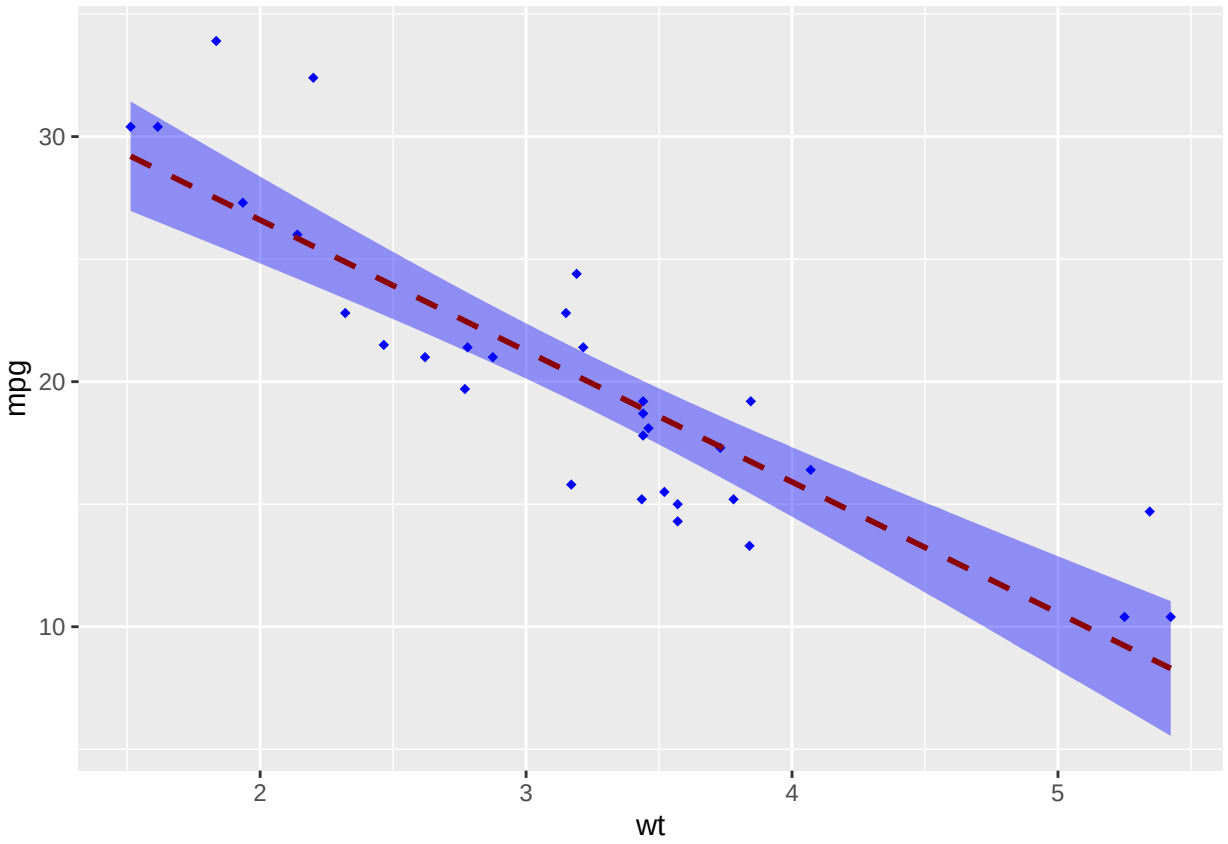
```
# Changer la couleur et la forme des points
# Changer le type de trait et la couleur
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point(shape=18, color="blue")+
  geom_smooth(method=lm, se=FALSE, linetype="dashed",
              color="darkred")
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
# Changer la couleur de remplissage de l'intervalle de confiance
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point(shape=18, color="blue")+
  geom_smooth(method=lm,  linetype="dashed",
              color="darkred", fill="blue")
```

```
## `geom_smooth()` using formula 'y ~ x'
```

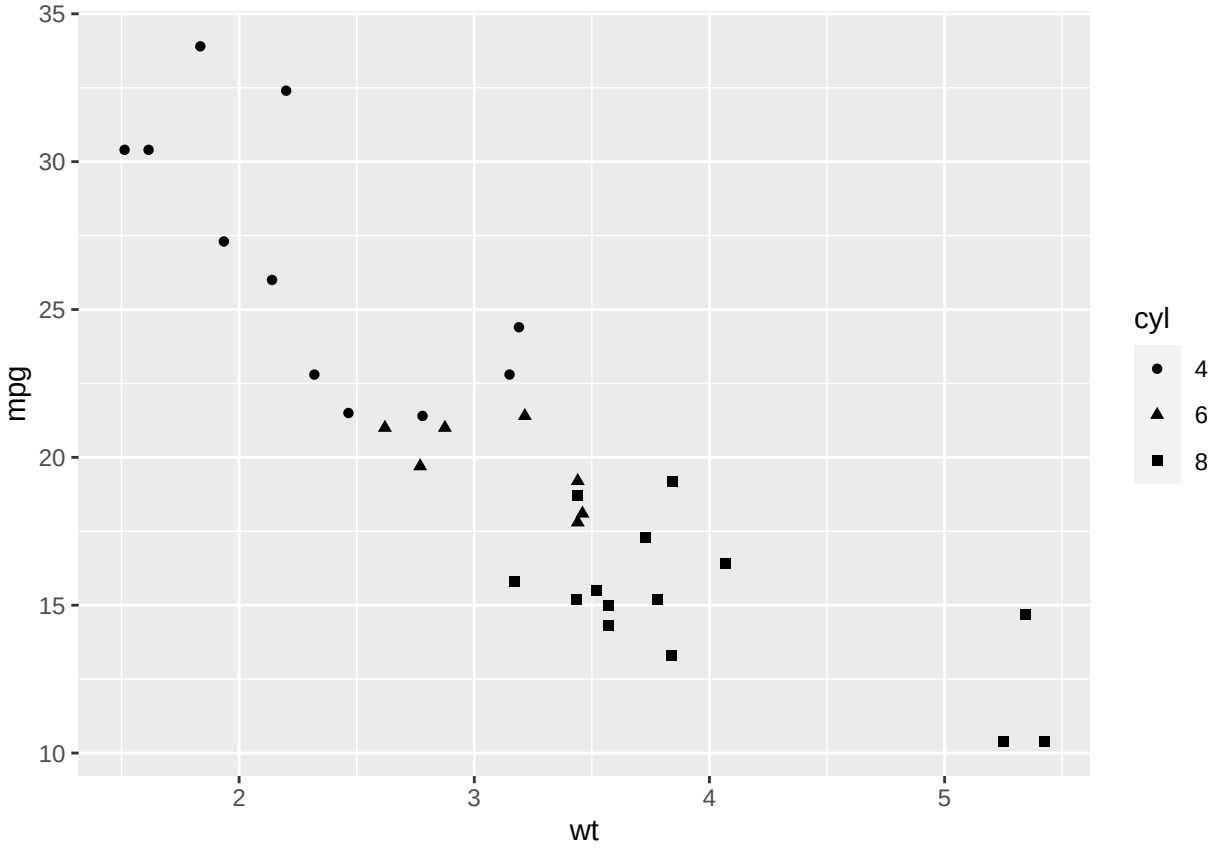


Nuage de points avec plusieurs groupes

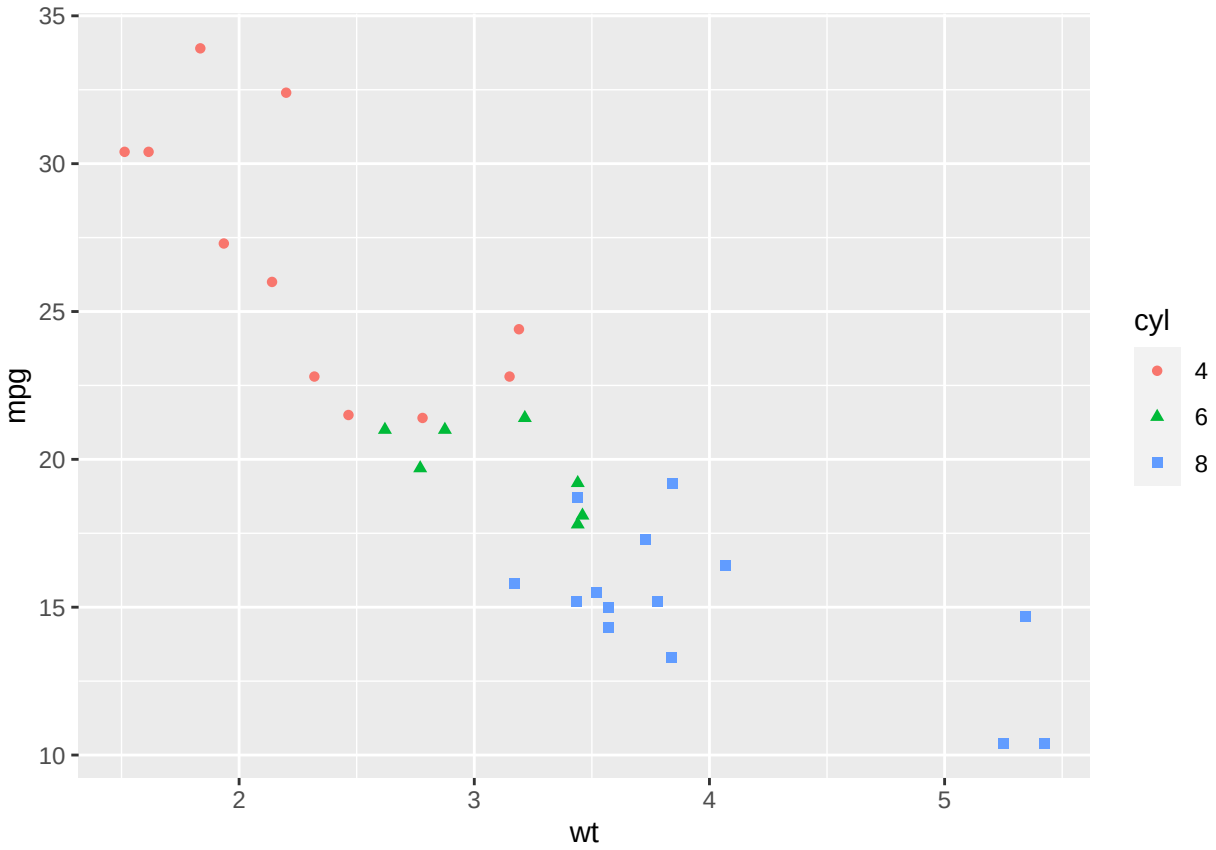
Cette section décrit comment changer les couleurs et les types de points automatiquement et manuellement.

Changer la couleur, le type, la taille des points automatiquement

```
# Changer le type de points en fonction des niveaux de cyl
ggplot(mtcars, aes(x=wt, y=mpg, shape=cyl)) +
  geom_point()
```

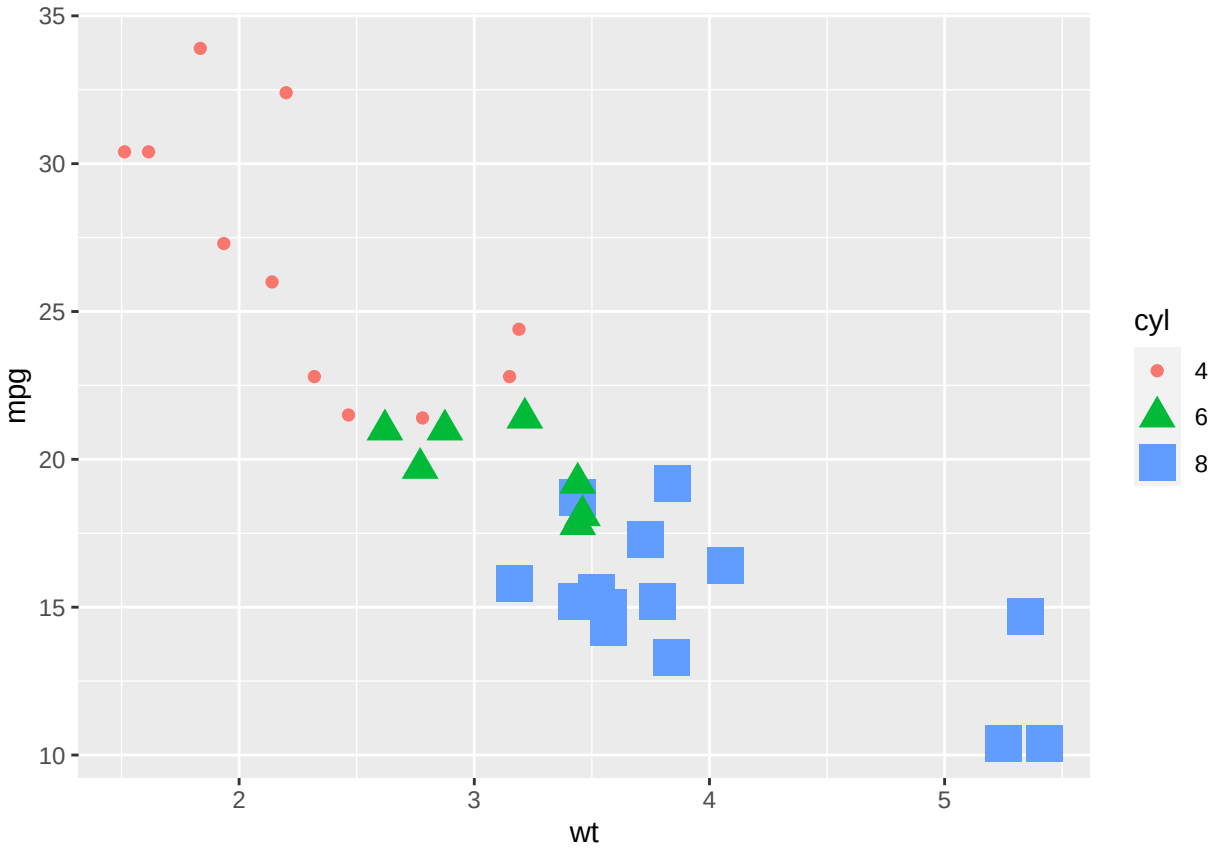



```
# Changer le type et la couleur
ggplot(mtcars, aes(x=wt, y=mpg, shape=cyl, color=cyl)) +
  geom_point()
```




```
# Changer le type, la couleur et la taille
ggplot(mtcars, aes(x=wt, y=mpg, shape=cyl, color=cyl, size=cyl)) +
  geom_point()
```

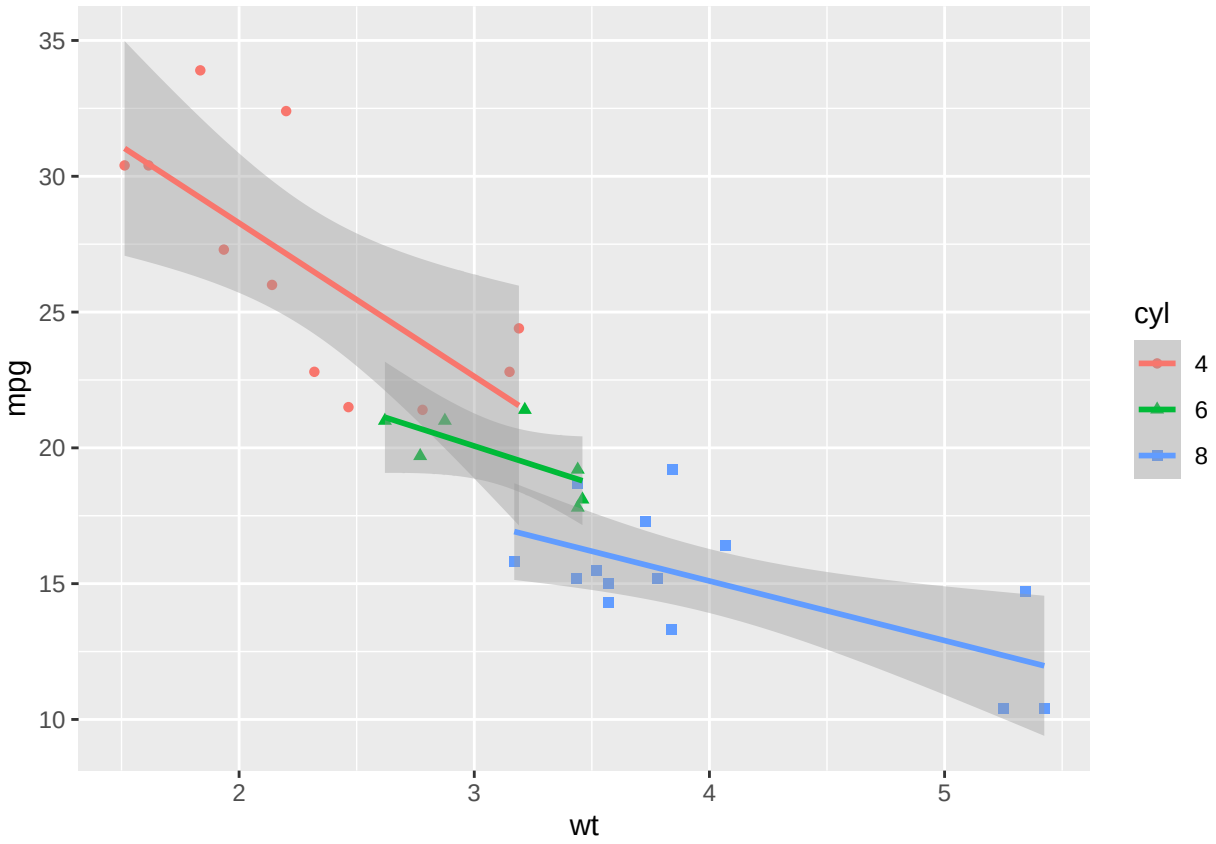
```
## Warning: Using size for a discrete variable is not advised.
```



Ajouter des droites de régression

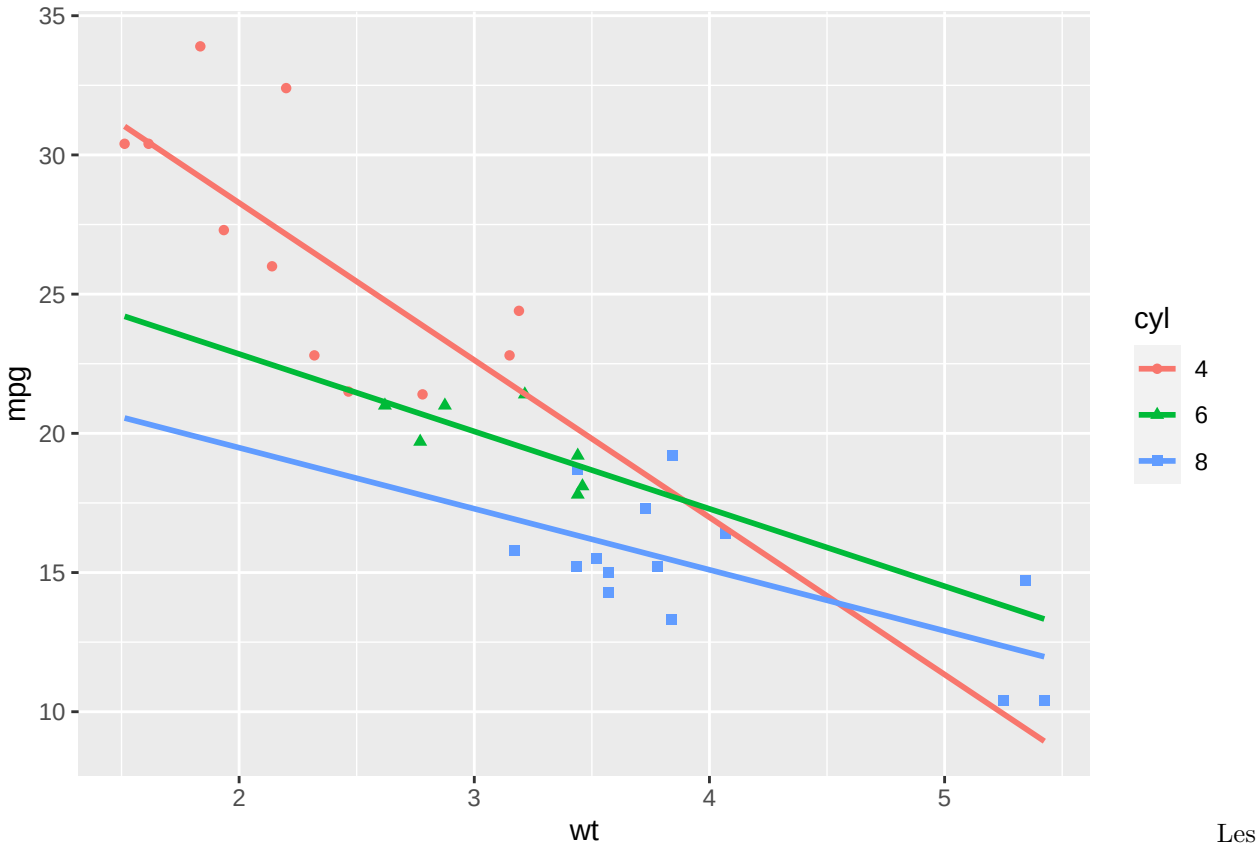
```
# Ajouter des lignes de régression
ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
  geom_point() +
  geom_smooth(method=lm)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
# Supprimer les intervalles de confiance
# Etendre les droites de régression
ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
  geom_point() +
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE)
```

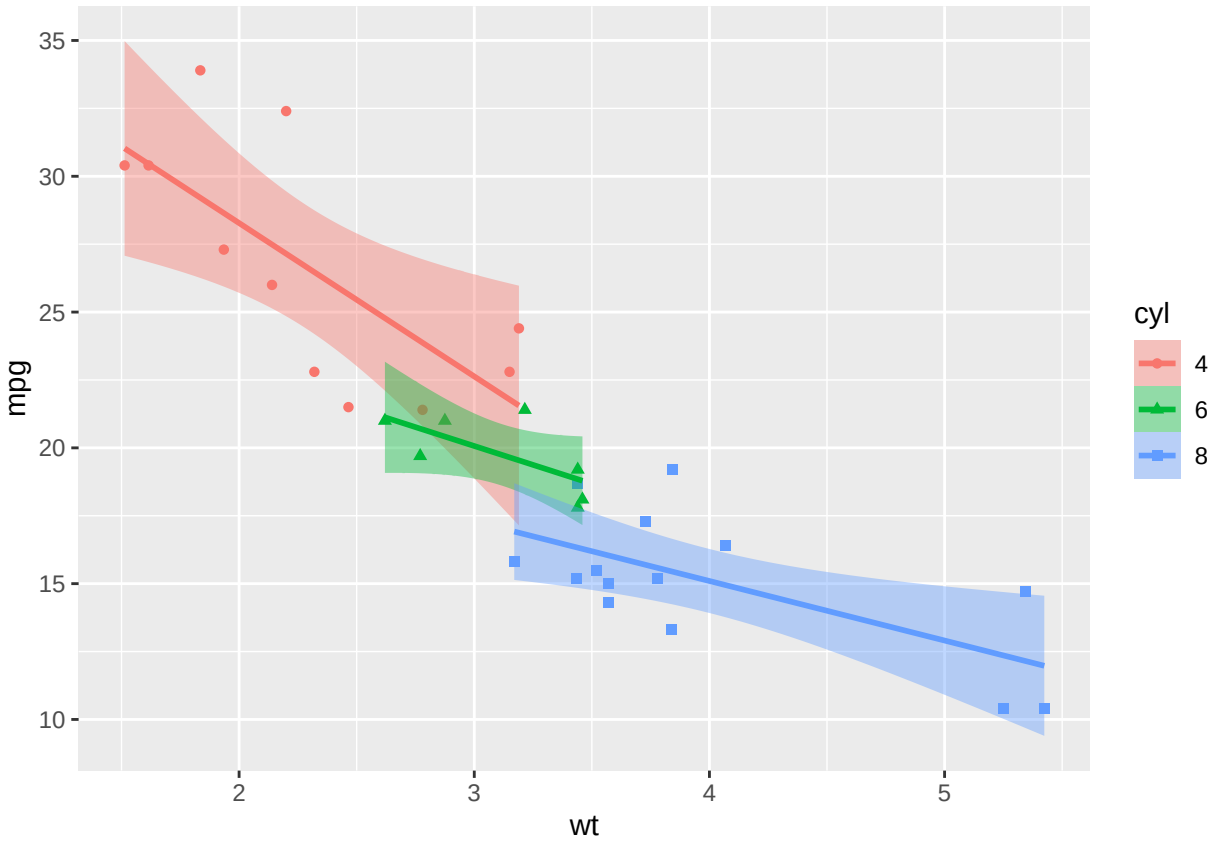
```
## `geom_smooth()` using formula 'y ~ x'
```



Les couleurs de remplissage des intervalles de confiances peuvent être changées comme suit:

```
ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
  geom_point() +
  geom_smooth(method=lm, aes(fill=cyl))
```

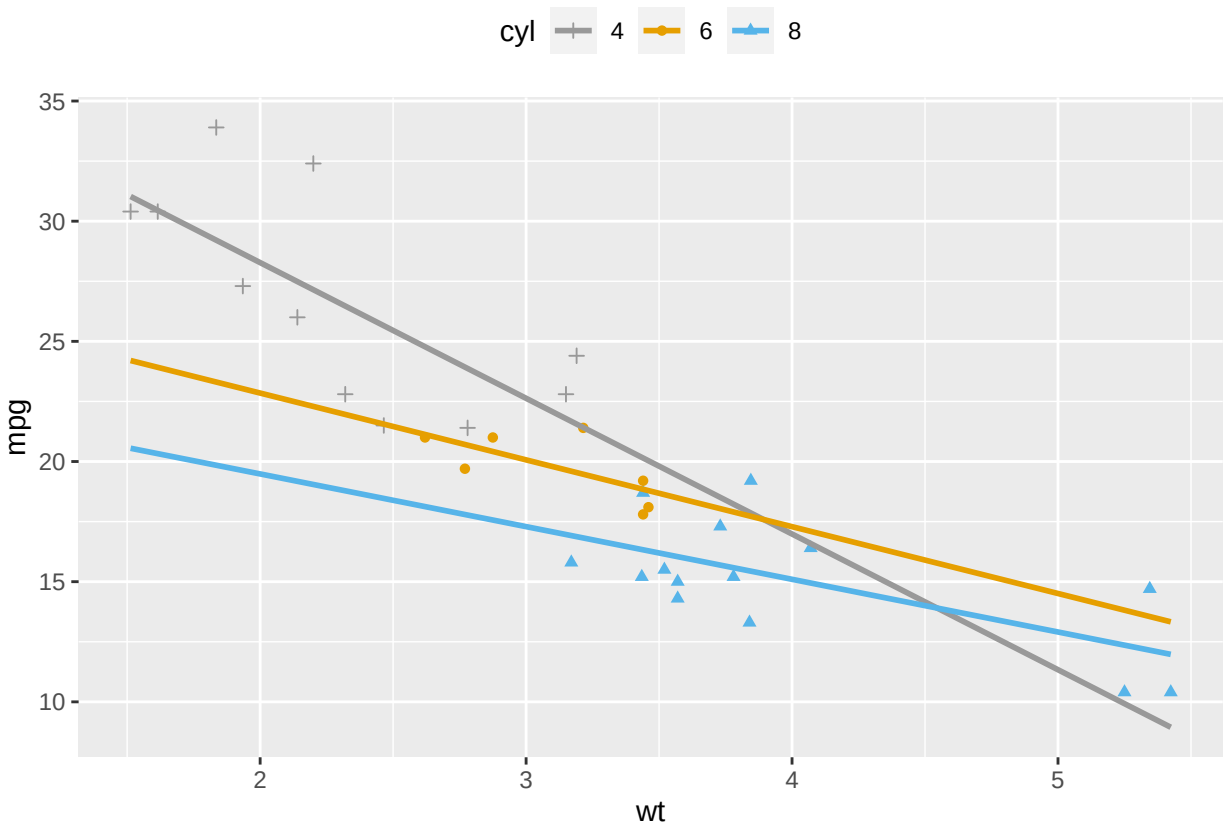
```
## `geom_smooth()` using formula 'y ~ x'
```



Changer la couleur, le type, la taille des points manuellement

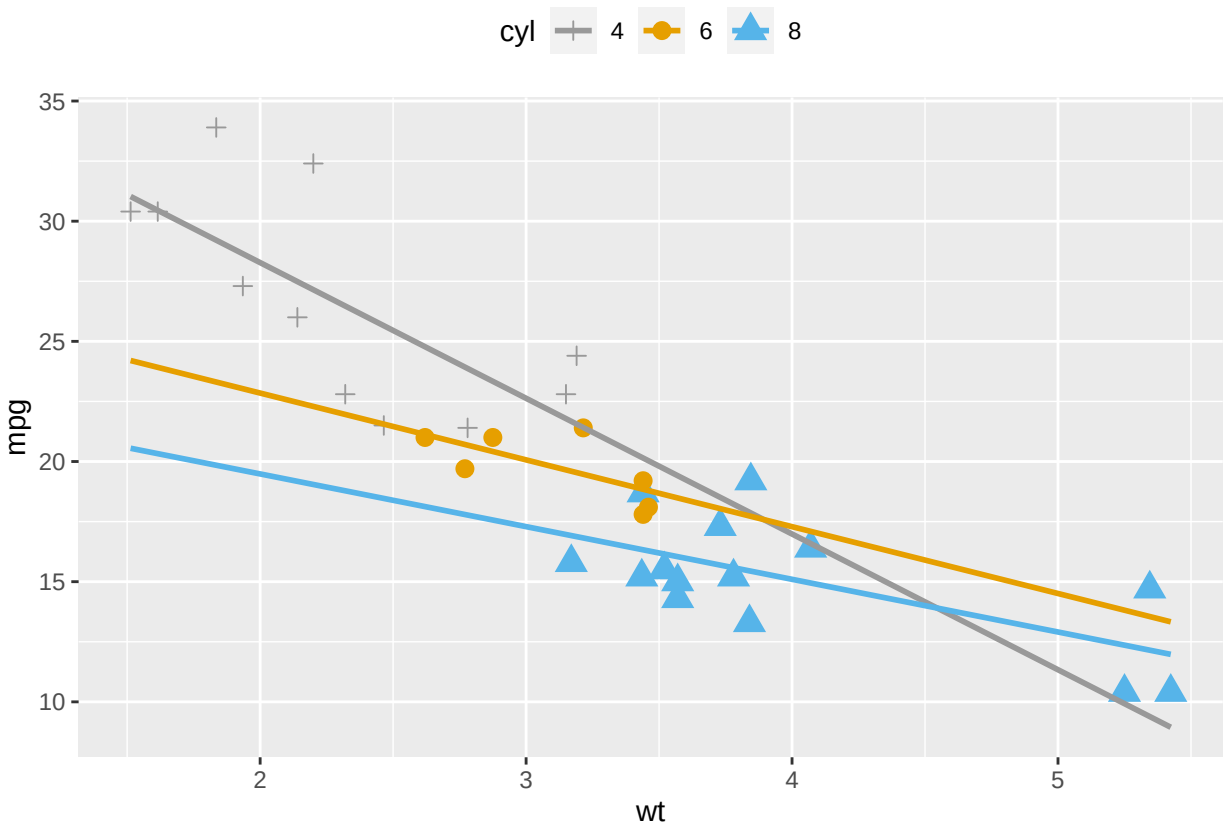
```
# Changer le type de points et la couleur manuellement
ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
  geom_point() +
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE)+
  scale_shape_manual(values=c(3, 16, 17))+
  scale_color_manual(values=c('#999999','#E69F00', '#56B4E9'))+
  theme(legend.position="top")
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
# Changer la taille des points manuellement
ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl))+
  geom_point(aes(size=cyl)) +
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE)+
  scale_shape_manual(values=c(3, 16, 17))+
  scale_color_manual(values=c('#999999','#E69F00', '#56B4E9'))+
  scale_size_manual(values=c(2,3,4))+
  theme(legend.position="top")
```

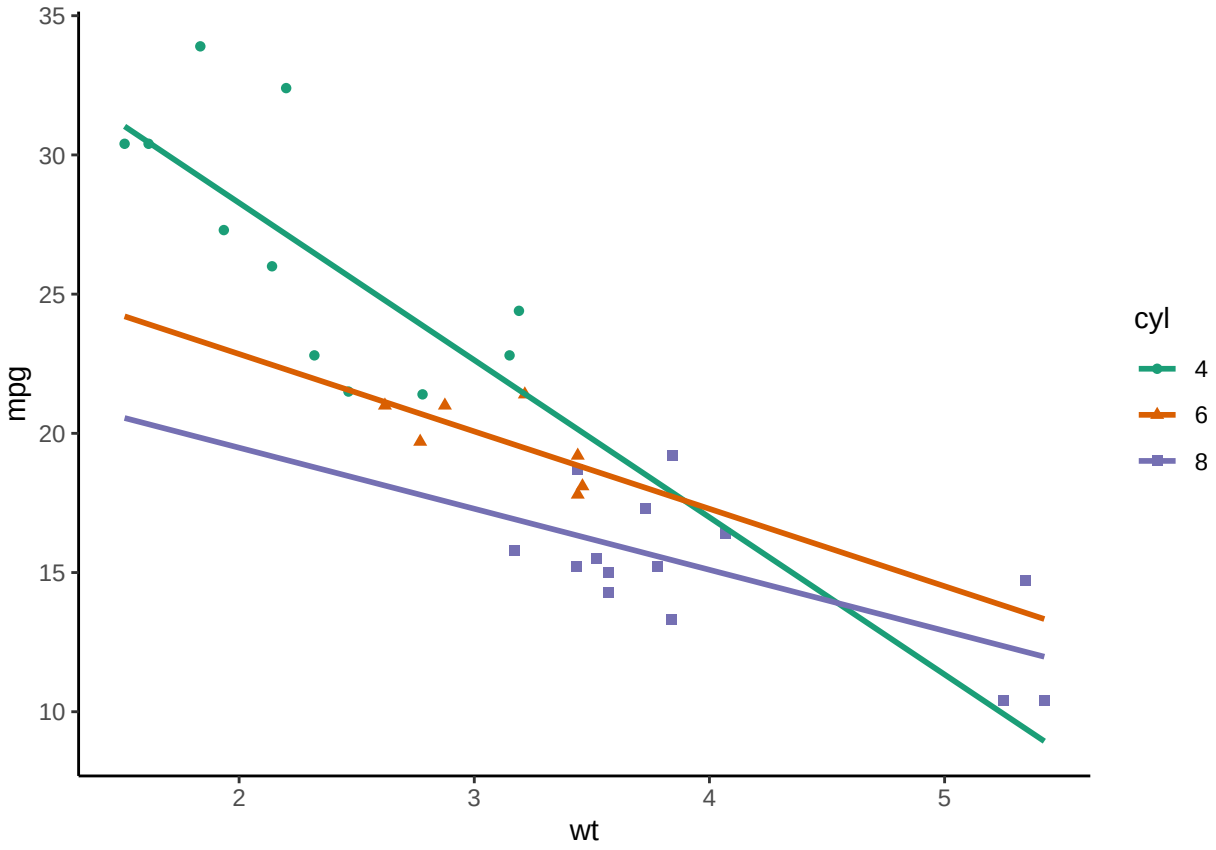
```
## `geom_smooth()` using formula 'y ~ x'
```



Il est également possible de modifier manuellement la couleur des points et des traits en utilisant les fonctions: * `scale_color_brewer()` : pour utiliser les palettes de couleurs du package RColorBrewer * `scale_color_grey()` : pour utiliser les palettes de couleurs grises

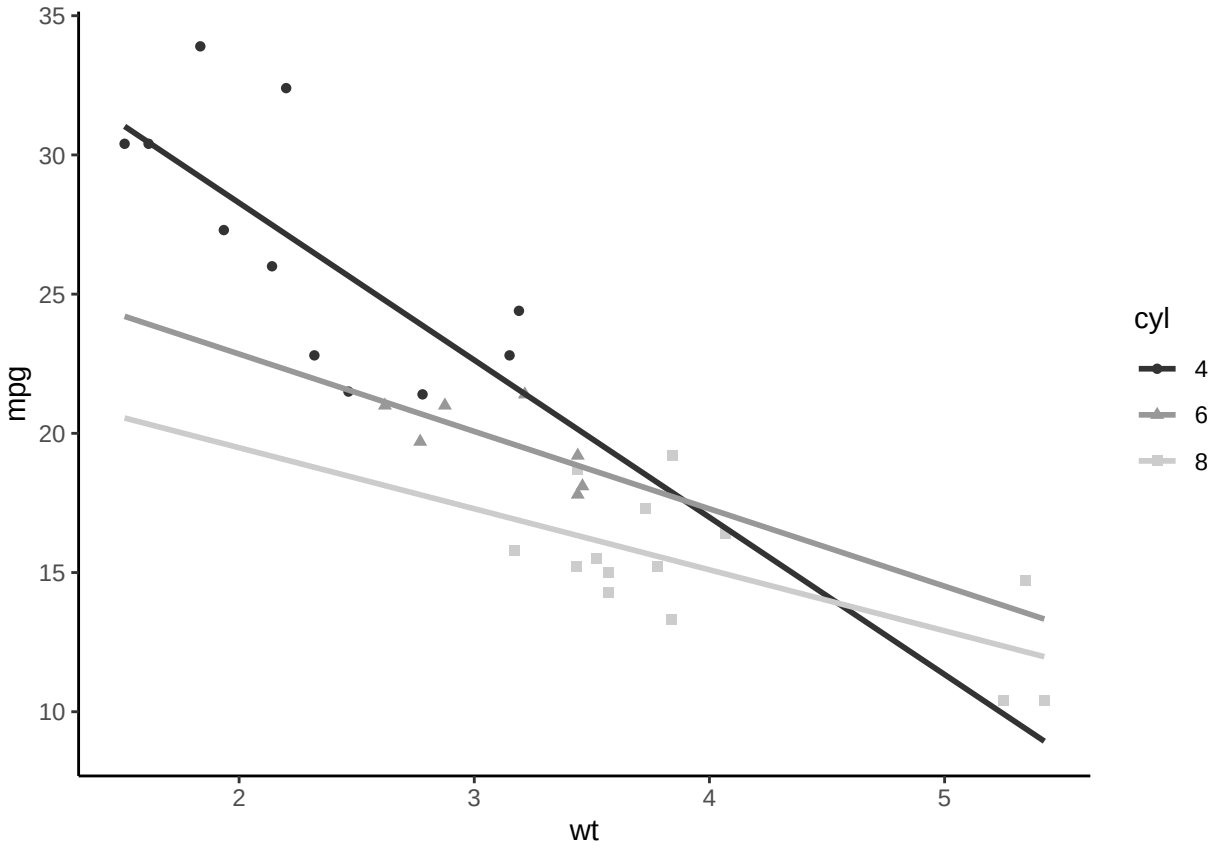
```
p <- ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
  geom_point() +
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE)+
  theme_classic()
# Utiliser les palettes brewer
p+scale_color_brewer(palette="Dark2")
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
# Utiliser les couleurs grises
p + scale_color_grey()
```

```
## `geom_smooth()` using formula 'y ~ x'
```

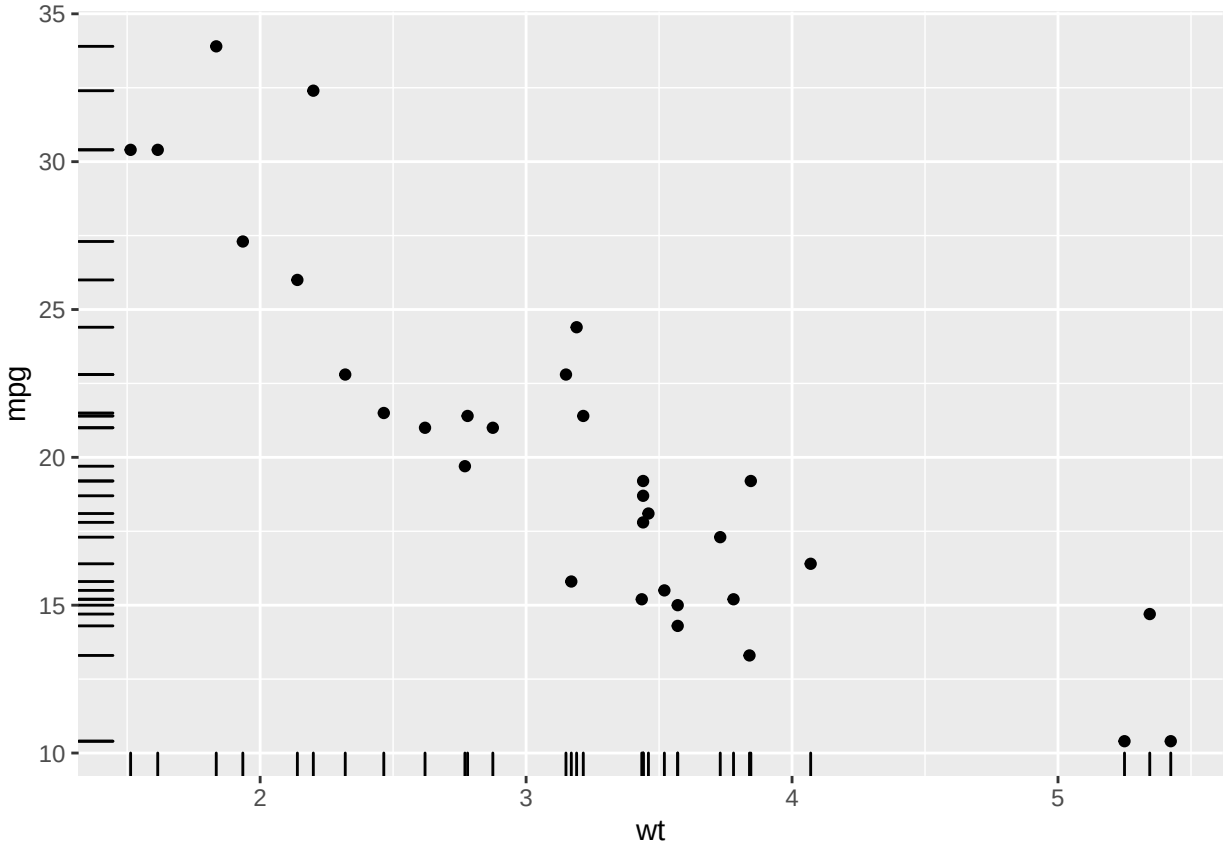


Ajouter la densité marginale

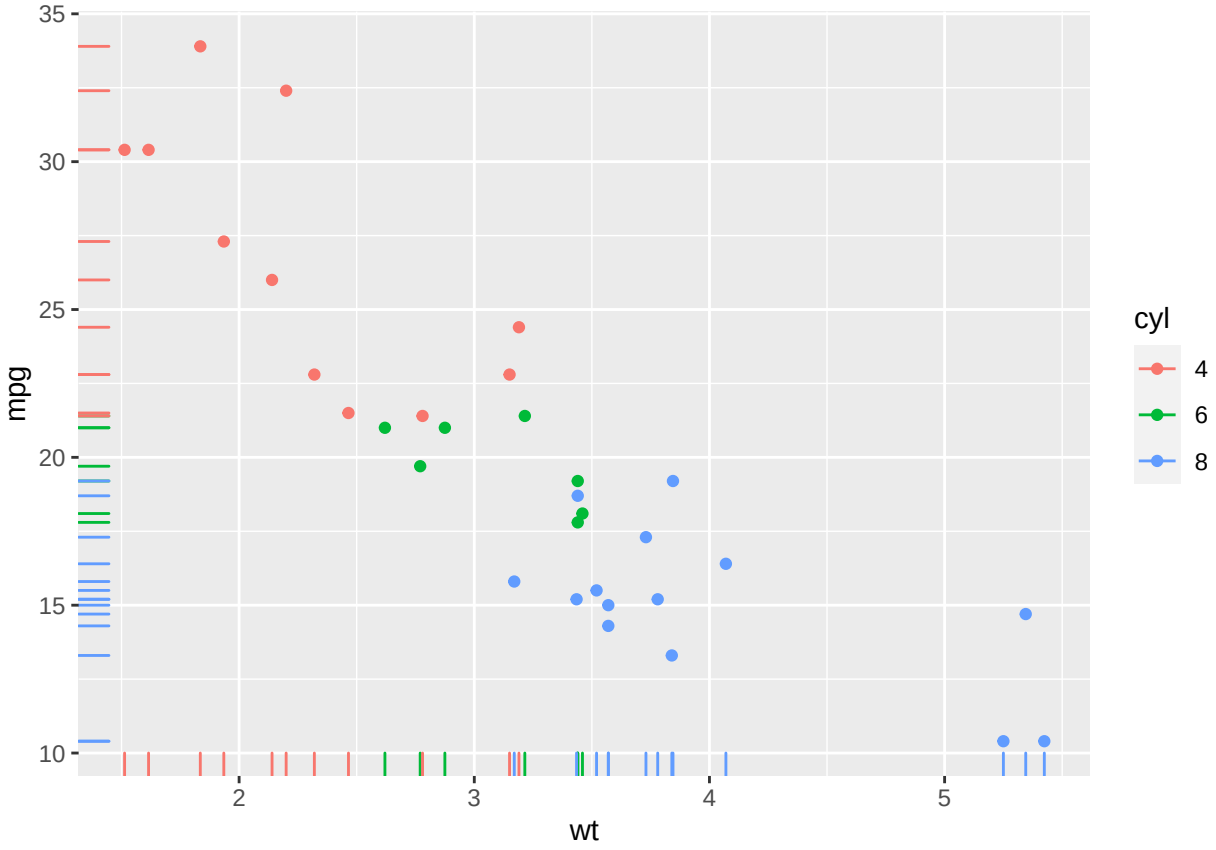
La fonction `geom_rug()` peut être utilisée: `geom_rug(sides ="bl")`

`sides` : côté sur lequel il faudrait ajouter la densité. La valeur possible est une chaîne de caractère contenant l'un des éléments "trbl", pour top (haut), right (droite), bottom (bas), et left (gauche).

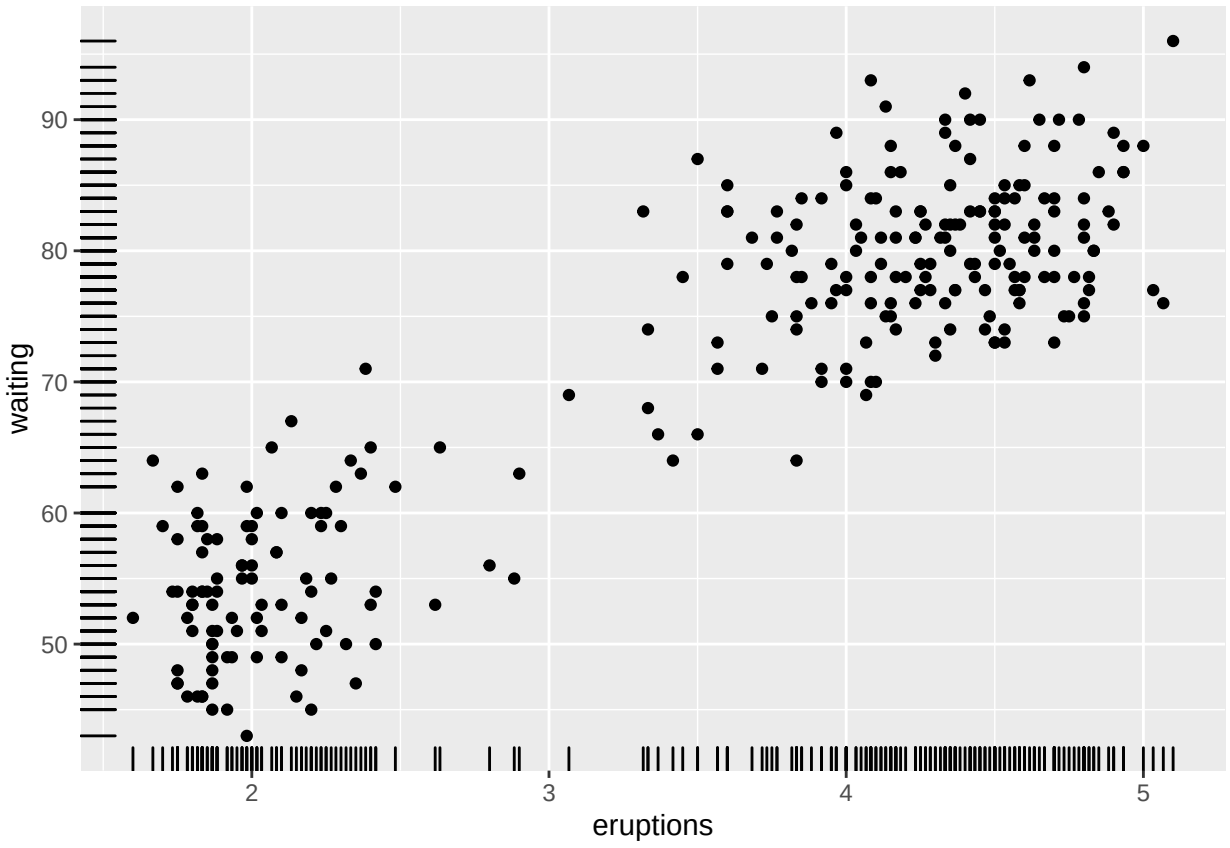
```
# Ajouter la densité marginale
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point() + geom_rug()
```




```
# Changer les couleurs
ggplot(mtcars, aes(x=wt, y=mpg, color=cyl)) +
  geom_point() + geom_rug()
```



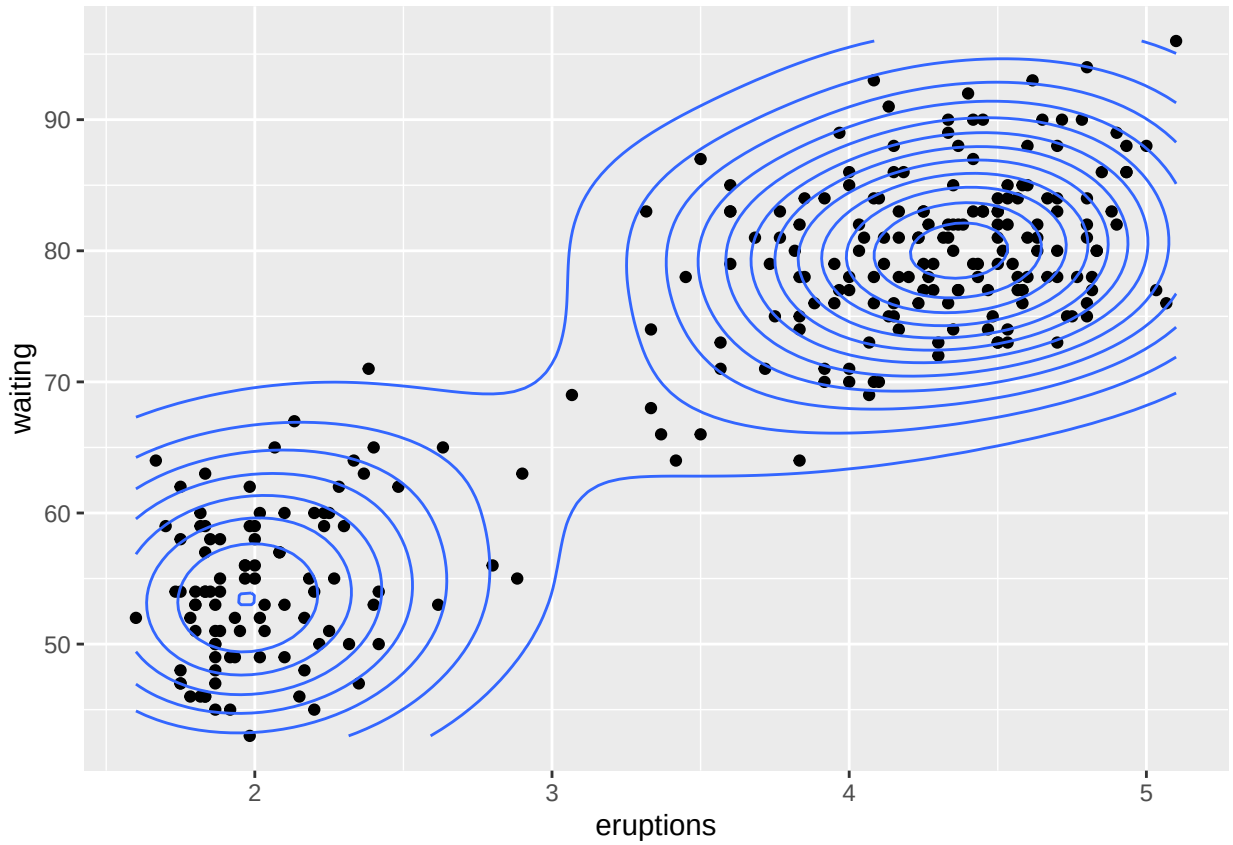

```
# Utiliser le jeu de données faithful
ggplot(faithful, aes(x=eruptions, y=waiting)) +
  geom_point() + geom_rug()
```



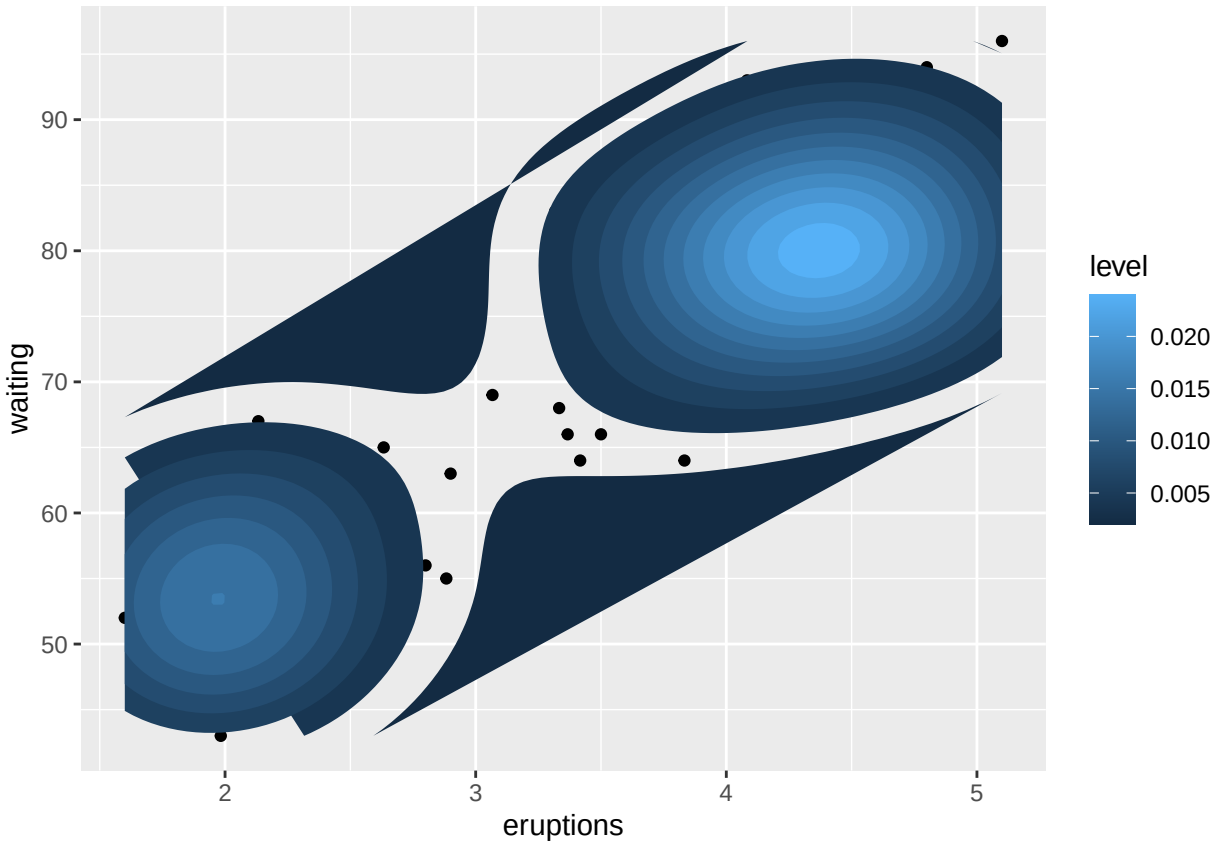
Nuage de pointsavec estimation de la densité 2d

Les fonctions `geom_density2d()` ou `stat_density2d()` peuvent être utilisées:

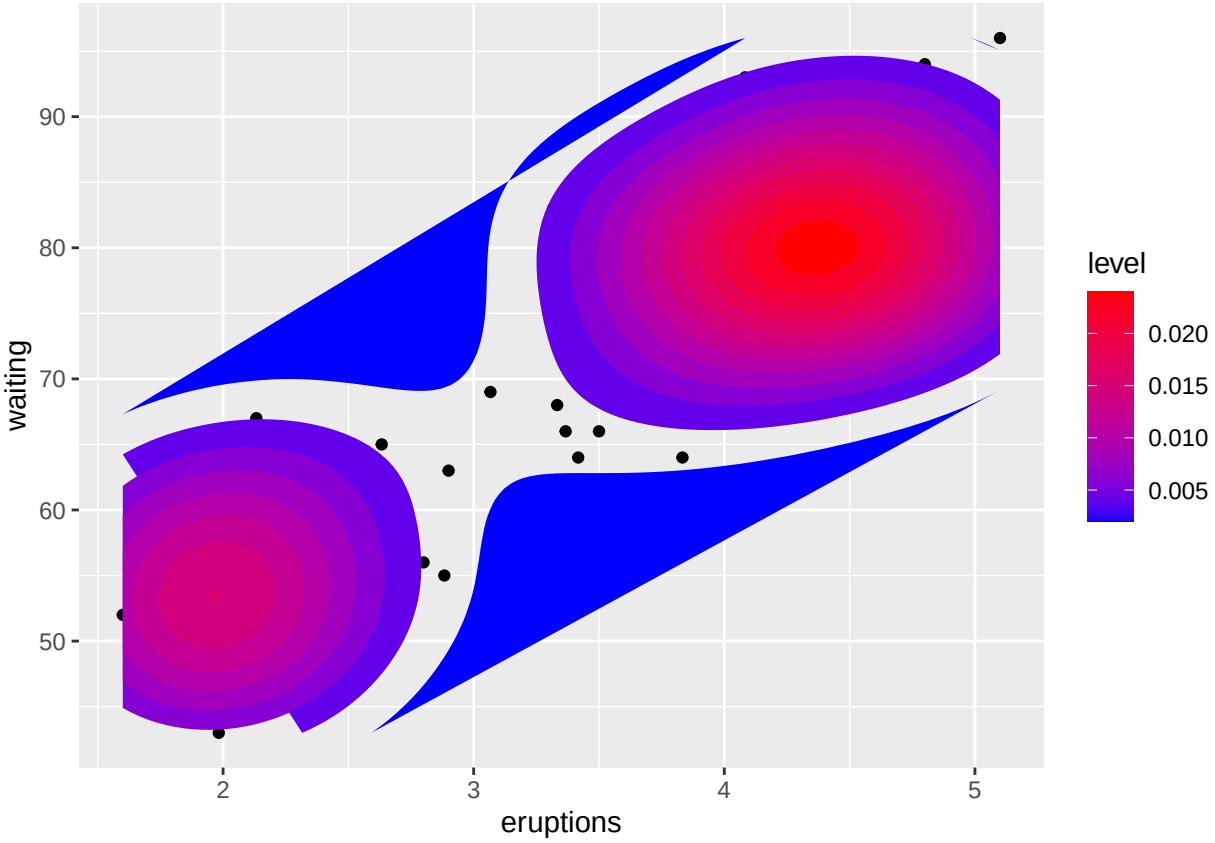
```
# Nuage de points avec estimation de la densité 2d
sp <- ggplot(faithful, aes(x=eruptions, y=waiting)) +
  geom_point()
sp + geom_density2d()
```




```
# Gradient de couleur
sp + stat_density2d(aes(fill = ..level..), geom="polygon")
```



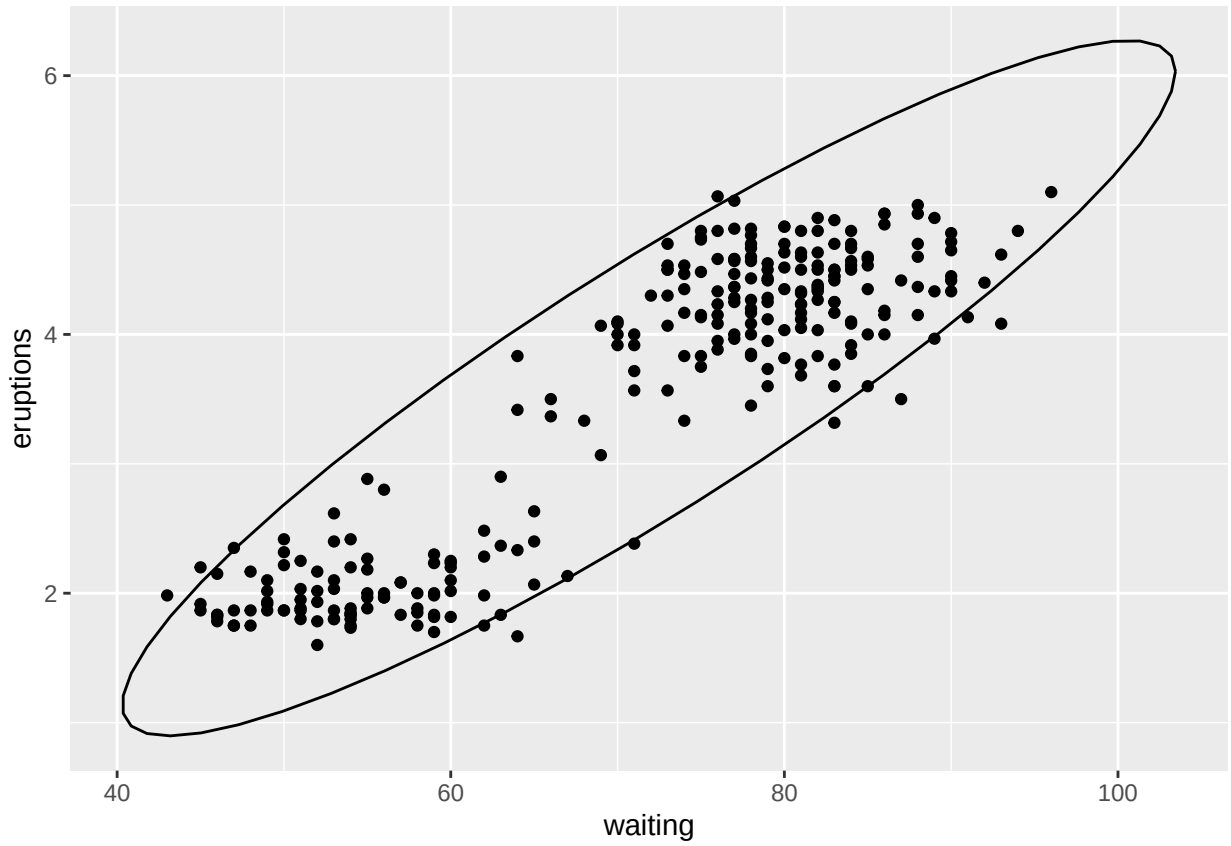

```
# Changer le gradient de couleur
sp + stat_density2d(aes(fill = ..level..), geom="polygon")+
  scale_fill_gradient(low="blue", high="red")
```



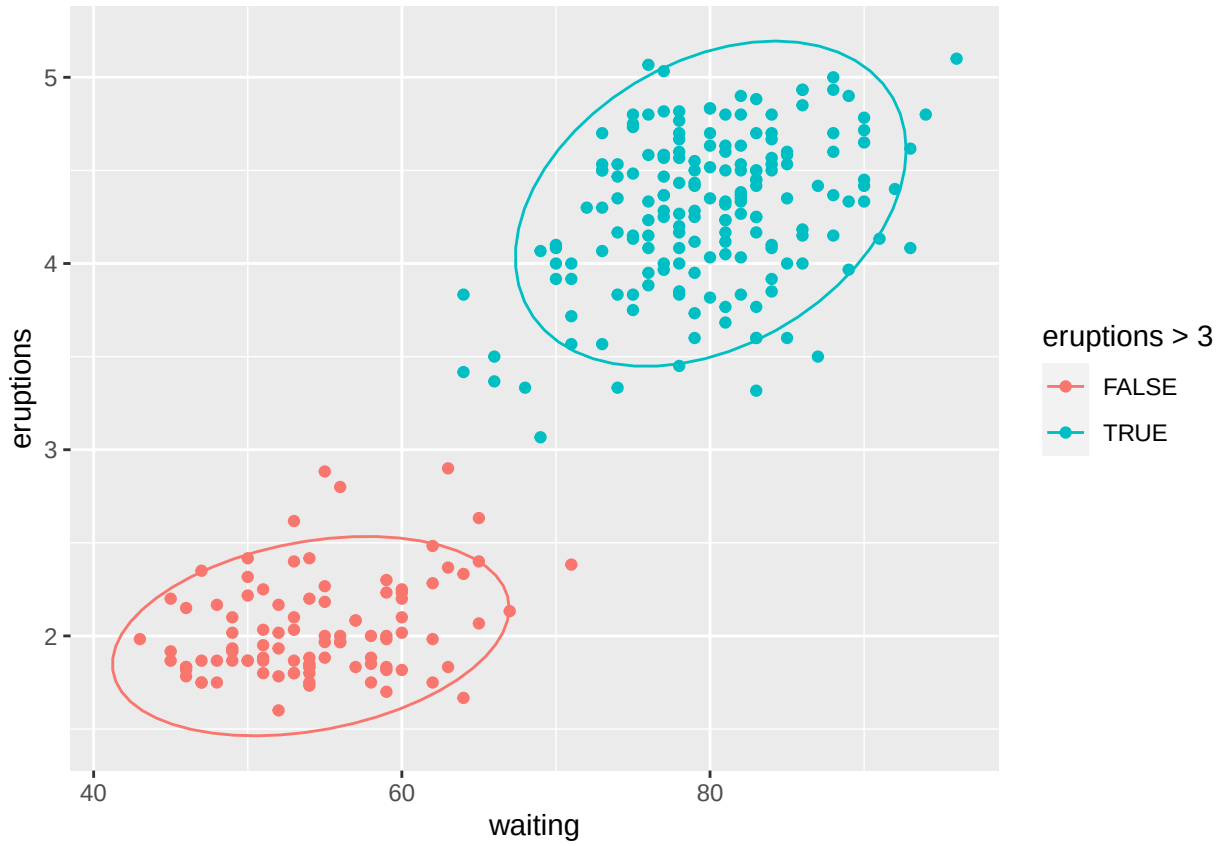
Nuage de points avec ellipse

La fonction `stat_ellipse()` peut être utilisée comme suit:

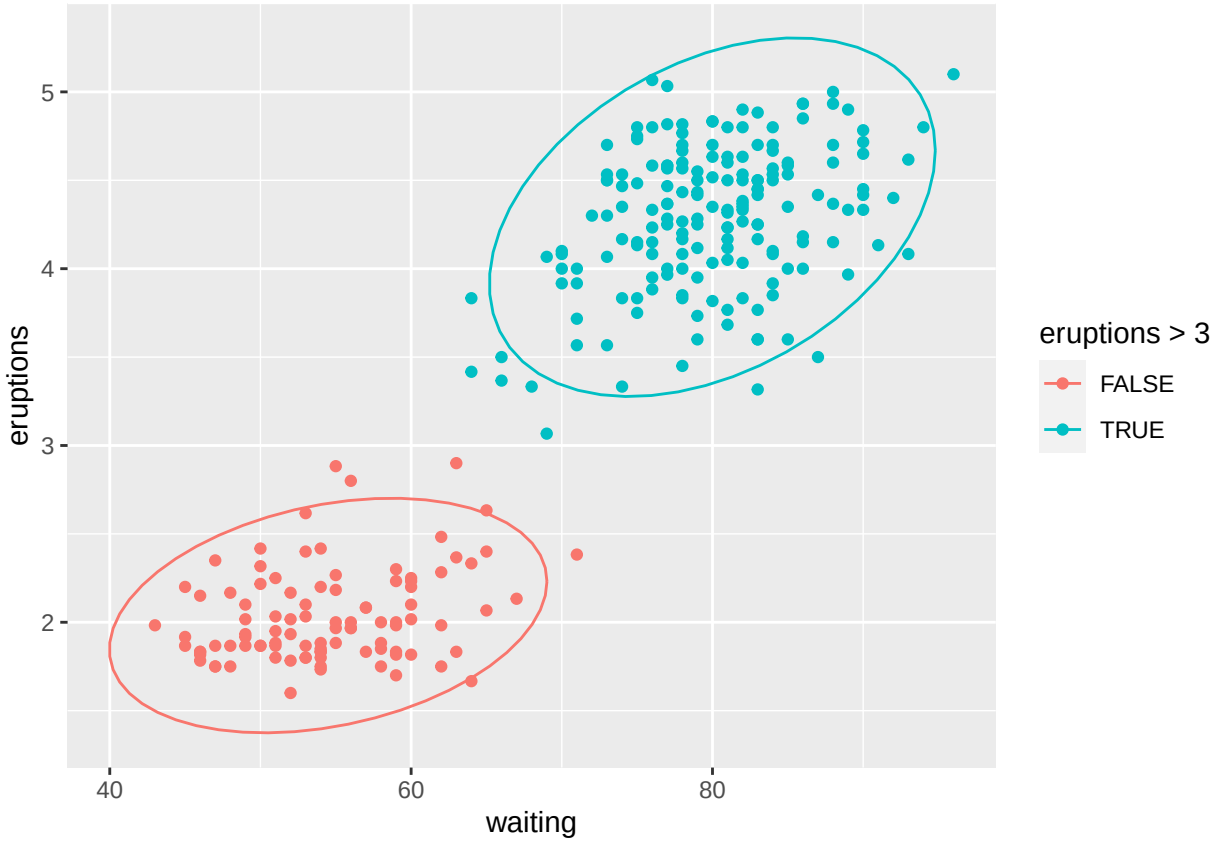
```
# Une ellipse autour de tous les points
ggplot(faithful, aes(waiting, eruptions))+
  geom_point()+
  stat_ellipse()
```




```
# Ellipses par groupes
p <- ggplot(faithful, aes(waiting, eruptions, color = eruptions > 3))+
  geom_point()
p + stat_ellipse()
```




```
# Changer le type d'ellipses:
# Valeurs possibles "t", "norm", "euclid"
p + stat_ellipse(type = "norm")
```



Nuage de point avec distribution marginale

Step 1/3 Créer des données:

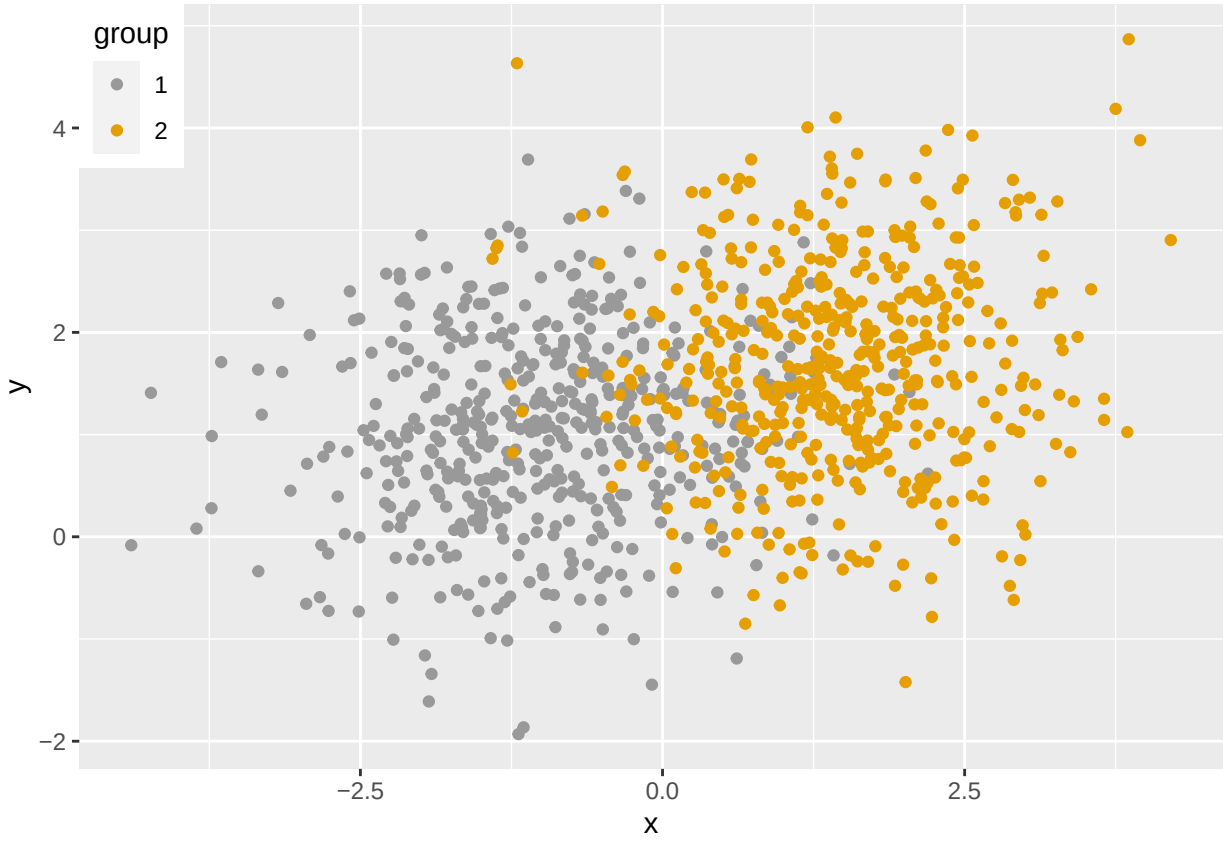
```
set.seed(1234)
x <- c(rnorm(500, mean = -1), rnorm(500, mean = 1.5))
y <- c(rnorm(500, mean = 1), rnorm(500, mean = 1.7))
group <- as.factor(rep(c(1,2), each=500))
df <- data.frame(x, y, group)
head(df)
```

```
##             x          y group
## 1 -2.20706575 -0.2053334     1
## 2 -0.72257076  1.3014667     1
## 3  0.08444118 -0.5391452     1
## 4 -3.34569770  1.6353707     1
## 5 -0.57087531  1.7029518     1
## 6 -0.49394411 -0.9058829     1
```

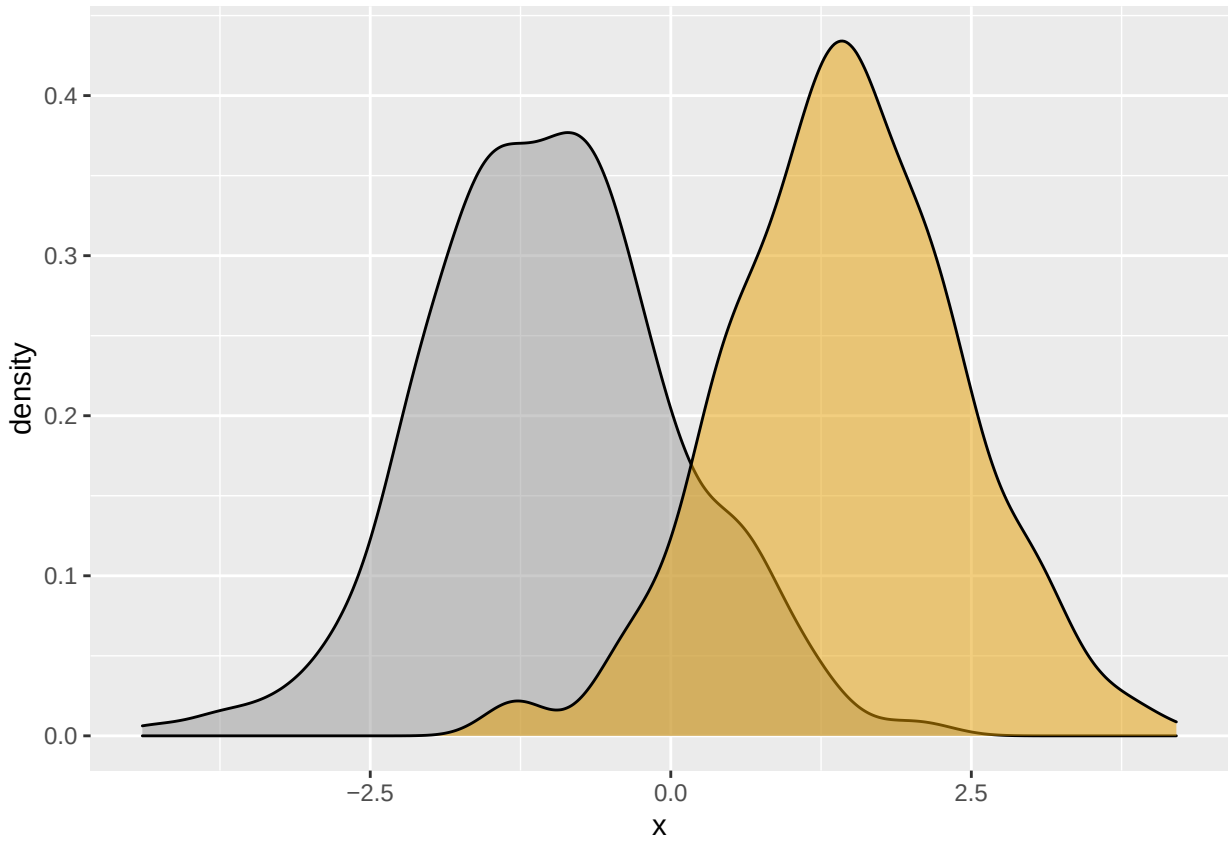
Step 2/3 Créer des graphiques

```
# Nuage de points colorés par groupes
scatterPlot <- ggplot(df,aes(x, y, color=group)) +
```

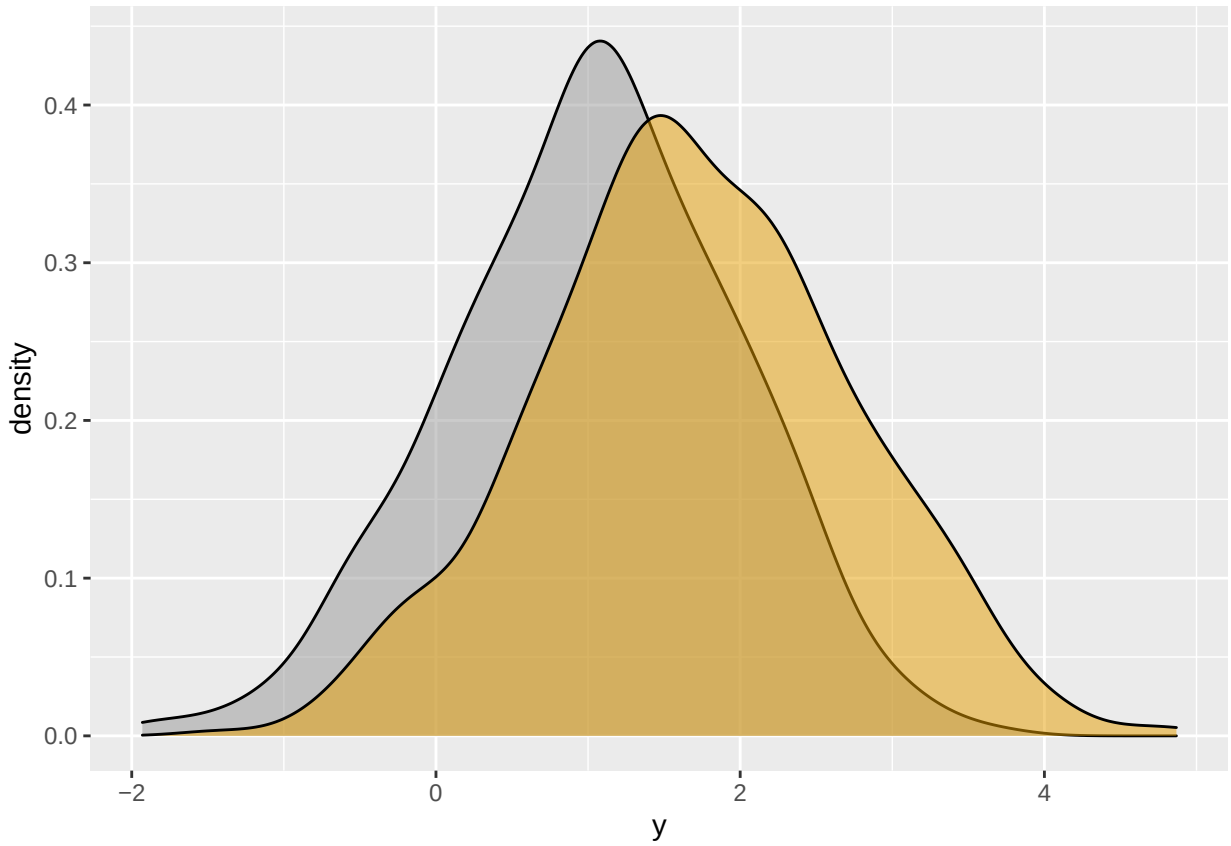
```
  geom_point() +
  scale_color_manual(values = c('#999999','#E69F00')) +
  theme(legend.position=c(0,1), legend.justification=c(0,1))
scatterPlot
```




```
# Courbe de densité marginale de x (panel du haut)
xdensity <- ggplot(df, aes(x, fill=group)) +
  geom_density(alpha=.5) +
  scale_fill_manual(values = c('#999999','#E69F00')) +
  theme(legend.position = "none")
xdensity
```




```
# Courbe de densité marginale de y (panel de droite)
ydensity <- ggplot(df, aes(y, fill=group)) +
  geom_density(alpha=.5) +
  scale_fill_manual(values = c('#999999','#E69F00')) +
  theme(legend.position = "none")
ydensity
```



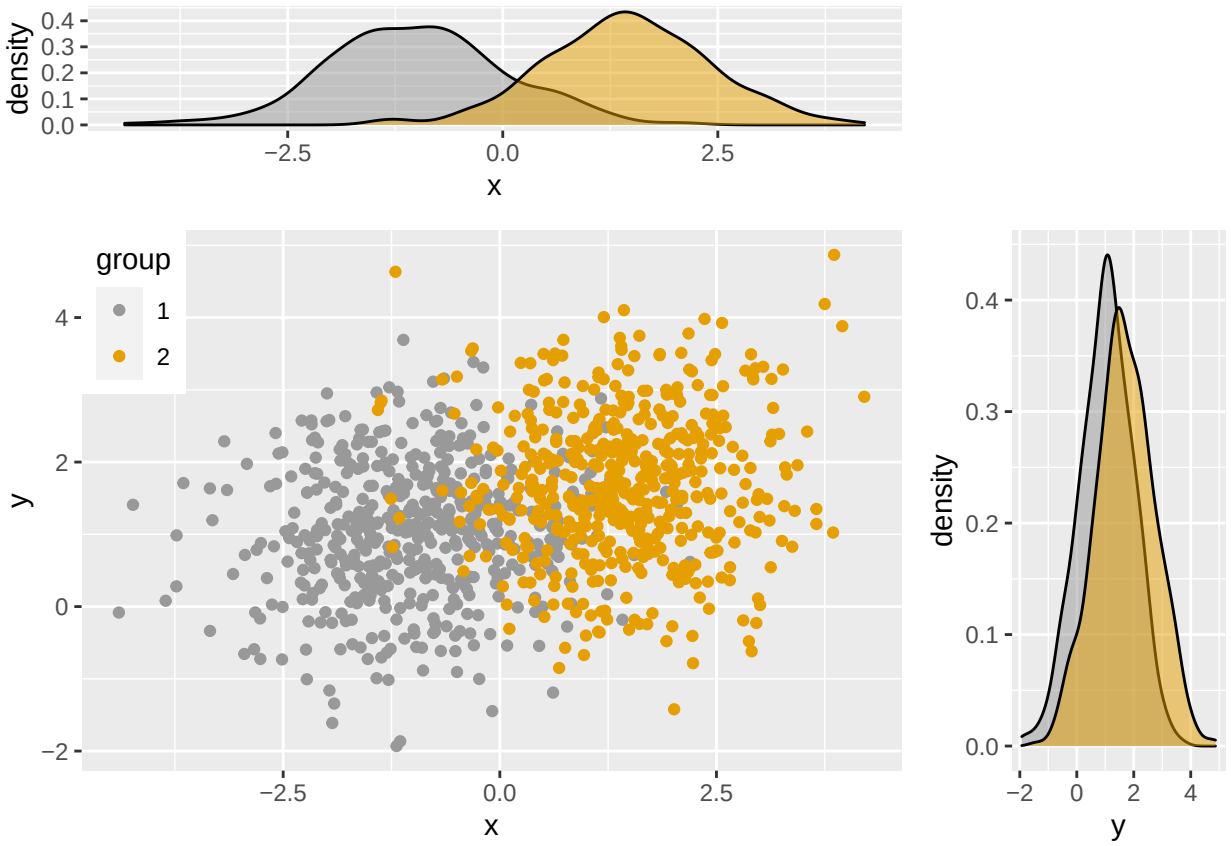
Créer un emplacement vide :

```
blankPlot <- ggplot()+geom_blank(aes(1,1))+
  theme(plot.background = element_blank(),
   panel.grid.major = element_blank(),
   panel.grid.minor = element_blank(),
   panel.border = element_blank(),
   panel.background = element_blank(),
   axis.title.x = element_blank(),
   axis.title.y = element_blank(),
   axis.text.x = element_blank(),
   axis.text.y = element_blank(),
   axis.ticks = element_blank()
     )
```

Step 3/3 Regrouper les graphiques:

Pour mettre plusieurs graphiques sur la même page, le package `gridExtra` peut être utilisé. Arranger le graphique avec des largeurs et des hauteurs adaptées pour chaque ligne et chaque colonne:

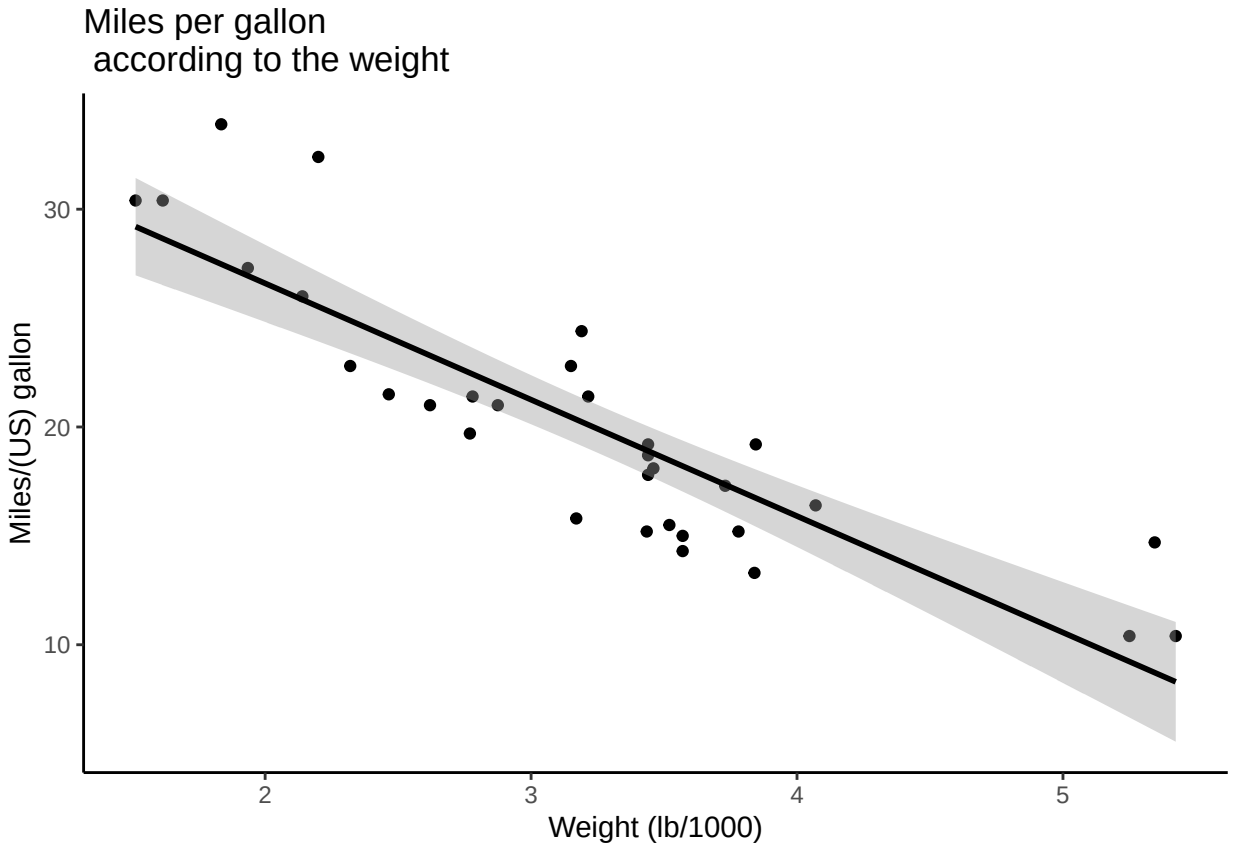
```
library(gridExtra)
grid.arrange(xdensity, blankPlot, scatterPlot, ydensity,
        ncol=2, nrow=2, widths=c(4, 1.4), heights=c(1.4, 4))
```



Nuages de points personnalisés

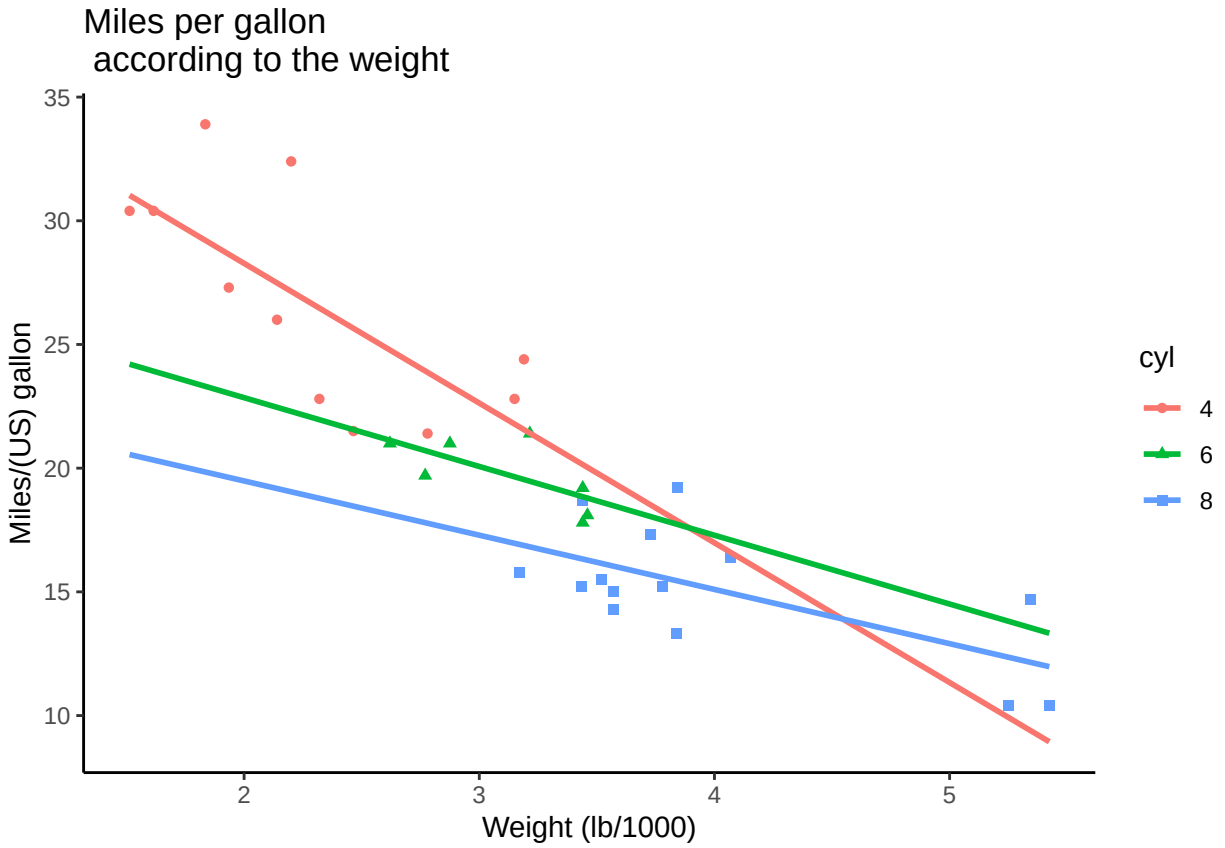
```
# Nuage de points simple
ggplot(mtcars, aes(x=wt, y=mpg)) +
  geom_point()+
  geom_smooth(method=lm, color="black")+
  labs(title="Miles per gallon \n according to the weight",
       x="Weight (lb/1000)", y = "Miles/(US) gallon")+
  theme_classic()
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
# Changer la couleur/ le type par groupe
# Supprimer l'intervalle de confiance
p <- ggplot(mtcars, aes(x=wt, y=mpg, color=cyl, shape=cyl)) +
  geom_point()+
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE)+
  labs(title="Miles per gallon \n according to the weight",
       x="Weight (lb/1000)", y = "Miles/(US) gallon")
p + theme_classic()
```

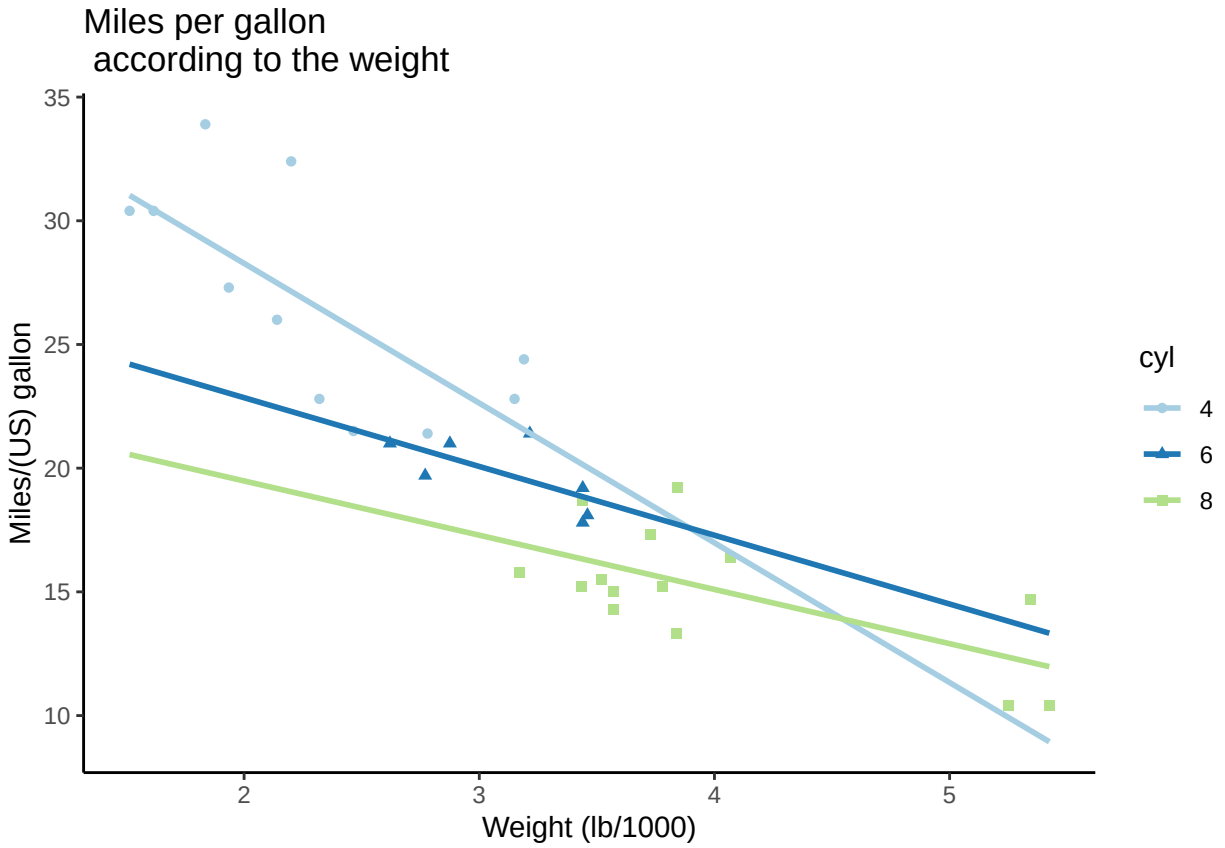
```
## `geom_smooth()` using formula 'y ~ x'
```



Changer les couleurs manuellement

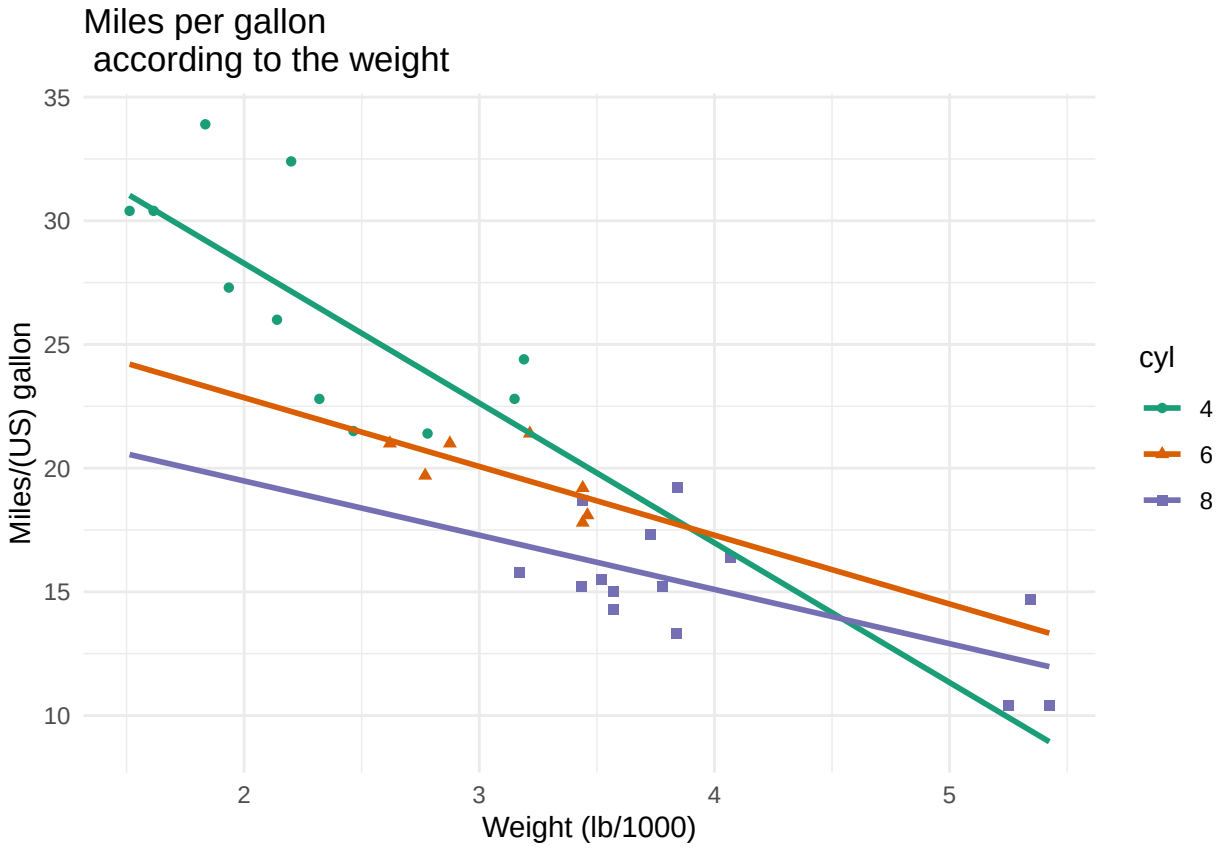
```
# Couleurs continues
p + scale_color_brewer(palette="Paired") + theme_classic()
```

```
## `geom_smooth()` using formula 'y ~ x'
```



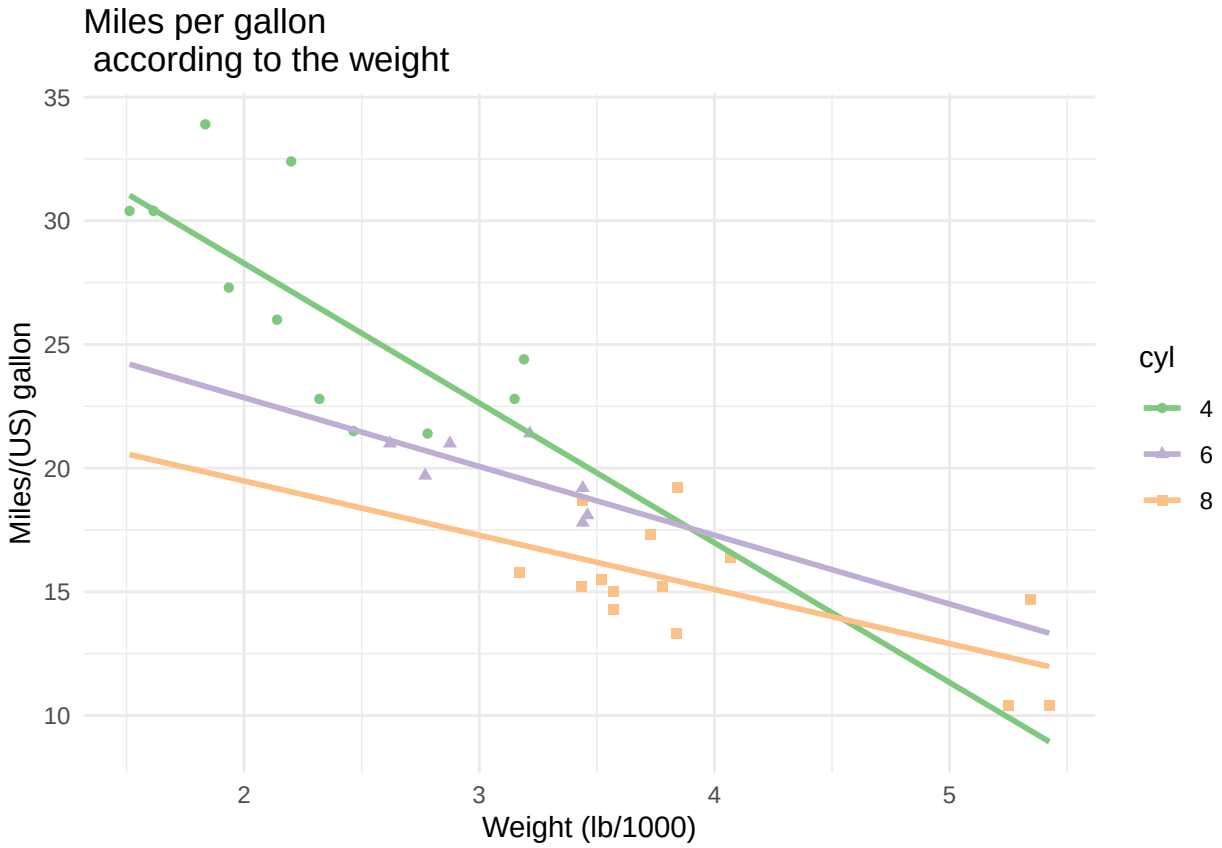

```
# Couleurs discrètes
p + scale_color_brewer(palette="Dark2") + theme_minimal()
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
# Couleurs en gradient
p + scale_color_brewer(palette="Accent") + theme_minimal()
```

```
## `geom_smooth()` using formula 'y ~ x'
```


Miles per gallon
according to the weight
Miles/(US) gallon
35
30
25
20
15
10
2
3
4
5
Weight (lb/1000)
cyl
4
6
8

ggplot Cheat sheet

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.

required

```
ggplot (data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
  stat = <STAT>, position = <POSITION>) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

Not required, sensible defaults supplied

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

aesthetic mappings — data — geom

qplot(x = cty, y = hwy, data = mpg, geom = "point") Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

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.

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()
(Useful for expanding limits)

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(group = group))
x, y, alpha, color, fill, group, 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

continuous x , continuous y

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

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

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

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, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

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

discrete x , discrete y

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

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

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_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, 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))

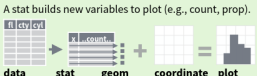
l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

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_density2d(aes(fill = ..level..), geom = "polygon")

variable created by stat

c + stat_bin(binwidth = 1, origin = 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() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) **x** | ..x.., ..y..

e + stat_identity(na.rm = TRUE)

ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=list(df=5)) **sample, x, y** | ..sample.., ..theoretical..

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

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

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

e + stat_unique()

Scales

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_*_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 x values as date times. Use same arguments as scale_x_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 direction of x axis
scale_x_sqrt() - Plot x on square root scale

COLOR AND FILL SCALES (DISCRETE)

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

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")

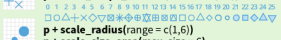
o + scale_fill_gradient(low="red", high="yellow")

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

o + scale_fill_gradientn(colours=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))



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

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

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

r + coord_trans(ytrans = "sqrt")
xtrans, ytrans, limx, limy
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 themes
r + theme_void()
Empty theme

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 labels

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

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", <AES> = "New <AES> legend title")

Use scale functions to update legend labels

t + annotate(geom = "text", x = 8, y = 9, label = "A")

geom to place | manual values for geom's aesthetics

Legends

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

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

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(0, 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))

RStudio