

Visualisation de données avec R

La “*Grammar of Graphics*”

Robin Cura & Lise Vaudor

15/10/2018

École Thématique GeoViz 2018

Sommaire

La Grammar of Graphics

Les composants d'un graphique ggplot2

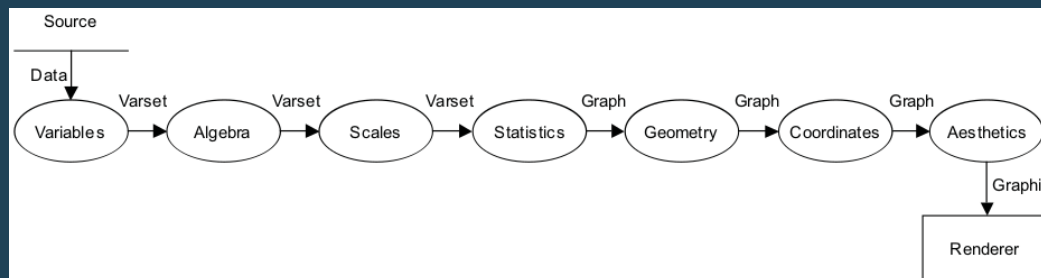
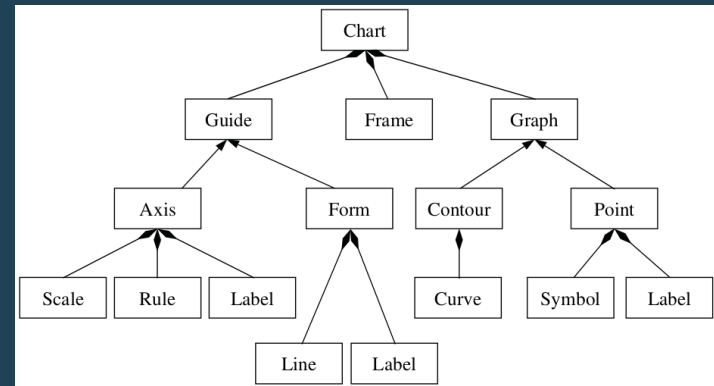
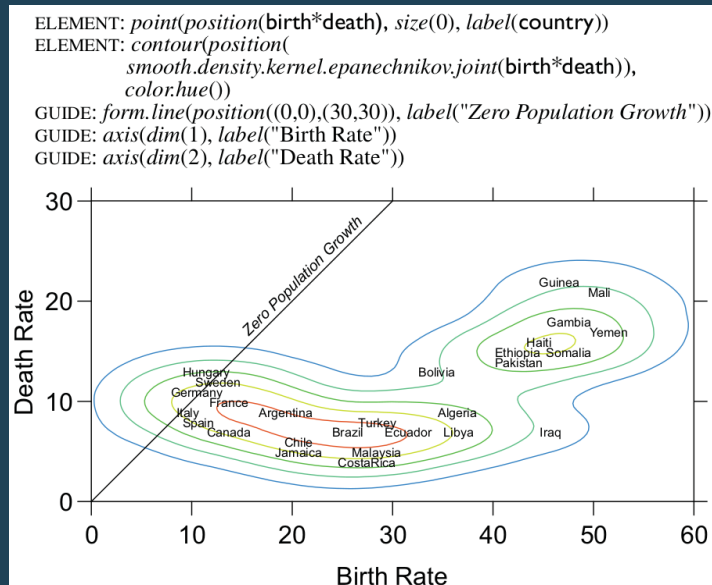
- Des données **tidy**
- Les géométries : **geom_**
- Mapper les variables aux géométries : les **aesthetics**
- Organiser les échelles : **scale_**
- Ratios et systèmes de coordonnées : **coords**
- Les **labels** d'un graphique
- Définir les légendes : les **guides**
- Décomposer un graphique selon des **facets**
- Superposition de couches
- Composition d'un planche avec **patchwork**

Exercice d'application

Ressources et supports

La Grammar of Graphics

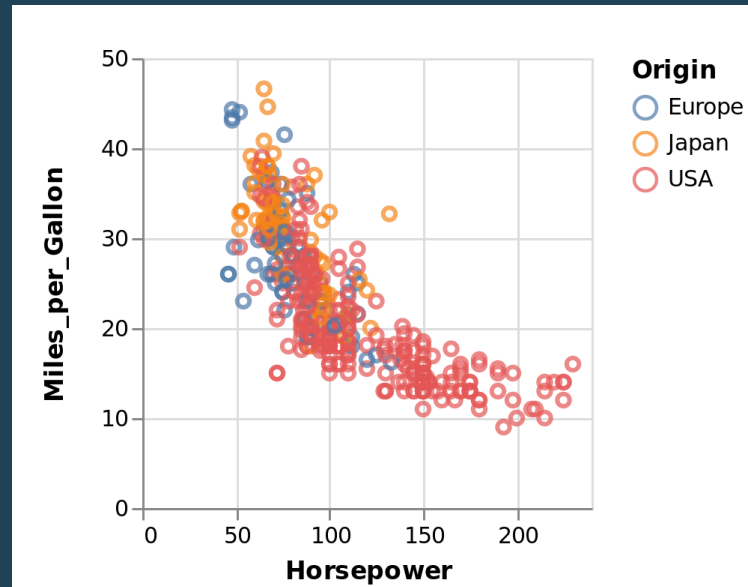
Conceptualisée par Leland Wilkinson (2006 [1999])



La Grammar of Graphics

VegaLite (JavaScript)

```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v2.json",
  "data": {"url": "data/cars.json"},
  "mark": "point",
  "encoding": {
    "x": {"field": "Horsepower", "type": "quantitative"},
    "y": {"field": "Miles_per_Gallon", "type": "quantitative"},
    "color": {"field": "Origin", "type": "nominal"}
  }
}
```

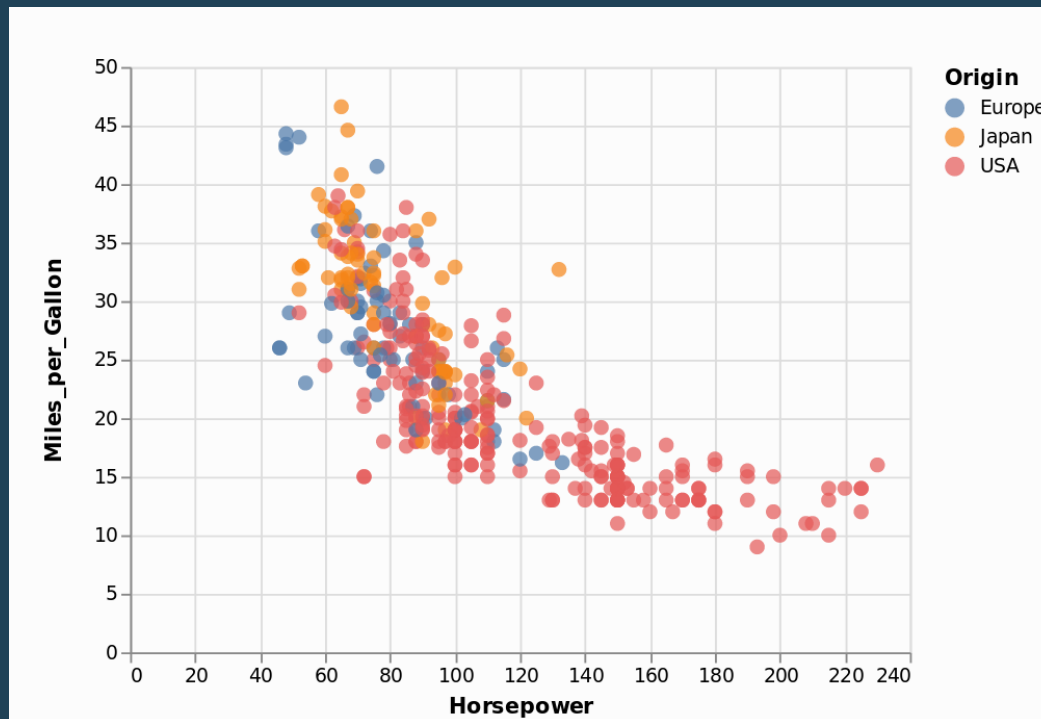


La *Grammar of Graphics*

Altair (Python)

```
import altair as alt
```

```
alt.Chart(cars).mark_circle(size=60).encode(x='Horsepower', y='Miles_per_Gallon', color='Origin')
```

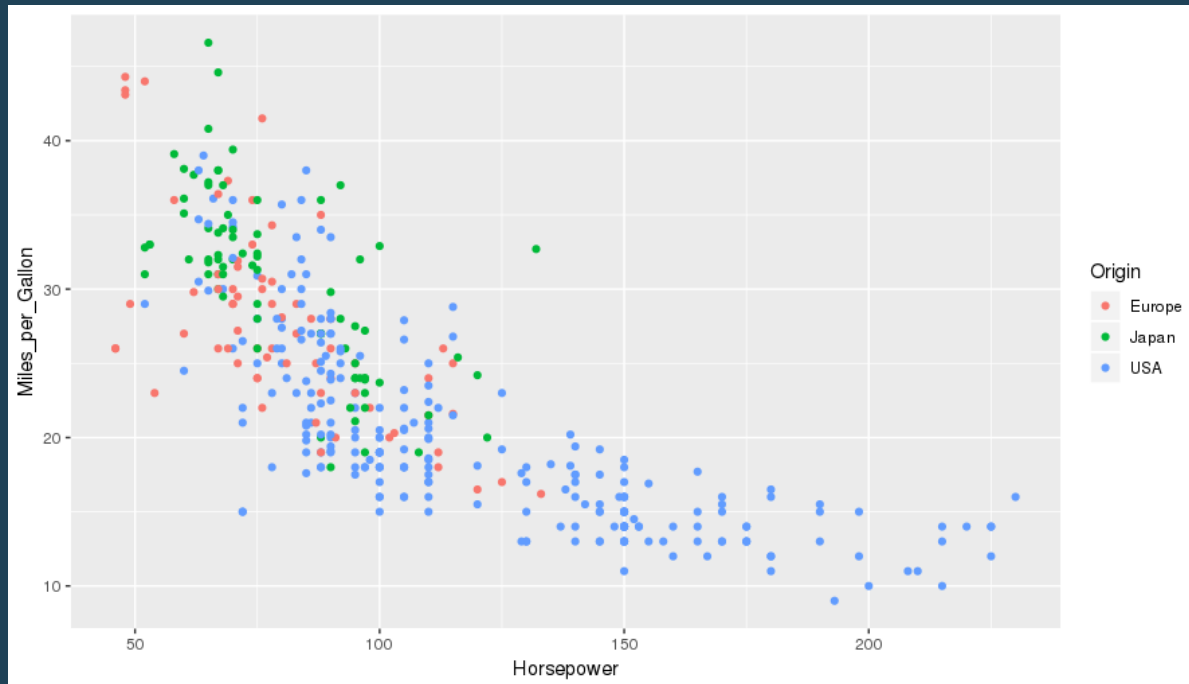


La Grammar of Graphics

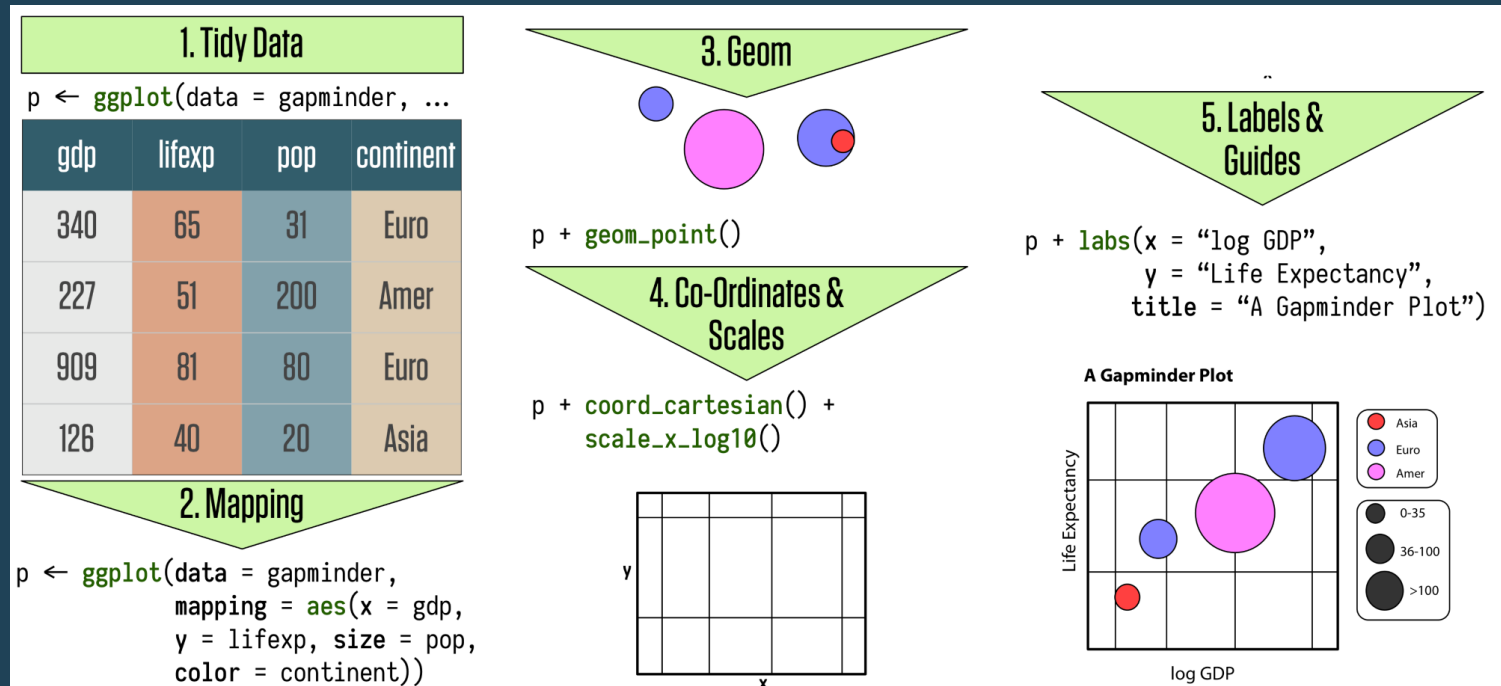
ggplot2 (R)

```
library(ggplot2)
```

```
ggplot(cars) +  
  geom_point(aes(x = Horsepower, y = Miles_per_Gallon, color = Origin))
```



ggplot2 : une implémentation de la *grammar of graphics*



Healy, K. (2018). Data Visualization: A Practical Introduction. S.I.: Princeton University Press. <http://socviz.co/>

ggplot2 : une implémentation de la *grammar of graphics*

Décrire la syntaxe d'un graphique à l'aide d'une grammaire composée d'éléments atomiques :

- Des données (**data**)
- Des objets géométriques (point, ligne, rectangle, barre etc.) (**geom_***)
 - Une *mapping* entre les données et les "esthétiques" géométriques correspondantes (*aesthetics*, **aes()**)
- Des échelles pour définir les étendues et transformations de l'espace du graphique (**scales_**)
- Un système de coordonnées pour définir le lien entre les échelles (**coord_**)
- Des éléments d'habillage : textes (*labels*, **labs**) et légendes (*guides*, **guides**)
- Éventuellement, des éléments de décomposition des graphiques, les **facets**

Tous ces éléments s'accumulent, comme des propriétés supplémentaires, via l'opérateur **+**

Les données

- Tout graphique **ggplot2** doit commencer par un appel aux données :

```
library(ggplot2) # ou library(tidyverse)
df_dmr_spatialise <- readRDS(file = "dans_ma_rue_clean.RDS")
ggplot(data = df_dmr_spatialise)
```

- Les données doivent être structurées de manière **tidy**, dans un **data.frame** (ou un **tibble**) :
 - 1 ligne = 1 individu
 - 1 colonne = 1 variable homogène et cohérente
 - ▷ On privilégie des données au format **long**
- Ex. pour stocker des données temporelles :











Ville	CP	Pop1999	Pop2011
Paris	75000	2125	2250
Lyon	69000	445	491
Marseille	13000	796	851


Ville	CP	Annee	Population
Paris	75000	1999	2125
Paris	75000	2011	2250
Lyon	69000	1999	445
Lyon	69000	2011	491
Marseille	13000	1999	796
Marseille	13000	2011	851

Les géométries

- Le composant de base des graphiques : quelle géométrie donner à chaque couche (*layer*) de données ?

<https://ggplot2.tidyverse.org/reference/index.html#section-layer-geoms>

	<code>geom_abline()</code> <code>geom_hline()</code> <code>geom_vline()</code>	Reference lines: horizontal, vertical, and diagonal
	<code>geom_bar()</code> <code>geom_col()</code> <code>stat_count()</code>	Bar charts
	<code>geom_bin2d()</code> <code>stat_bin_2d()</code>	Heatmap of 2d bin counts
	<code>geom_blank()</code>	Draw nothing
	<code>geom_boxplot()</code> <code>stat_boxplot()</code>	A box and whiskers plot (in the style of Tukey)
	<code>geom_contour()</code> <code>stat_contour()</code>	2d contours of a 3d surface
	<code>geom_count()</code> <code>stat_sum()</code>	Count overlapping points
	<code>geom_density()</code> <code>stat_density()</code>	Smoothed density estimates
	<code>geom_density_2d()</code> <code>stat_density_2d()</code>	Contours of a 2d density estimate
	<code>geom_dotplot()</code>	Dot plot
	<code>geom_errorbarh()</code>	Horizontal error bars
	<code>geom_hex()</code> <code>stat_bin_hex()</code>	Hexagonal heatmap of 2d bin counts
	<code>geom_freqpoly()</code> <code>geom_histogram()</code> <code>stat_bin()</code>	Histograms and frequency polygons
	<code>geom_jitter()</code>	Jittered points
	<code>geom_crossbar()</code> <code>geom_errorbar()</code> <code>geom_linerange()</code> <code>geom_pointrange()</code>	Vertical intervals: lines, crossbars & errorbars

	<code>geom_map()</code>	Polygons from a reference map
	<code>geom_path()</code> <code>geom_line()</code> <code>geom_step()</code>	Connect observations
	<code>geom_point()</code>	Points
	<code>geom_polygon()</code>	Polygons
	<code>geom_qq_line()</code> <code>stat_qq_line()</code> <code>geom_qq()</code> <code>stat_qq()</code>	A quantile-quantile plot
	<code>geom_quantile()</code> <code>stat_quantile()</code>	Quantile regression
	<code>geom_ribbon()</code> <code>geom_area()</code>	Ribbons and area plots
	<code>geom_rug()</code>	Rug plots in the margins
	<code>geom_segment()</code> <code>geom_curve()</code>	Line segments and curves
	<code>geom_smooth()</code> <code>stat_smooth()</code>	Smoothed conditional means
	<code>geom_spoke()</code>	Line segments parameterised by location, direction and distance
	<code>geom_label()</code> <code>geom_text()</code>	Text
	<code>geom_raster()</code> <code>geom_rect()</code> <code>geom_tile()</code>	Rectangles
	<code>geom_violin()</code> <code>stat_ydensity()</code>	Violin plot
	<code>stat_sf()</code> <code>geom_sf()</code> <code>coord_sf()</code>	Visualise sf objects

Les géométries

- Une liste complète dans la [cheatsheet de ggplot2](#)

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemployment))
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 = 2)) - x, yend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path() (lineend = "butt", linejoin = "round", linewidth = 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 = unemployment - 900, ymax = unemployment + 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(fill))
```

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 = "b"), 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

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

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, unemployment))
```

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)

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

Les aesthetics

- Chaque géométrie comporte des propriétés, génériques (**x**, **y**, **colour**...) ou spécifiques (**linetype**, **shape**, **xmin**...) qui doivent être rapportées (avec un **mapping**) à des variables : ce sont les "esthétiques" (**aes**)
- Ex. : une géométrie de type ponctuelle (**geom_point**) requiert au moins des coordonnées (**x** et **y**), et peut être enrichie avec des propriétés spécifiques (**alpha**, **color**, **fill**, **shape**, **size**, **stroke**) :

```
pointData <- df_dmr_spatialise %>%  
  filter(  
    TYPE == "Problème sur un chantier"  
  )  
  
ggplot(pointData) +  
  geom_point(aes(x = Long,  
                 y = Lat,  
                 color = CODE_POSTAL))
```

Les aesthetics

- Chaque géométrie comporte des propriétés, génériques (**x**, **y**, **colour**...) ou spécifiques (**linetype**, **shape**, **xmin**...) qui doivent être rapportées (avec un **mapping**) à des variables : ce sont les "esthétiques" (**aes**)
- Ex. : une géométrie de type ponctuelle (**geom_point**) requiert au moins des coordonnées (**x** et **y**), et peut être enrichie avec des propriétés spécifiques (**alpha**, **color**, **fill**, **shape**, **size**, **stroke**)
- On peut aussi définir des propriétés "globales", qui ne dépendront pas du contenu du jeu de données :

```
pointData <- df_dmr_spatialise %>%  
  filter(  
    TYPE == "Problème sur un chantier"  
  )  
  
ggplot(pointData) +  
  geom_point(aes(x = Long,  
                 y = Lat,  
                 color = CODE_POSTAL),  
             size = 3,  
             alpha = .2)
```

Les aesthetics

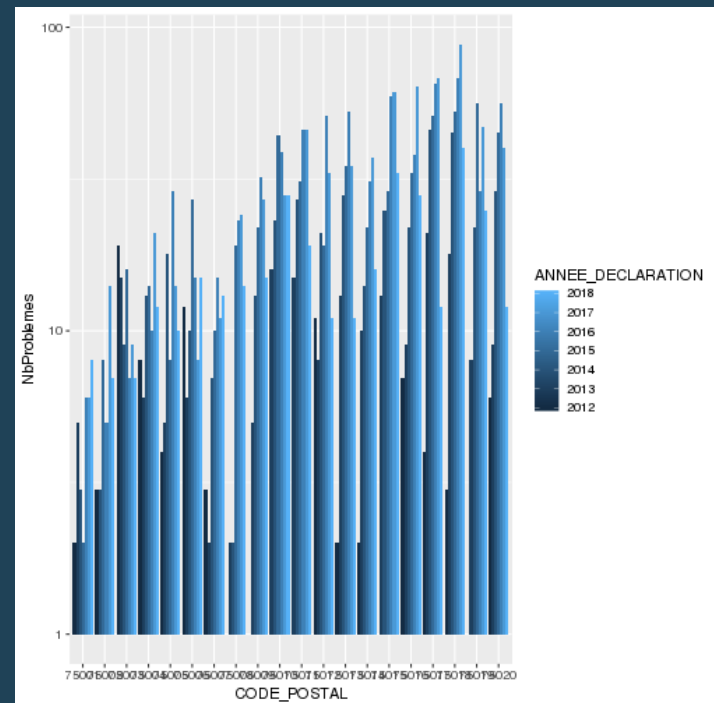
- Chaque géométrie comporte des propriétés, génériques (**x**, **y**, **colour**...) ou spécifiques (**linetype**, **shape**, **xmin**...) qui doivent être rapportées (avec un **mapping**) à des variables : ce sont les "esthétiques" (**aes**)
- Ex. : une géométrie de type "colonne" (**geom_col**) requiert au moins des coordonnées (**x** et **y**), et peut être enrichie avec des propriétés spécifiques (**x**, **y**, **alpha**, **color**, **fill**, **group**, **linetype**, **size**, **position**)

```
colData <- df_dmr_spatialise %>%  
  filter(  
    TYPE == "Problème sur un chantier"  
  ) %>%  
  group_by(CODE_POSTAL,  
           ANNEE_DECLARATION) %>%  
  summarise(NbProblemes = n())  
  
ggplot(colData) +  
  geom_col(aes(x = CODE_POSTAL,  
              y = NbProblemes,  
              fill = ANNEE_DECLARATION,  
              group = ANNEE_DECLARATION),  
          position = "dodge")
```

Les échelles

- Quand on veut modifier la manière dont les axes sont conçus, on peut jouer sur les **échelles** (**scale_**)
- Par exemple, pour choisir une échelle logarithmique pour les ordonnées du graphique précédent :

```
colData <- df_dmr_spatialise %>%  
  filter(  
    TYPE == "Problème sur un chantier"  
  ) %>%  
  group_by(CODE_POSTAL,  
    ANNEE_DECLARATION) %>%  
  summarise(NbProblemes = n())  
  
ggplot(colData) +  
  geom_col(aes(x = CODE_POSTAL,  
    y = NbProblemes,  
    fill = ANNEE_DECLARATION,  
    group = ANNEE_DECLARATION),  
    position = "dodge") +  
  scale_y_log10()
```



Les échelles

- Quand on veut modifier la manière dont les axes sont conçus, on peut jouer sur les **échelles** (**scale_**)
- Par exemple, pour choisir une échelle logarithmique pour les ordonnées du graphique précédent
- Et pour modifier l'échelle de couleurs :

```
colData <- df_dmr_spatialise %>%  
  filter(  
    TYPE == "Problème sur un chantier"  
  ) %>%  
  group_by(CODE_POSTAL,  
    ANNEE_DECLARATION) %>%  
  summarise(NbProblemes = n())  
  
ggplot(colData) +  
  geom_col(aes(x = CODE_POSTAL,  
    y = NbProblemes,  
    fill = as.character(ANNEE_DECLARATION),  
    group = as.character(ANNEE_DECLARATION)),  
    position = "dodge") +  
  scale_y_log10() +  
  scale_fill_brewer(palette = "Greens", direction = -1)
```


Les coordonnées

- On l'a vu avec les "cartes" de répartition, **ggplot2** affiche un graphique en maximisant l'espace disponible pour chaque axe : quand les rapports entre les coordonnées **x** et **y** ont un sens, il faut l'explicitier avec les **coord_** :

```
pointData <- df_dmr_spatialise %>% filter( TYPE == "Problème sur un chantier")
```

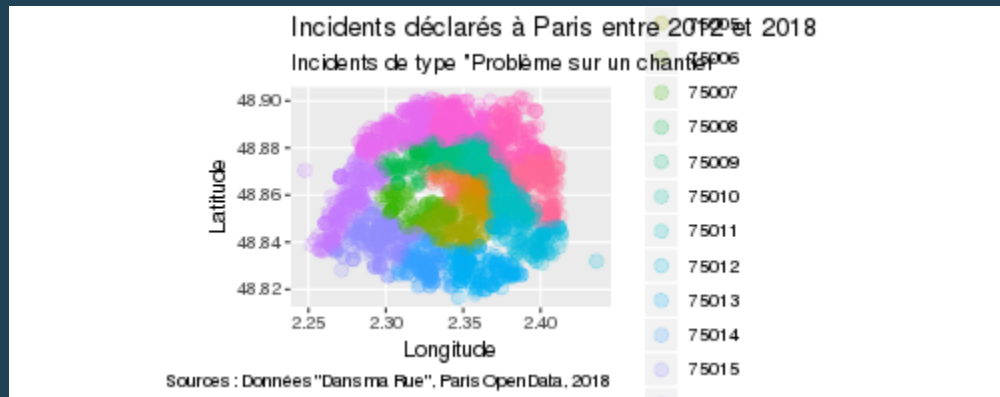
```
ggplot(pointData) +  
  geom_point(aes(x = Long, y = Lat,  
                 color = CODE_POSTAL),  
             size = 3, alpha = .2) +  
  coord_fixed(ratio = 1) # On fixe 1x = 1y
```

```
ggplot(pointData) +  
  geom_point(aes(x = Long, y = Lat,  
                 color = CODE_POSTAL),  
             size = 3, alpha = .2) +  
  coord_map(projection = "mercator")  
# ratio x/y selon la projection de Mercator
```

Habillage : textes

- Par défaut, les textes (titres des axes etc.) prennent comme valeur automatique le nom des variables qui leurs sont attribuées.
- On peut modifier tous ces éléments, afin d'habiller correctement les graphiques produits, avec l'instruction **labs** (*labels*) :

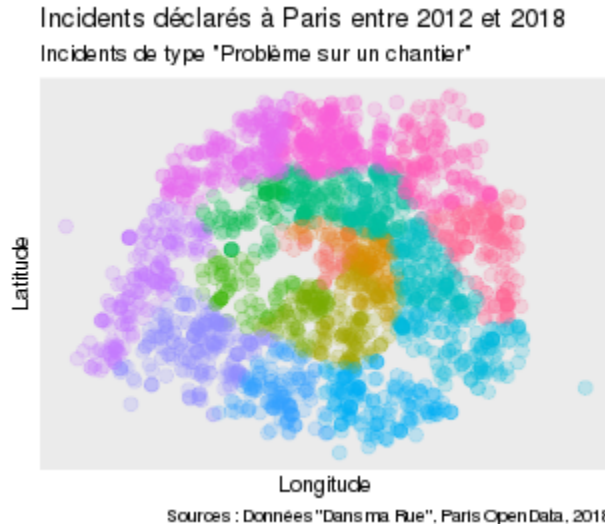
```
ggplot(pointData) +  
  geom_point(aes(x = Long, y = Lat, color = CODE_POSTAL), size = 3, alpha = .2) +  
  coord_map(projection = "mercator") +  
  labs(title = 'Incidents déclarés à Paris entre 2012 et 2018',  
        subtitle = 'Incidents de type "Problème sur un chantier"',  
        caption = 'Sources : Données "Dans ma Rue", Paris OpenData, 2018',  
        x = "Longitude", y = "Latitude")
```



Habillage : légendes

- On peut modifier et désactiver les légendes directement depuis les échelles (**scales**) correspondantes.

```
ggplot(pointData) +  
  geom_point(aes(x = Long, y = Lat, color = CODE_POSTAL), size = 3, alpha = .2) +  
  coord_map(projection = "mercator") +  
  labs(title = 'Incidents déclarés à Paris entre 2012 et 2018',  
        subtitle = 'Incidents de type "Problème sur un chantier"',  
        caption = 'Sources : Données "Dans ma Rue", Paris OpenData, 2018',  
        x = "Longitude", y = "Latitude") +  
  scale_color_discrete(guide = FALSE) +  
  scale_x_continuous(labels = NULL, breaks = NULL, minor_breaks = NULL) +  
  scale_y_continuous(labels = NULL, breaks = NULL, minor_breaks = NULL)
```



Décomposer un graphique : les *facets*

On peut "décomposer" un graphique (en *small multiples*) avec les instructions `facet_` :

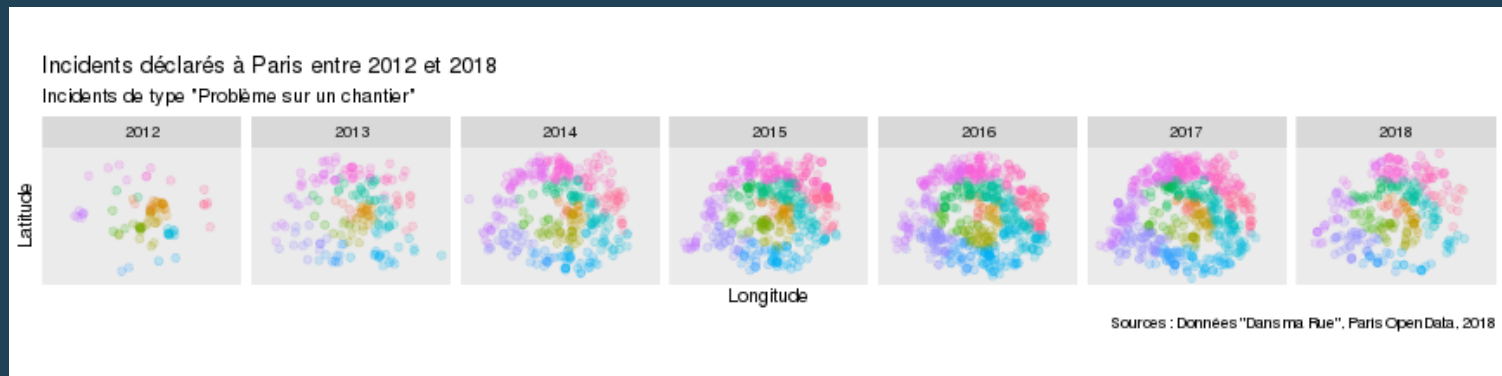
- `facet_wrap(~VARIABLE)` : décompose le graphique en autant de modalités que contenues dans `VARIABLE`. On règle leur agencement avec les arguments `nrow` (nombre de lignes) et/ou `ncol` (nombre de colonnes).

```
ggplot(pointData) +  
  geom_point(aes(x = Long, y = Lat, color = CODE_POSTAL), size = 2, alpha = .2) +  
  coord_map(projection = "mercator") +  
  labs(title = 'Incidents déclarés à Paris entre 2012 et 2018',  
        subtitle = 'Incidents de type "Problème sur un chantier"',  
        caption = 'Sources : Données "Dans ma Rue", Paris OpenData, 2018',  
        x = "Longitude", y = "Latitude") +  
  scale_color_discrete(guide = FALSE) +  
  scale_x_continuous(labels = NULL, breaks = NULL, minor_breaks = NULL) +  
  scale_y_continuous(labels = NULL, breaks = NULL, minor_breaks = NULL) +  
  facet_wrap(~ANNEE_DECLARATION, nrow = 1)
```

Décomposer un graphique : les *facets*

On peut "décomposer" un graphique (en *small multiples*) avec les instructions `facet_` :

- `facet_wrap(~VARIABLE)` : décompose le graphique en autant de modalités que contenues dans `VARIABLE`. On règle leur agencement avec les arguments `nrow` (nombre de lignes) et/ou `ncol` (nombre de colonnes).



Décomposer un graphique : les *facets*

On peut "décomposer" un graphique (en *small multiples*) avec les instructions `facet_` :

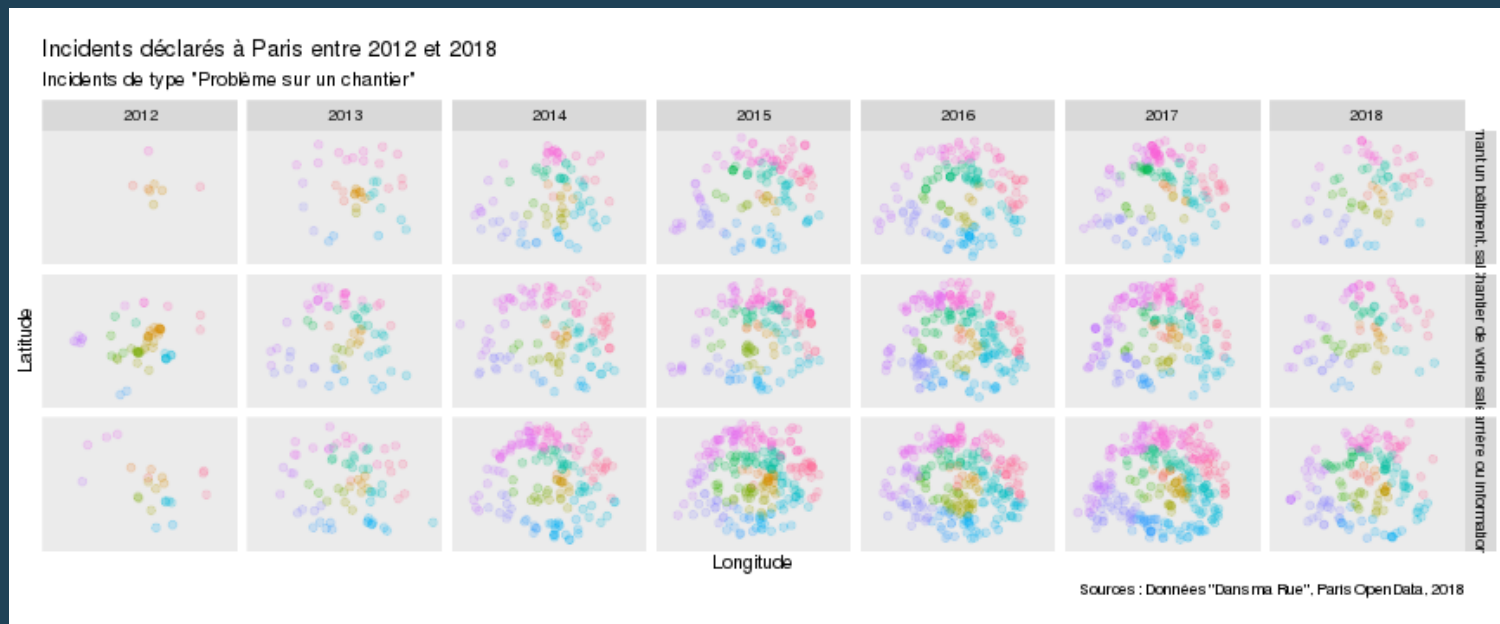
- `facet_grid(VARIABLE1~VARIABLE2)` : décompose le graphique en un croisement des modalités contenues dans `VARIABLE1` et `VARIABLE2`.

```
ggplot(pointData %>% filter(!is.na(SOUSTYPE))) +  
  geom_point(aes(x = Long, y = Lat, color = CODE_POSTAL), size = 2, alpha = .2) +  
  coord_map(projection = "mercator") +  
  labs(title = 'Incidents déclarés à Paris entre 2012 et 2018',  
        subtitle = 'Incidents de type "Problème sur un chantier"',  
        caption = 'Sources : Données "Dans ma Rue", Paris OpenData, 2018',  
        x = "Longitude", y = "Latitude") +  
  scale_color_discrete(guide = FALSE) +  
  scale_x_continuous(labels = NULL, breaks = NULL, minor_breaks = NULL) +  
  scale_y_continuous(labels = NULL, breaks = NULL, minor_breaks = NULL) +  
  facet_grid(SOUSTYPE~ANNEE_DECLARATION)
```

Décomposer un graphique : les *facets*

On peut "décomposer" un graphique (en *small multiples*) avec les instructions `facet_` :

- `facet_grid(VARIABLE1~VARIABLE2)` : décompose le graphique en un croisement des modalités contenues dans `VARIABLE1` et `VARIABLE2`.



Superposer des couches graphiques

ggplot2 permet d'empiler des couches de représentations portant sur les mêmes données (représentations différentes) ou sur d'autres jeux de données (si tant est qu'ils se situent dans les mêmes étendues graphiques). Comme pour tous les éléments de **ggplot**, cela se fait avec l'opérateur **+** :

```
maCarte <- ggplot(pointData) +  
  geom_point(aes(x = Long, y = Lat)) +  
  coord_map(projection = "mercator")  
maCarte
```

```
maCarte +  
  geom_density2d(aes(x = Long, y = Lat))
```


Superposer des couches graphiques

- **N.B.** : l'ordre d'ajout des couches est important : les dernières couches ajoutées seront positionnées "au dessus" des couches précédentes :

```
ggplot(pointData) +  
  aes(x = Long, y = Lat) +  
  geom_point(colour = "white", alpha = .4) +  
  stat_density_2d(aes(fill = stat(level)),  
                  geom = "polygon") +  
  coord_map(projection = "mercator")
```

```
ggplot(pointData) +  
  aes(x = Long, y = Lat) +  
  stat_density_2d(aes(fill = stat(level)),  
                  geom = "polygon") +  
  geom_point(colour = "white", alpha = .4) +  
  coord_map(projection = "mercator")
```

Composer une planche de graphiques

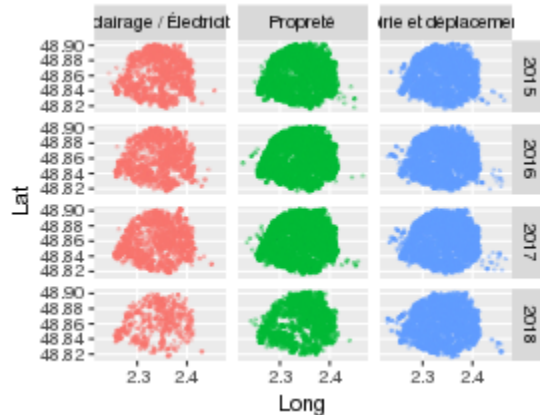
- Avec le package **patchwork**, on peut facilement agencer différents graphiques

```
library(patchwork) # devtools::install_github("thomasp85/patchwork")
```

```
carteData <- df_dmr_spatialise %>%  
  filter(TYPE %in% c("Propreté", "Voirie et déplacements", "Éclairage / Électricité")) %>%  
  filter(ANNEE_DECLARATION >= 2015) %>%  
  mutate(TRIMESTRE = paste(ANNEE_DECLARATION, TRIMESTRE_DECLARATION, sep="-")) %>%  
  arrange(TRIMESTRE, TYPE)
```

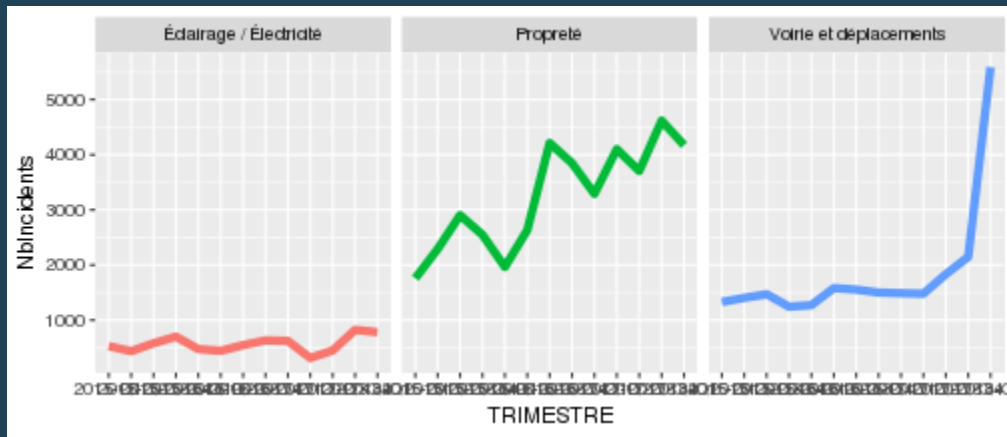
```
maCarte <- ggplot(carteData) +  
  geom_point(aes(Long, Lat, colour = TYPE), size = .5, alpha = .3) +  
  facet_grid(ANNEE_DECLARATION~TYPE) +  
  coord_map(projection = "mercator") +  
  scale_colour_discrete(guide = FALSE)
```

```
maCarte
```



Composer une planche de graphiques

```
evolData <- carteData %>%  
  group_by(TRIMESTRE, TYPE) %>%  
  summarise(NbIncidents = n())  
  
evolNombre <- ggplot(evolData) +  
  geom_line(aes(TRIMESTRE, NbIncidents, colour = TYPE, group = TYPE), size = 2) +  
  facet_wrap(~TYPE, nrow = 1) +  
  scale_colour_discrete(guide = FALSE)  
  
evolNombre
```

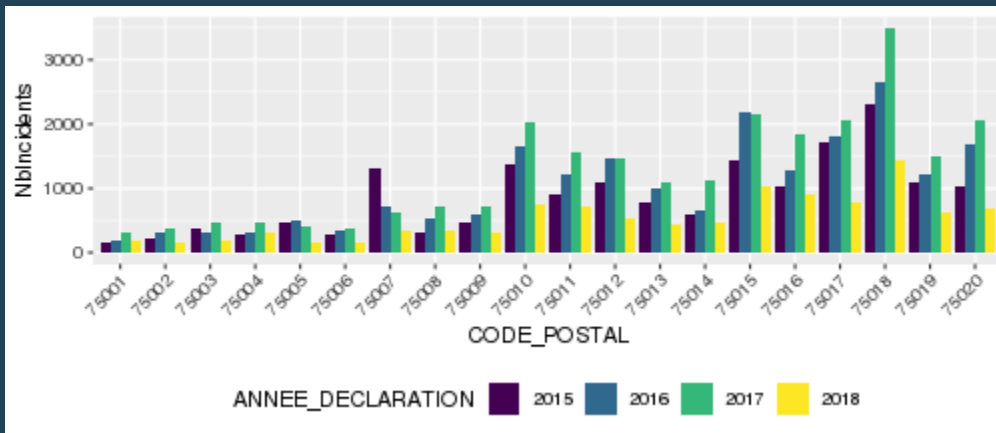


Composer une planche de graphiques

```
arrdtData <- carteData %>%
  group_by(ANNEE_DECLARATION, CODE_POSTAL) %>%
  summarise(NbIncidents = n()) %>%
  ungroup() %>%
  mutate(ANNEE_DECLARATION = as.character(ANNEE_DECLARATION))

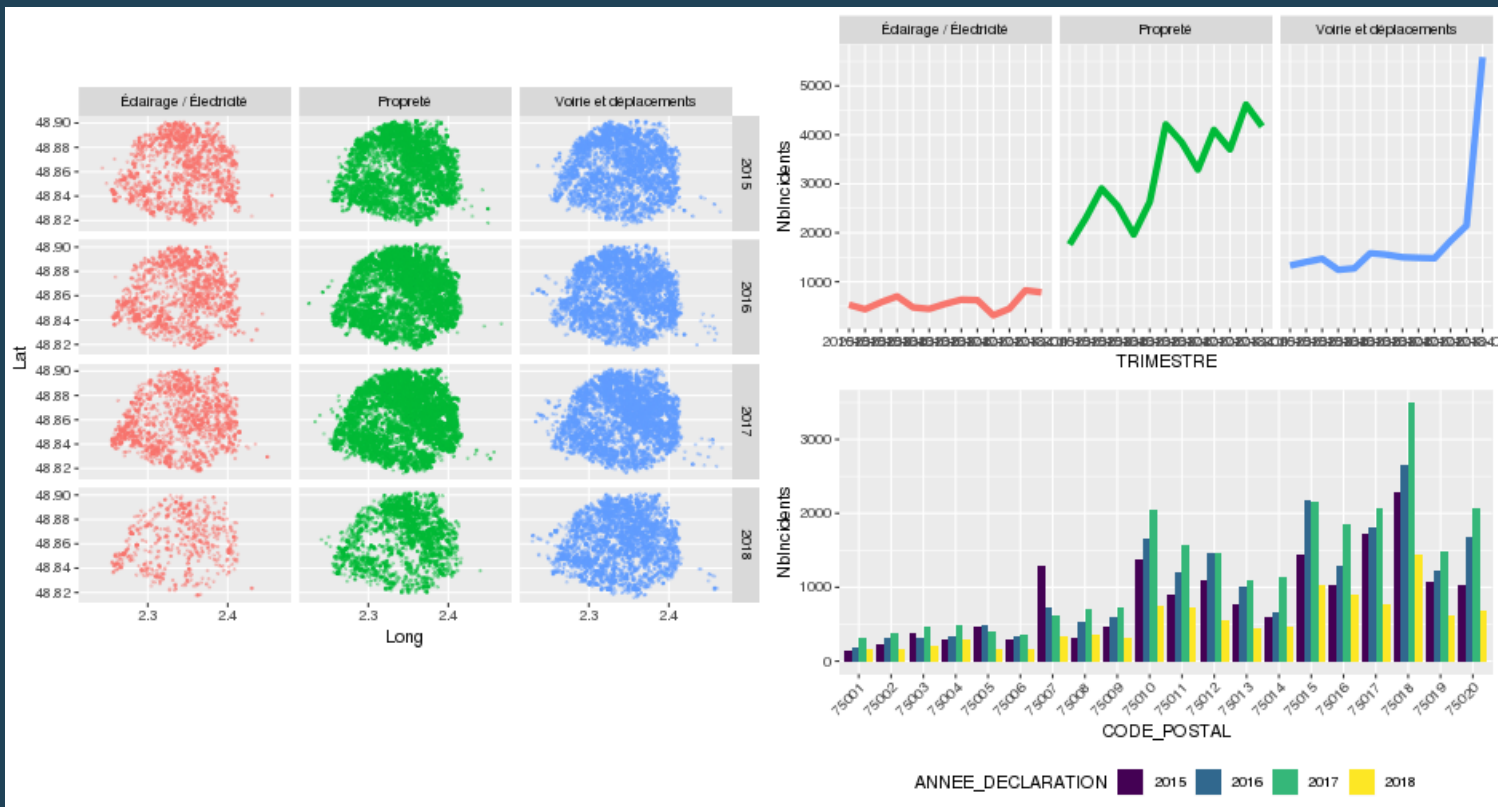
evolArrdt <- ggplot(arrdtData) +
  geom_col(aes(CODE_POSTAL, NbIncidents, fill = ANNEE_DECLARATION), position = "dodge") +
  scale_fill_viridis_d() +
  theme(legend.position="bottom",
        axis.text.x = element_text(angle = 45, hjust = 1))

evolArrdt
```



Composer une planche de graphiques

```
(maCarte | (evolNombre / evolArrdt))
```



Cheatsheets

Data Visualization with ggplot2 :: CHEAT SHEET



www.rstudio.com/resources/cheatsheets/#ggplot2

Où trouver de la documentation/aide ?

- Articles rstudio
- Blogs ? Lesquels ?
- Livres

Un peu de pratique !

Proposer des représentations graphiques, à partir du jeu de données "Dans ma Rue", permettant de saisir l'évolution, dans le temps et dans l'espace, des signalements d'anomalies.