

Initiation aux logiciels R et RStudio

Pour l'analyse de données et les représentations graphiques

Benoît Simon-Bouhet

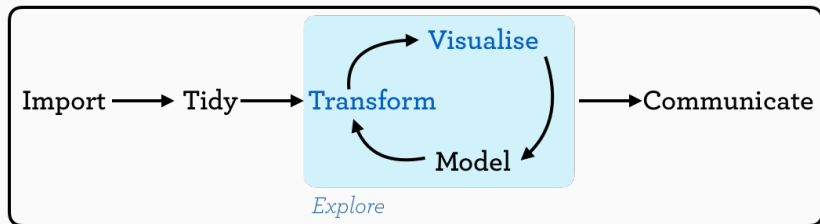
IFP Énergies Nouvelles

Juin 2025

Le tidyverse : vue d'ensemble

Les grands principes

C'est quoi les "Data Sciences"?



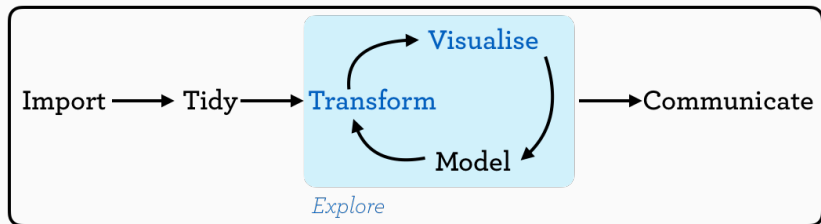
Program

Import :

- ▶ readr
- ▶ readxl
- ▶ haven
- ▶ httr
- ▶ rvest
- ▶ xml2

Les grands principes

C'est quoi les "Data Sciences"?



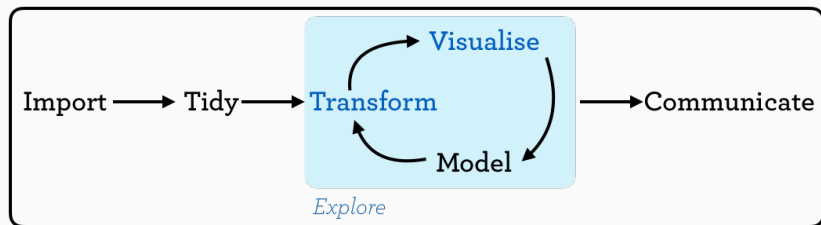
Program

Tidy :

- ▶ tibble
- ▶ tidyr

Les grands principes

C'est quoi les "Data Sciences"?



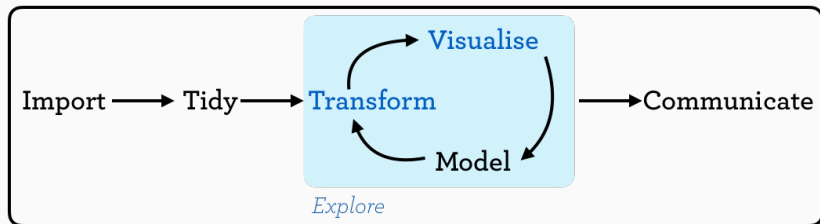
Program

Transform :

- ▶ dplyr
- ▶ forcats
- ▶ hms
- ▶ lubridate
- ▶ stringr

Les grands principes

C'est quoi les "Data Sciences"?



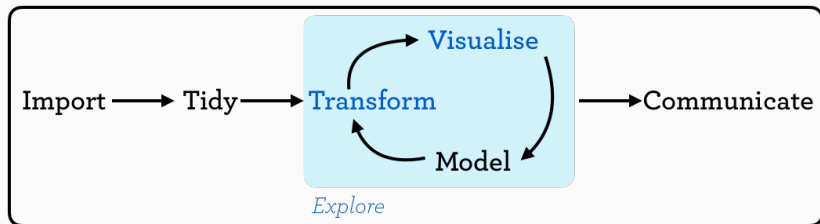
Program

Visualise :

► ggplot2

Les grands principes

C'est quoi les "Data Sciences"?



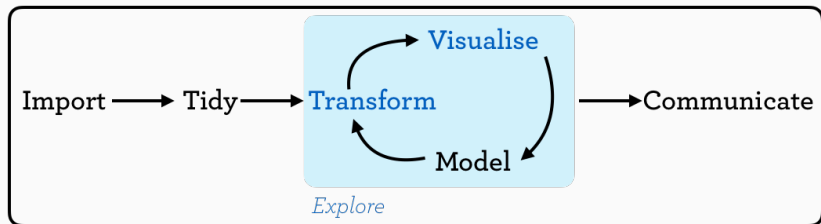
Program

Model :

- ▶ broom
- ▶ modelr

Les grands principes

C'est quoi les "Data Sciences"?



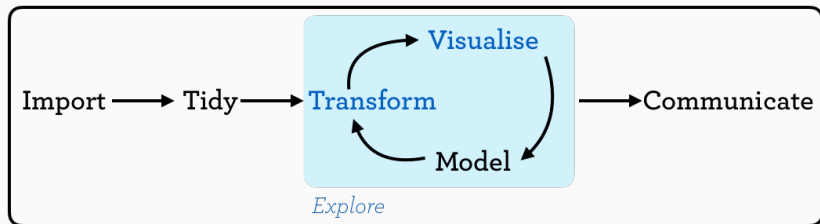
Program

Communicate :

- ▶ knitr
- ▶ rmarkdown
- ▶ blogdown
- ▶ bookdown
- ▶ shiny

Les grands principes

C'est quoi les "Data Sciences"?



Program

Program :

- ▶ purrr
- ▶ magritr

Visualisation :

1. La *g*rammaire des *g*raphiques

1. La grammaire des graphiques

Installation des packages nécessaires

```
## install.packages("tidyverse")  
## install.packages("nycflights13")  
library(tidyverse)  
library(nycflights13)
```

1. La grammaire des graphiques

Principes du package `ggplot2`

Pour faire un graphique avec `ggplot2`, il nous faut au moins les 3 ingrédients suivants :

- ▶ **`data`** : le nom d'un tableau contenant les données
- ▶ **`geom`** : les objets géométriques que l'on souhaite voir apparaître sur le graphique (des points, des lignes, des boîtes à moustache...)
- ▶ **`aes`** : les attributs esthétiques des objets géométriques (position, taille, couleur, transparence...)

1. La grammaire des graphiques

Les données

Le tableau `flights` du package `nycflights13` :

```
flights
```

```
# A tibble: 336,776 x 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013	1	1	517	515	2	830
2	2013	1	1	533	529	4	850
3	2013	1	1	542	540	2	923
4	2013	1	1	544	545	-1	1004
5	2013	1	1	554	600	-6	812
6	2013	1	1	554	558	-4	740
7	2013	1	1	555	600	-5	913
8	2013	1	1	557	600	-3	709
9	2013	1	1	557	600	-3	838
10	2013	1	1	558	600	-2	753

```
# i 336,766 more rows
```

```
# i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,  
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,  
#   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,  
#   minute <dbl>, time_hour <dtm>
```

1. La grammaire des graphiques

Les données

Dans un premier temps nous travaillerons uniquement avec les vols de la compagnie **Alaska Airlines** :

```
# On filtre les données de la compagnie AS
alaska_flights <- flights %>%
  filter(carrier == "AS")
```

```
# On s'assure qu'on a un tableau beaucoup plus petit
dim(alaska_flights)

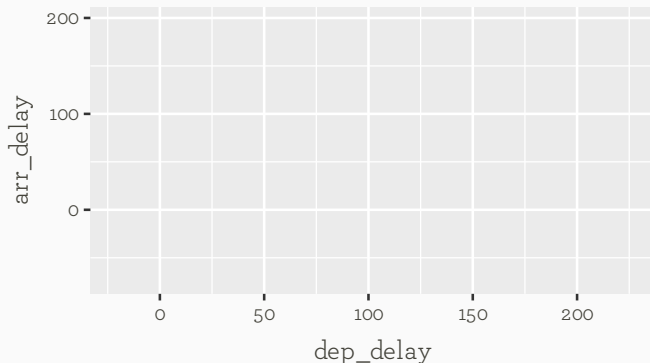
[1] 714  19
```

1. La grammaire des graphiques

Premier graphique

Les retards des vols au décollage et à l'atterrissage sont-ils liés?

```
ggplot(data = alaska_flights,  
       mapping = aes(x = dep_delay, y = arr_delay))
```

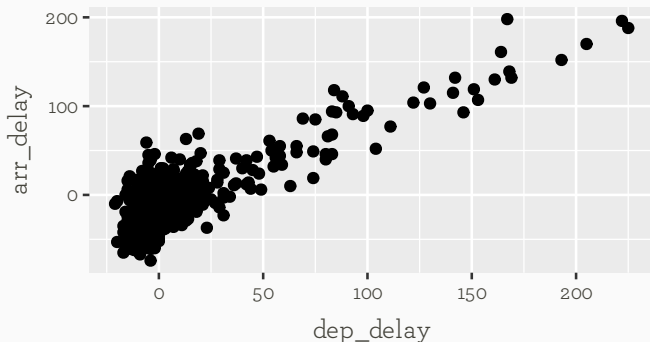


1. La grammaire des graphiques

Premier graphique

```
ggplot(data = alaska_flights,  
       mapping = aes(x = dep_delay, y = arr_delay)) +  
  geom_point()
```

Warning: Removed 5 rows containing missing values or values outside the scale
range (``geom_point()``).

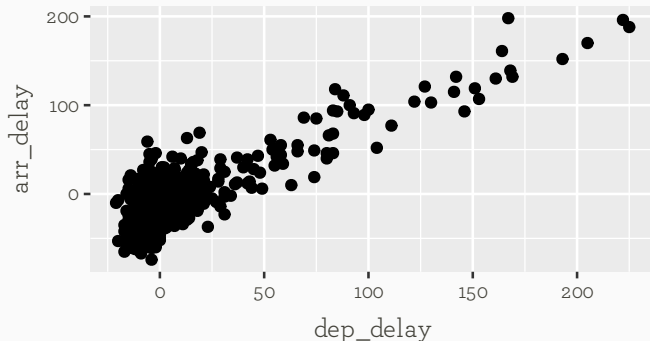


1. La grammaire des graphiques

Premier graphique

On peut se passer du nom des arguments data et mapping :

```
ggplot(alaska_flights, aes(x = dep_delay, y = arr_delay)) +  
  geom_point()
```



Visualisation :

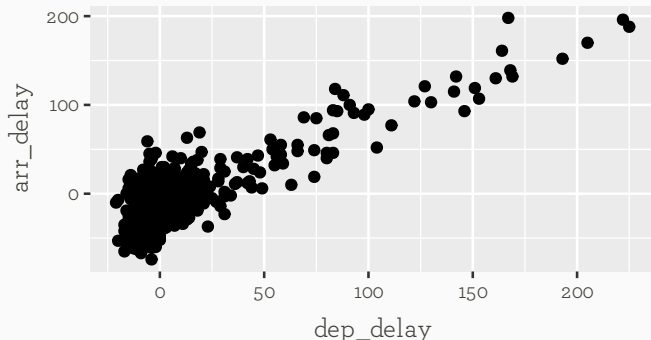
2. Nuages de points

2. Nuages de points

`geom_point()`

Les nuages de points sont créés grâce à la fonction `geom_point()` qui ajoute au graphique une couche contenant l'objet géométrique "points" :

```
ggplot(alaska_flights, aes(x = dep_delay, y = arr_delay)) +  
  geom_point()
```

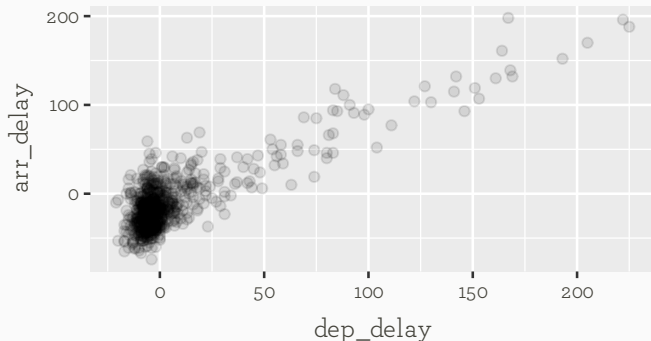


2. Nuages de points

Overplotting

Lorsque de nombreux points se superposent, on peut améliorer l'aspect du graphique en jouant sur la **transparence** des points :

```
ggplot(alaska_flights, aes(x = dep_delay, y = arr_delay)) +  
  geom_point(alpha = 0.1)
```

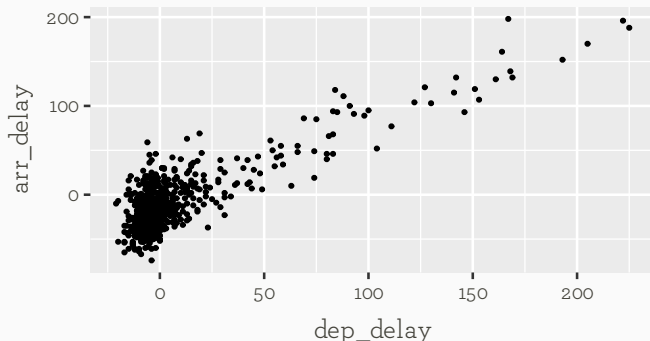


2. Nuages de points

Overplotting

Lorsque de nombreux points se superposent, on peut améliorer l'aspect du graphique en jouant sur la **taille** des points :

```
ggplot(alaska_flights, aes(x = dep_delay, y = arr_delay)) +  
  geom_point(size = 0.4)
```

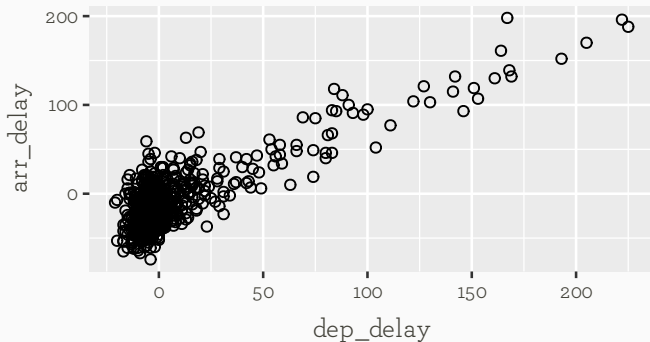


2. Nuages de points

Overplotting

Lorsque de nombreux points se superposent, on peut améliorer l'aspect du graphique en jouant sur le **symbole** utilisé :

```
ggplot(alaska_flights, aes(x = dep_delay, y = arr_delay)) +  
  geom_point(shape = 1)
```

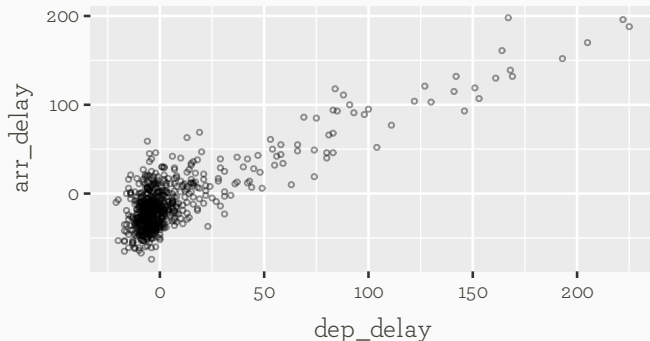


2. Nuages de points

Overplotting

Il est bien sûr possible de combiner plusieurs modifications :

```
ggplot(alaska_flights, aes(x = dep_delay, y = arr_delay)) +  
  geom_point(alpha = 0.4, size = 0.6, shape = 1)
```

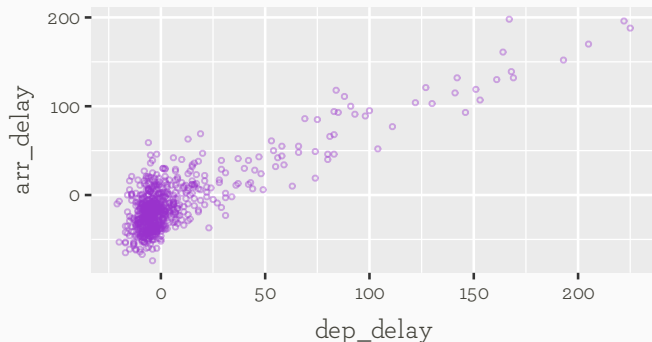


2. Nuages de points

Overplotting

Il est enfin possible de modifier la couleur des points :

```
ggplot(alaska_flights, aes(x = dep_delay, y = arr_delay)) +  
  geom_point(alpha = 0.4, size = 0.6,  
            shape = 1, color = "darkorchid")
```

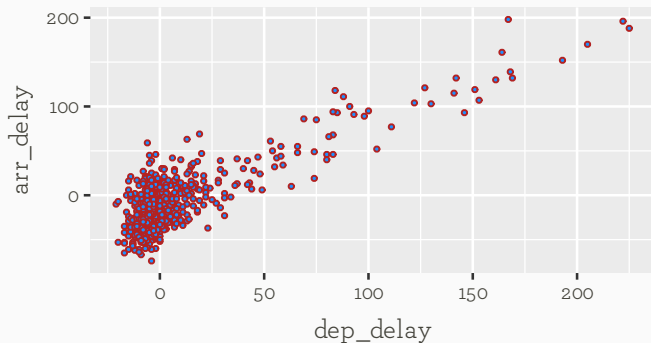


2. Nuages de points

Overplotting

Pour certains symboles, on fait la distinction entre couleur de **contour** et de **remplissage** :

```
ggplot(alaska_flights, aes(x = dep_delay, y = arr_delay)) +  
  geom_point(size = 0.6, shape = 21,  
            fill = "dodgerblue", color = "firebrick")
```

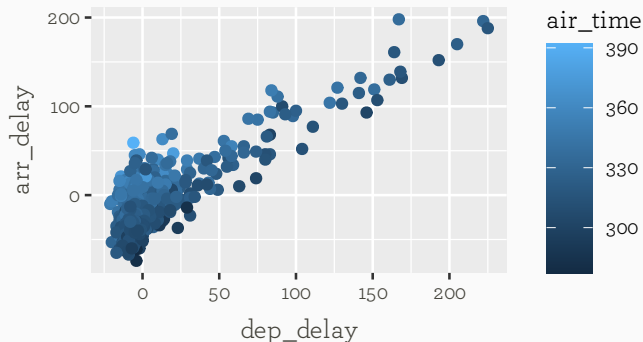


2. Nuages de points

Aesthetic mappings

Toutes ces options peuvent être associées à des variables :

```
ggplot(alaska_flights, aes(x = dep_delay, y = arr_delay,  
                           color = air_time)) +  
  geom_point()
```



2. Nuages de points

Aesthetic mappings

Est-ce que le jour de la semaine a une influence sur les retards des vols?

```
library(lubridate)
alaska_small <- alaska_flights %>%
  mutate(jour = wday(time_hour,
                     label = TRUE,
                     abbr = TRUE,
                     week_start = 1)) %>%
  select(dep_delay, arr_delay, jour)
```

2. Nuages de points

Aesthetic mappings

Est-ce que le jour de la semaine a une influence sur les retards des vols?

2. Nuages de points

Aesthetic mappings

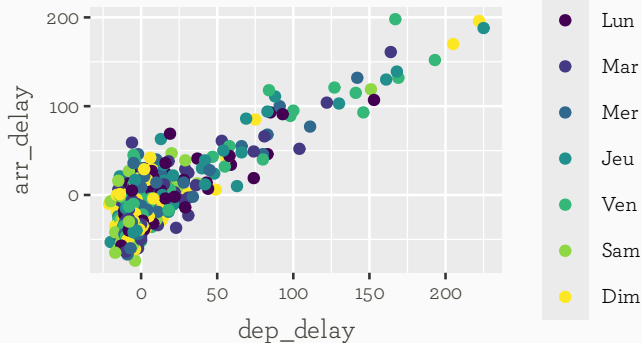
Est-ce que le jour de la semaine a une influence sur les retards des vols?

```
alaska_small  
  
# A tibble: 714 x 3  
  dep_delay arr_delay jour  
    <dbl>    <dbl> <ord>  
1      -1     -10 Mar  
2      -7     -19 Mar  
3      -3     -41 Mer  
4       3       1 Mer  
5      -1     -18 Jeu  
6       2      -9 Jeu  
7       0       1 Ven  
8      -7     -29 Ven  
9       0     -19 Sam  
10     -12     -12 Sam  
  
# i 704 more rows
```

2. Nuages de points

Aesthetic mappings

```
ggplot(alaska_small,  
       aes(x = dep_delay, y = arr_delay, color = jour)) +  
  geom_point()
```



2. Nuages de points

Aesthetic mappings

Les aesthetics possibles sont donc :

- ▶ x et y
- ▶ color (ou colour)
- ▶ fill
- ▶ shape
- ▶ size
- ▶ alpha

On peut placer ces arguments :

- ▶ **Dans** la fonction `aes()` pour associer une variable numérique ou catégorielle à ces caractéristiques esthétiques
- ▶ **En dehors** de la fonction `aes()` pour fixer manuellement la valeur voulue.

2. Nuages de points

Aesthetic mappings

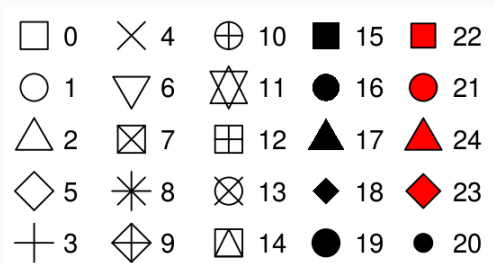
Lorsque l'on fixe ces éléments manuellement, il faut fournir des valeurs pertinentes :

color un nom de couleur, ou un numéro, ou un code hexadécimal, ou hsv, ou rgb, ou...

size un nombre de points en millimètres

alpha un degré d'opacité, compris entre 0 et 1

shape une valeur numérique comprise entre 0 et 24

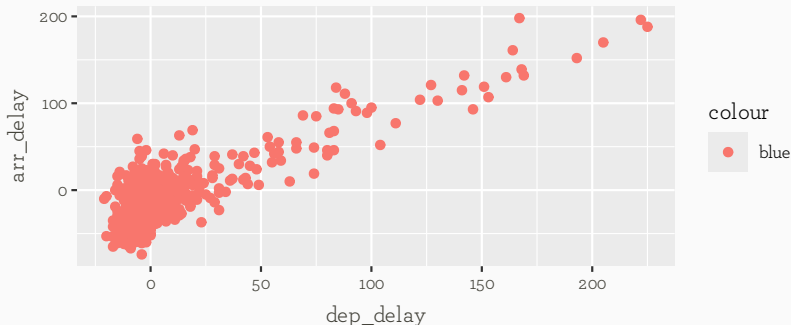


2. Nuages de points

Aesthetic mappings : exercices

Qu'est-ce qui ne va pas avec ce code? Pourquoi les points ne sont-ils pas bleus?

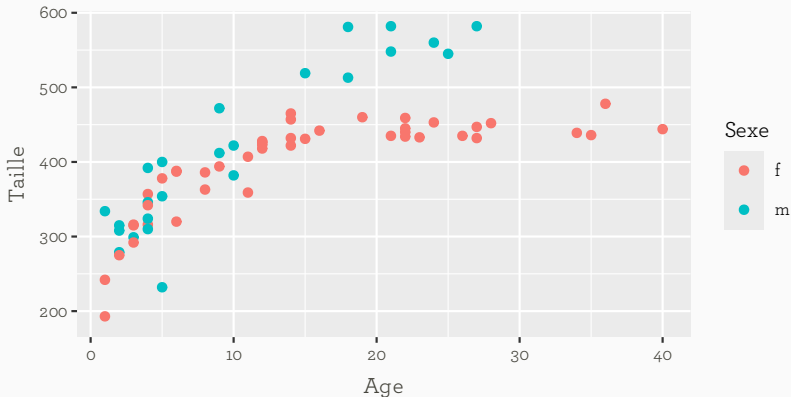
```
ggplot(alaska_flights,  
  aes(x = dep_delay, y = arr_delay, color = "blue")) +  
  geom_point()
```



2. Nuages de points

Aesthetic mappings : exercices

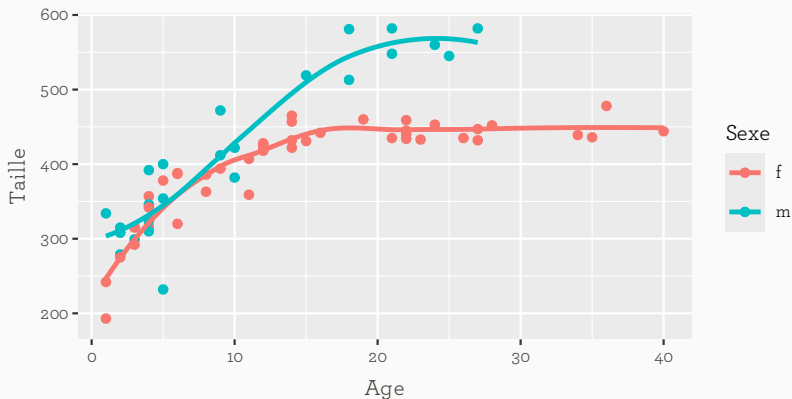
Avec les données contenues dans l'objet `dauphin`, faites le graphique suivant :



2. Nuages de points

Aesthetic mappings : exercices

En plus des points, ajoutez une couche supplémentaire à ce graphique en utilisant la fonction `geom_smooth()` :

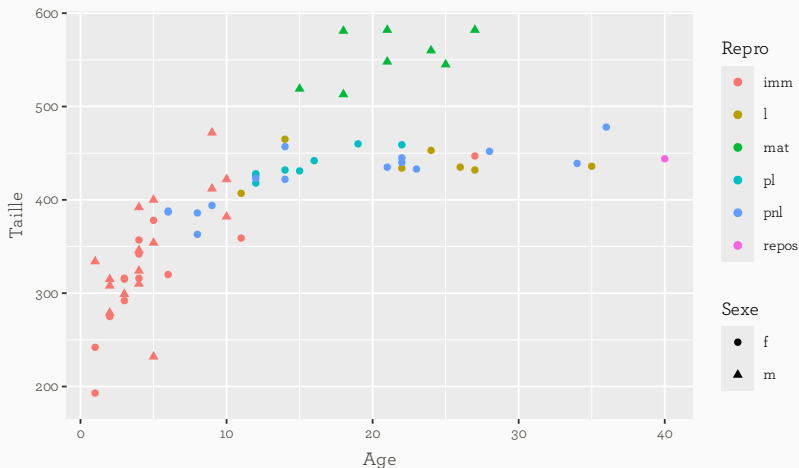


Que mettons-nous en évidence ici ?

2. Nuages de points

Aesthetic mappings : exercices

Toujours avec dauphin, créez maintenant ce nouveau graphique :



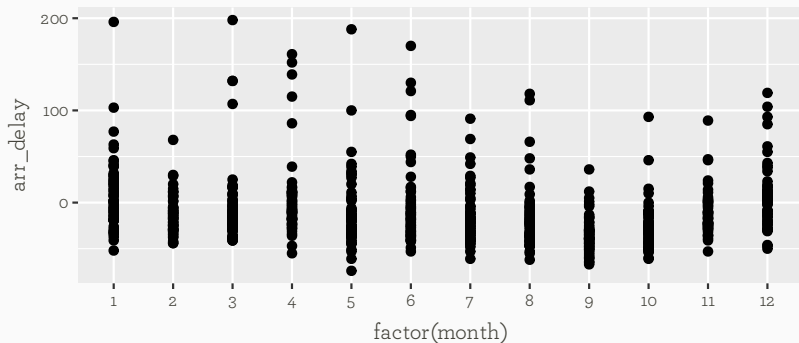
R nous dit qu'il a retiré 25 points. Pourquoi ?

2. Nuages de points

`geom_jitter()`

To jitter = trembler / être nerveux

```
ggplot(alaska_flights,  
       aes(x = factor(month), y = arr_delay)) +  
  geom_point()
```

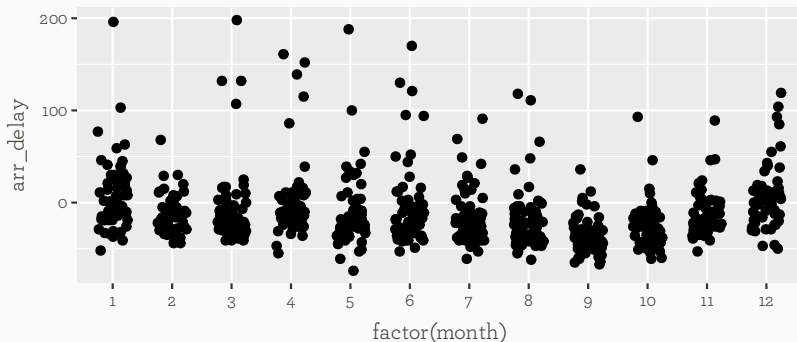


2. Nuages de points

`geom_jitter()`

To jitter = trembler / être nerveux

```
ggplot(alaska_flights,  
  aes(x = factor(month), y = arr_delay)) +  
  geom_jitter(height = 0, width = 0.25)
```



Visualisation :

3. Graphiques en lignes

3. Graphiques en lignes

Données météo

Nous allons maintenant travailler avec le jeu de données `weather` du package `nycflights13`.

- ▶ Examinez ce jeu de données
- ▶ Quelles sont les variables disponibles

Extraction des données météo de l'aéroport de Newark sur les 15 premiers jours de janvier :

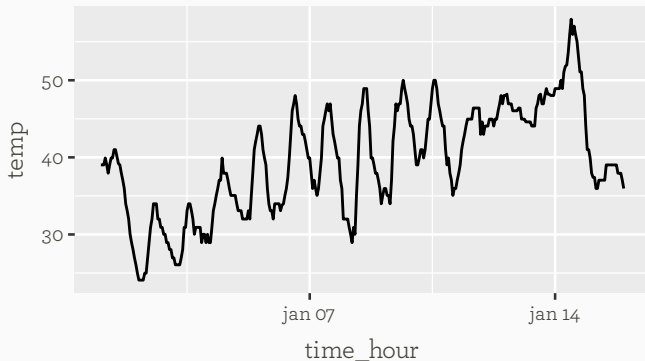
```
small_weather <- weather %>%  
  filter(origin == "EWR",  
         month == 1,  
         day <= 15)
```


3. Graphiques en lignes

`geom_line()`

Températures horaires à l'aéroport de Newark :

```
ggplot(small_weather, aes(x = time_hour, y = temp)) +  
  geom_line()
```

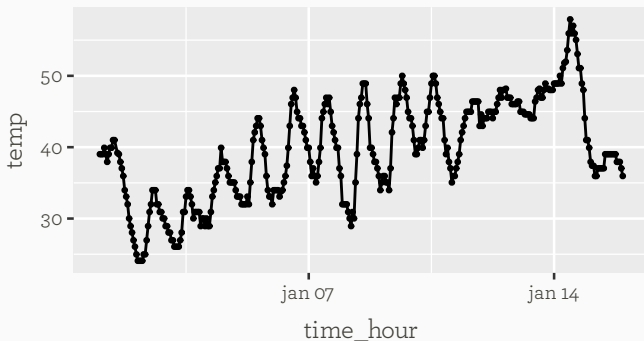


3. Graphiques en lignes

`geom_line()`

Il est tout à fait possible d'empiler les couches :

```
ggplot(small_weather, aes(x = time_hour, y = temp)) +  
  geom_line() +  
  geom_point(size = 0.5)
```

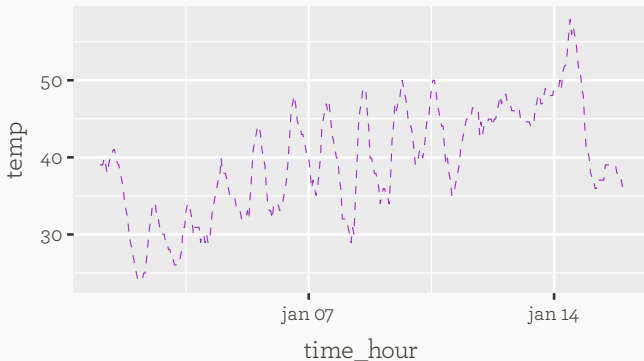


3. Graphiques en lignes

`geom_line()`

Comme pour `geom_point()`, nous pouvons modifier l'aspect de la ligne :

```
ggplot(small_weather, aes(x = time_hour, y = temp)) +  
  geom_line(size = 0.2, color = "darkorchid", linetype = 2)
```



3. Graphiques en lignes

`geom_line()`

Comme pour `geom_point()`, nous pouvons associer des variables aux caractéristiques esthétiques des lignes.

Pour l'illustrer, nous allons créer un nouveau jeu de données :

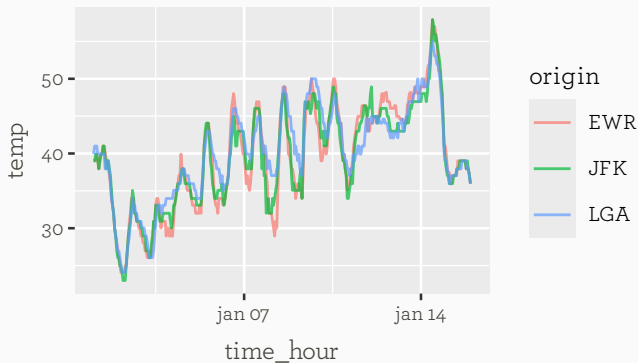
```
small_weather2 <- weather %>%  
  filter(month == 1,  
         day <= 15)
```

Selon vous, quelle est la différence entre `small_weather` et `small_weather2`?

3. Graphiques en lignes

`geom_line()`

```
ggplot(small_weather2,  
      aes(x = time_hour, y = temp, color = origin)) +  
  geom_line(alpha = 0.7)
```

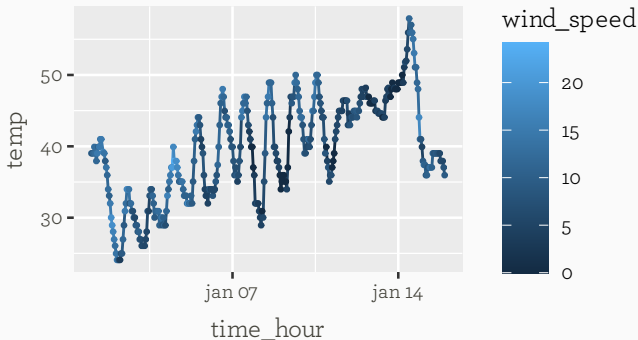


3. Graphiques en lignes

Où placer `aes()` ?

Comparer les 3 graphiques suivants et les commandes associées :

```
ggplot(small_weather,  
      aes(x = time_hour, y = temp, color = wind_speed)) +  
  geom_line() +  
  geom_point(size = 0.5)
```

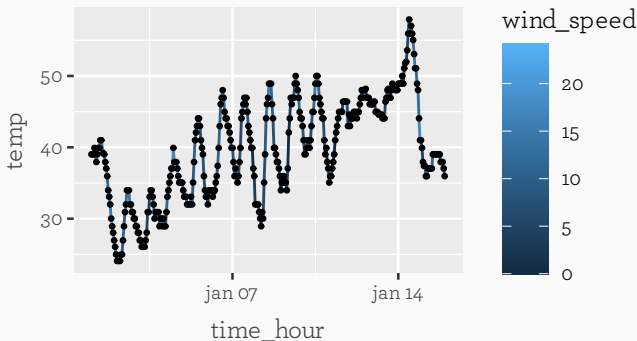


3. Graphiques en lignes

Où placer `aes()` ?

Comparer les 3 graphiques suivants et les commandes associées :

```
ggplot(small_weather,
      aes(x = time_hour, y = temp)) +
  geom_line(aes(color = wind_speed)) +
  geom_point(size = 0.5)
```

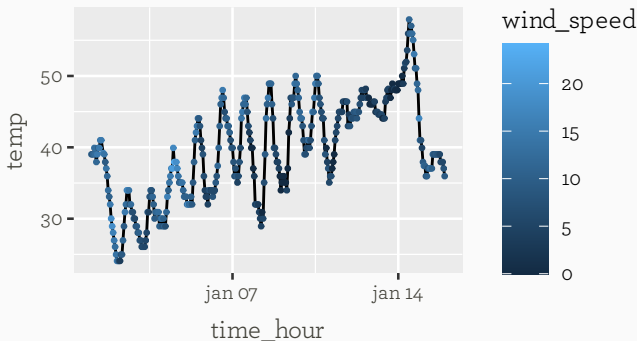


3. Graphiques en lignes

Où placer `aes()` ?

Comparer les 3 graphiques suivants et les commandes associées :

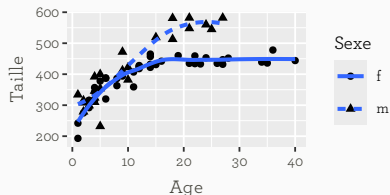
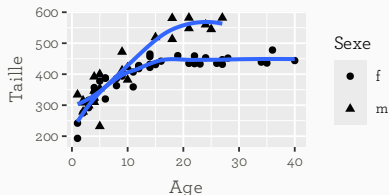
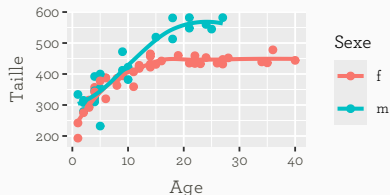
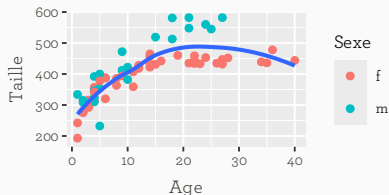
```
ggplot(small_weather,  
      aes(x = time_hour, y = temp)) +  
  geom_line() +  
  geom_point(aes(color = wind_speed), size = 0.5)
```



3. Graphiques en lignes

Où placer `aes()` ? Exercice

Avec le jeu de données `dauphin`, créez les 4 graphiques ci-dessous¹ :



1. Vous aurez besoin de la fonction `geom_smooth()` et de l'esthétique `group`.

Visualisation :

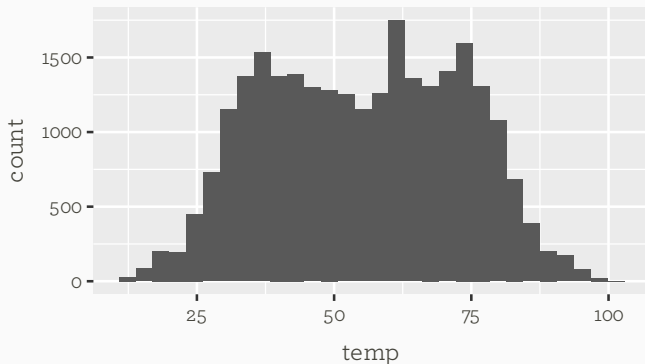
4. Les histogrammes

4. Les histogrammes

`geom_histogram()`

Distribution des températures horaires enregistrées en 2013 dans les 3 aéroports de New York :

```
ggplot(weather, aes(x = temp)) +  
  geom_histogram()
```

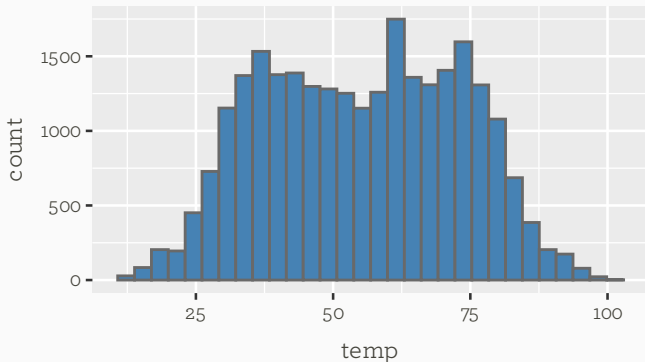


4. Les histogrammes

`geom_histogram()`

Comme d'habitude, on peut jouer sur un certain nombre de caractéristiques esthétiques :

```
ggplot(weather, aes(x = temp)) +  
  geom_histogram(fill = "steelblue", color = "dimgrey")
```



4. Les histogrammes

La taille des classes

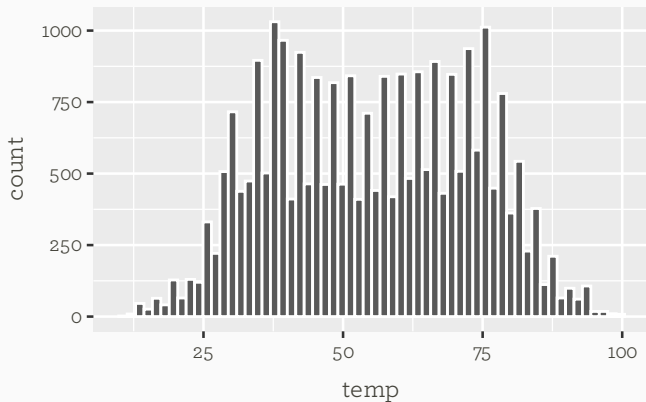
On peut spécifier manuellement la largeur des catégories de 3 façons différentes :

1. En ajustant le nombre de classes avec `bins`.
2. En précisant la largeur des classes avec `binwidth`.
3. En fournissant manuellement les limites des classes avec `breaks`.

4. Les histogrammes

La taille des classes

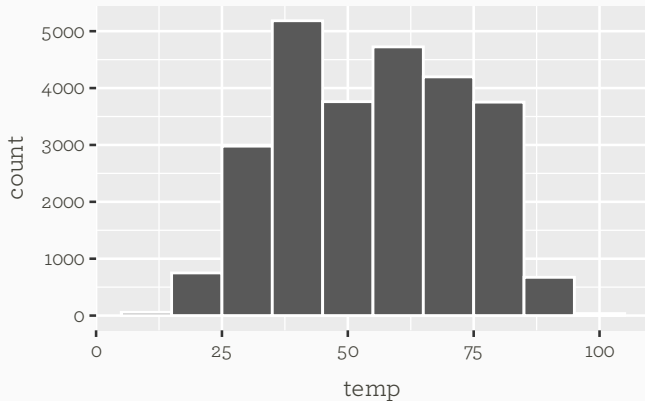
```
ggplot(weather, aes(x = temp)) +  
  geom_histogram(bins = 60, color = "white")
```



4. Les histogrammes

La taille des classes

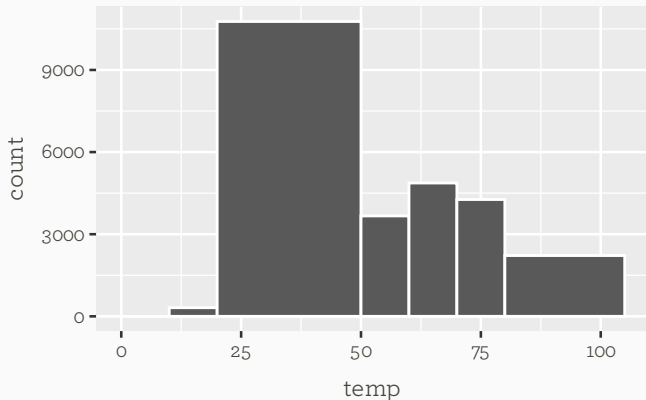
```
ggplot(weather, aes(x = temp)) +  
  geom_histogram(binwidth = 10, color = "white")
```



4. Les histogrammes

La taille des classes

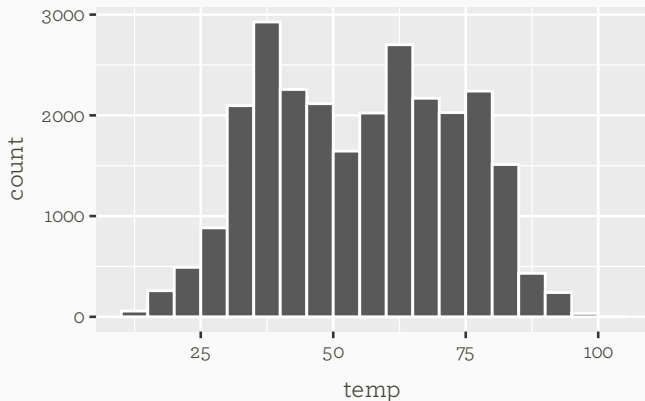
```
ggplot(weather, aes(x = temp)) +  
  geom_histogram(breaks = c(0, 10, 20, 50, 60, 70, 80, 105),  
                 color = "white")
```



4. Les histogrammes

La taille des classes

```
limits <- seq(from = 10, to = 105, by = 5)  
ggplot(weather, aes(x = temp)) +  
  geom_histogram(breaks = limits, color = "white")
```

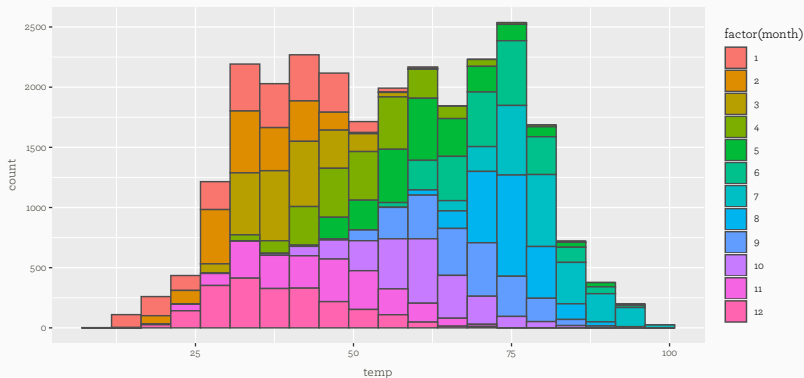


4. Les histogrammes

Variable supplémentaire

Comme pour les autres geom, il est possible d'associer une autre variable à une caractéristique esthétique de l'histogramme :

```
ggplot(weather, aes(x = temp, fill = factor(month))) +  
  geom_histogram(bins = 20, color = "grey30")
```



Visualisation :

5. Les facets

5. Les facets

Qu'est-ce que c'est ?

Une autre façon d'ajouter des variables supplémentaires est de séparer le graphiques en plusieurs facettes.

Definition

Un graphique composé de **facets** est un graphique composé de plusieurs sous-graphiques, chacun d'enre eux présentant une partie des données.

Deux fonctions principales permettent de créer des facets :

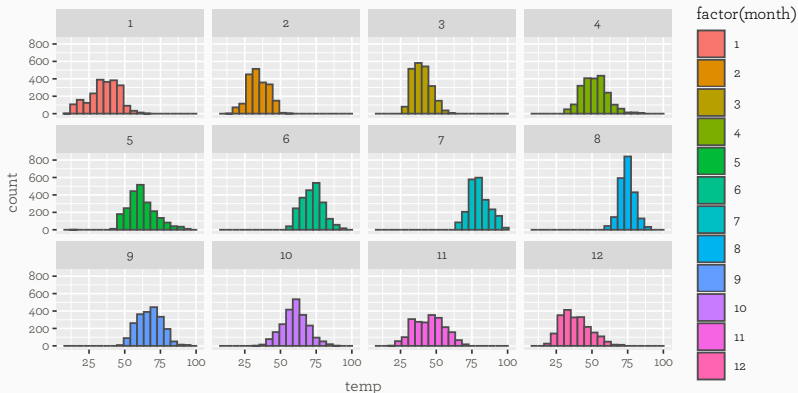
1. **facet_wrap()** pour séparer en fonction d'une seule variable
2. **facet_grid()** pour séparer en fonction de deux variables

Les variables utilisées pour “facetter” un graphique doivent être **catégorielles**.

5. Les facets

Exemple

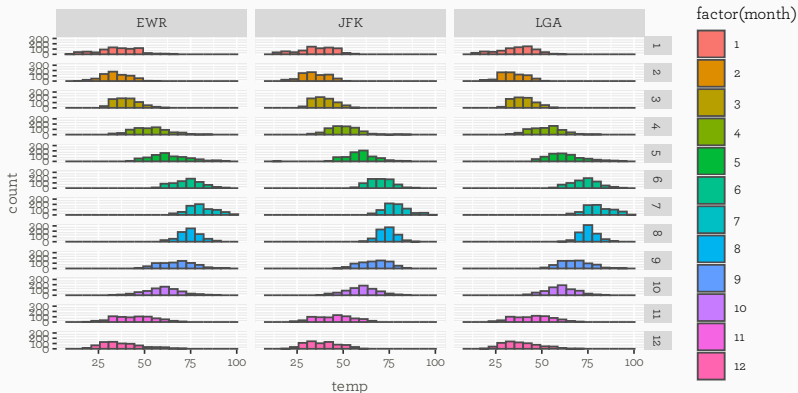
```
ggplot(weather, aes(x = temp, fill = factor(month))) +  
  geom_histogram(bins = 20, color = "grey30") +  
  facet_wrap(~factor(month), ncol = 4)
```



5. Les facets

Exemple

```
ggplot(weather, aes(x = temp, fill = factor(month))) +  
  geom_histogram(bins = 20, color = "grey30") +  
  facet_grid(factor(month) ~ origin)
```



Visualisation :

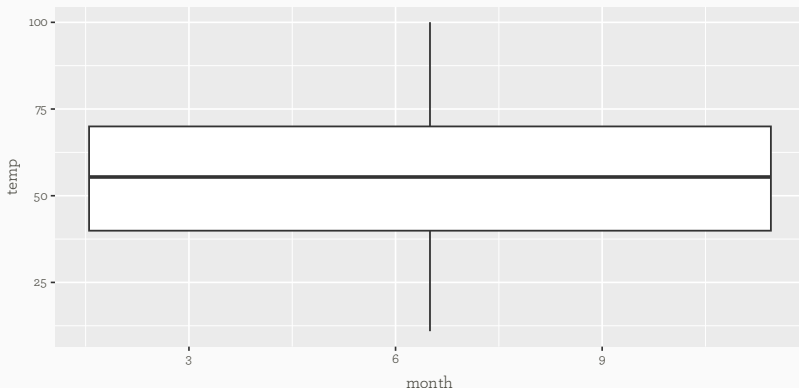
6. Les **boxplots**

6. Les boxplots

`geom_boxplot()`

Toujours avec weather, un boxplot fort peu utile...

```
ggplot(weather, aes(x = month, y = temp)) +  
  geom_boxplot()
```

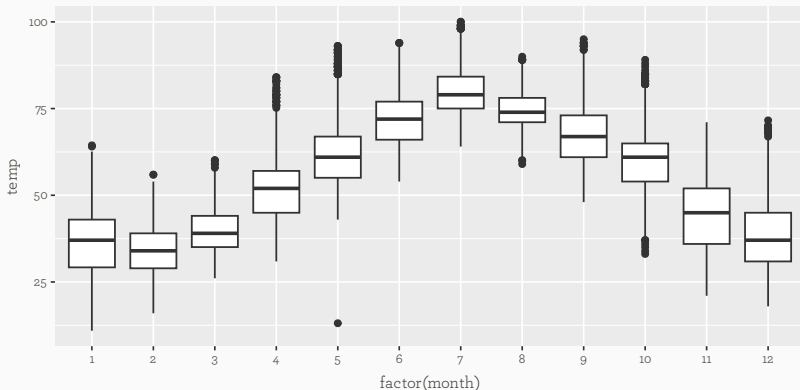


6. Les boxplots

`geom_boxplot()`

Les boxplots permettent de comparer la distribution d'une **variable numériques** pour plusieurs modalités d'une **variable catégorielle** :

```
ggplot(weather, aes(x = factor(month), y = temp)) +  
  geom_boxplot()
```

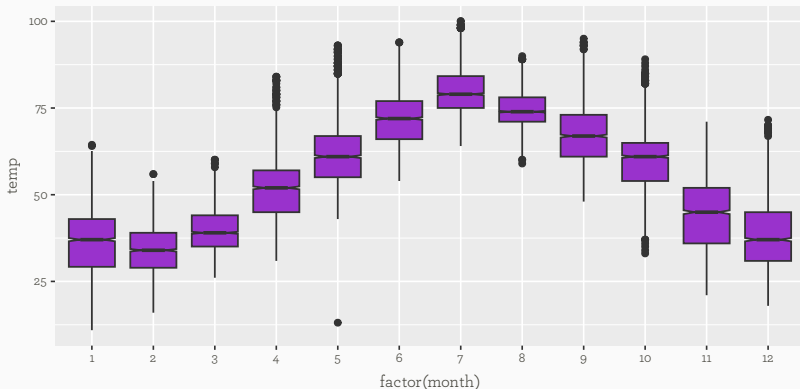


6. Les boxplots

`geom_boxplot()`

Ajout des intervalles de confiance à 95% des médianes :

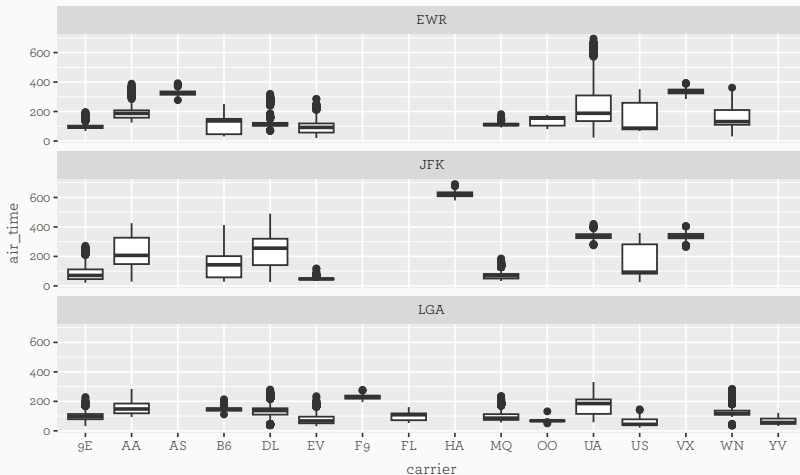
```
ggplot(weather, aes(x = factor(month), y = temp)) +  
  geom_boxplot(fill = "darkorchid", notch = TRUE)
```



6. Les boxplots

Exercice

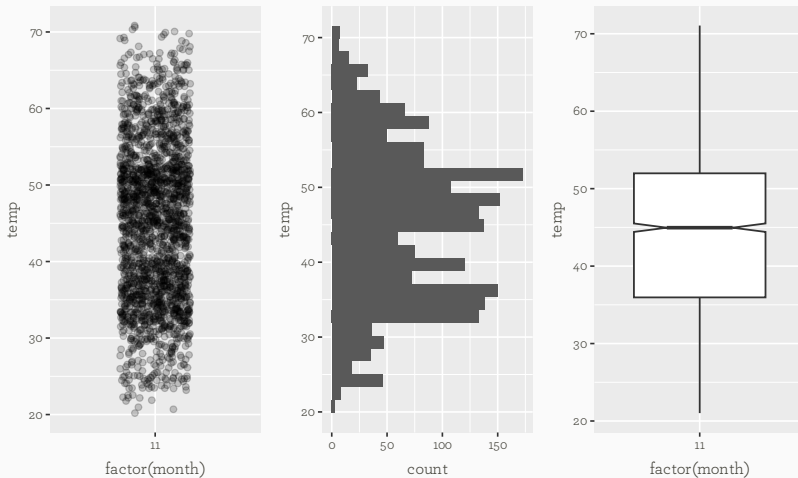
Avec le jeu de données `flights`, produisez le graphique suivant :



Que nous apprend-il?

6. Les boxplots

Une autre façon d'examiner des distributions



Visualisation :

7. Les diagrammes

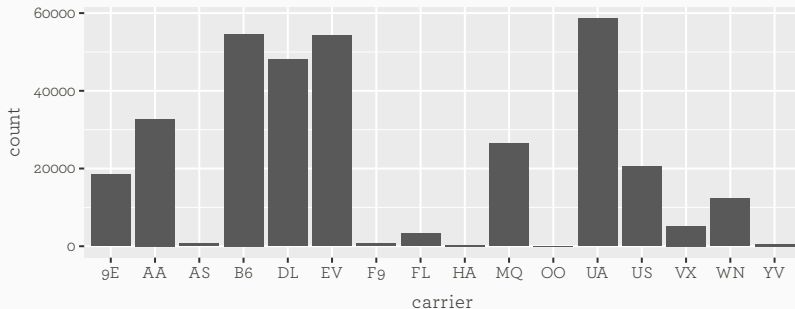
bâtons

7. Les diagrammes bâtons

`geom_bar()`

Revenons au jeu de données `flights`. Combien de vols ont été affrétés en 2013 par chaque compagnie aérienne ?

```
ggplot(flights, aes(x = carrier)) +  
  geom_bar()
```



7. Les diagrammes bâtons

`geom_bar()`

Affichez le contenu de la variable `carrier`.

- ▶ Selon vous, comment le graphique précédent a-t'il été produit?
- ▶ Quelle variable a été associée ("mappée") à l'axe des ordonnées?

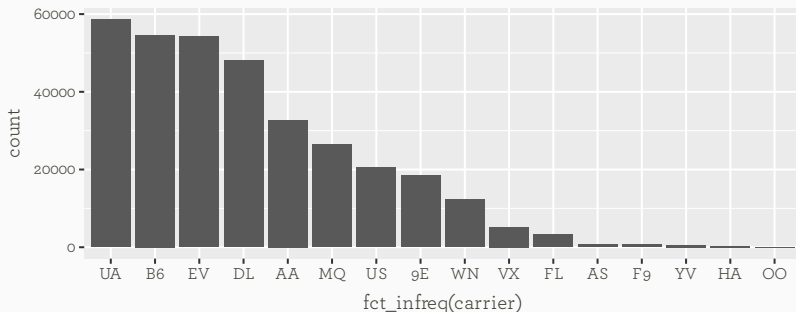
Si `carrier` est bien une variable de `flights`, `count` n'en est pas une.
D'où vient cette variable?

7. Les diagrammes bâtons

`geom_bar()`

Les diagrammes bâtons sont souvent plus parlants quand les catégories sont triées :

```
ggplot(flights, aes(x = fct_infreq(carrier))) +  
  geom_bar()
```



7. Les diagrammes bâtons

`geom_col()`

Parfois, les données brutes ne sont pas disponibles, et nous disposons seulement de données déjà comptées :

```
# On prend flights, puis...
carrier_table <- flights %>%
  # On groupe les données par compagnie, puis...
  group_by(carrier) %>%
  # On calcule le nb de vols par Cie, puis ...
  summarize(nombre = n()) %>%
  # On trie le tableau par nb de vols décroissant
  arrange(desc(nombre))
```

7. Les diagrammes bâtons

geom_col()

Parfois, les données brutes ne sont pas disponibles, et nous disposons seulement de données déjà comptées :

```
# Enfin, on affiche la nouvelle table  
carrier_table
```

```
# A tibble: 16 x 2
```

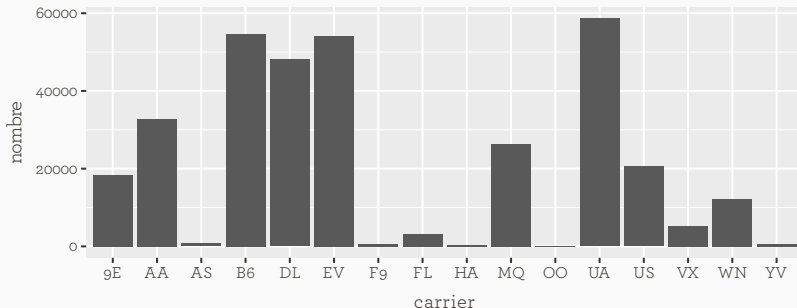
	carrier	nombre
	<chr>	<int>
1	UA	58665
2	B6	54635
3	EV	54173
4	DL	48110
5	AA	32729
6	MQ	26397
7	US	20536
8	9E	18460
9	WN	12275
10	VX	5162
11	FL	3260
12	AS	714
13	F9	685
14	YV	601
15	HA	342
16	00	32

7. Les diagrammes bâtons

`geom_col()`

Que faire si on ne dispose que de ces données “résumées”?

```
ggplot(carrier_table, aes(x = carrier, y = nombre)) +  
  geom_col()
```



Pourquoi les barres ne sont-elles pas ordonnées?

7. Les diagrammes bâtons

`geom_col()`

La table `carrier_table` est triée, mais pas les niveaux du facteur `carrier`:

```
factor(carrier_table$carrier)
```

```
[1] UA B6 EV DL AA MQ US 9E WN VX FL AS F9 YV HA OO  
Levels: 9E AA AS B6 DL EV F9 FL HA MQ OO UA US VX WN YV
```

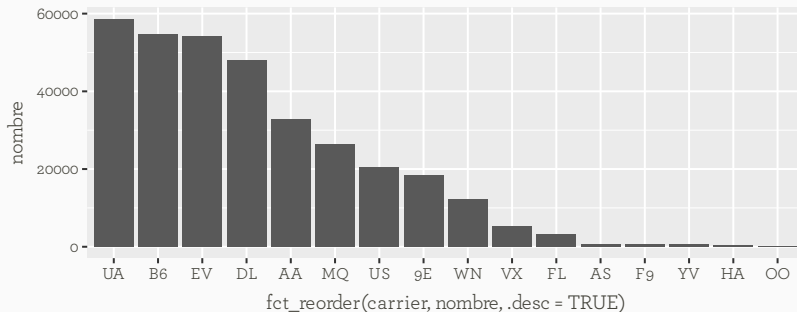
Les modalités sont toujours triées par ordre alphabétique.

Pour les modifier, il faut utiliser la fonction `fct_reorder()`.

7. Les diagrammes bâtons

`geom_col()`

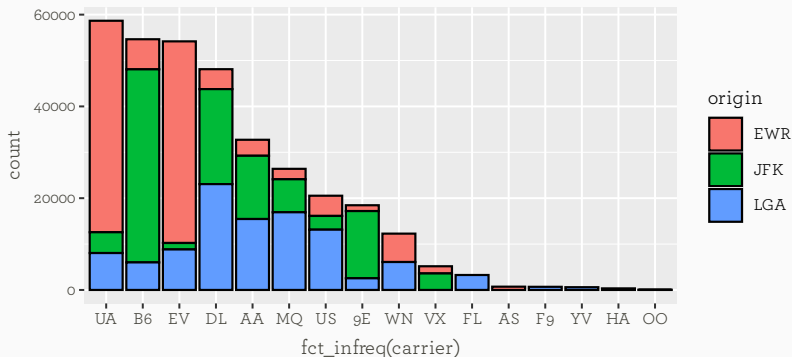
```
ggplot(carrier_table,  
  aes(x = fct_reorder(carrier, nombre, .desc = TRUE),  
      y = nombre)) +  
  geom_col()
```



7. Les diagrammes bâtons

Comparer 2 variables catégorielles

```
ggplot(flights, aes(x = fct_infreq(carrier), fill = origin)) +  
  geom_bar(color = "black")
```



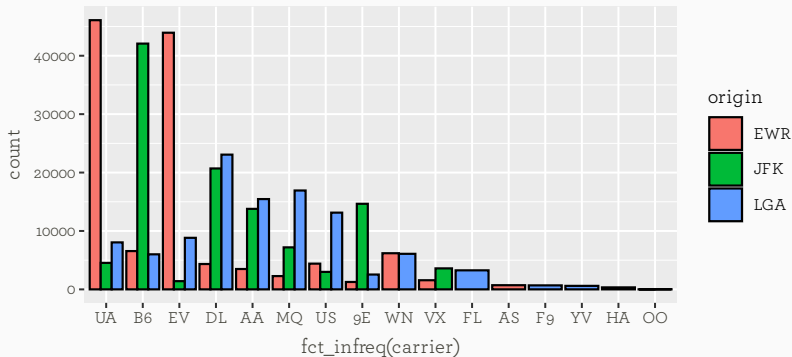
Il est rare que les barres empilées soient le choix le plus judicieux.

7. Les diagrammes bâtons

Comparer 2 variables catégorielles

Pour augmenter la lisibilité, on peut modifier la position des barres :

```
ggplot(flights, aes(x = fct_infreq(carrier), fill = origin)) +  
  geom_bar(color = "black", position = "dodge")
```

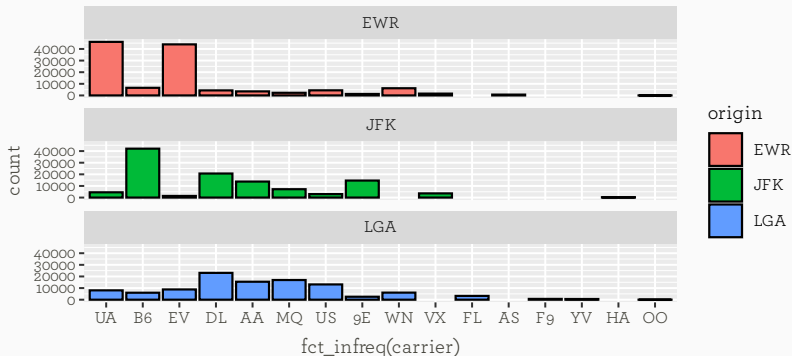


7. Les diagrammes bâtons

Comparer 2 variables catégorielles

Le plus simple reste souvent l'utilisation des facets :

```
ggplot(flights, aes(x = fct_infreq(carrier), fill = origin)) +  
  geom_bar(color = "black") +  
  facet_wrap(~origin, ncol = 1)
```



7. Les diagrammes bâtons

Comparer 2 variables catégorielles

Selon les types d'objets géométriques de vos graphiques, les choix possibles sont :

- ▶ “stacked”
- ▶ “identity”
- ▶ “dodge”
- ▶ “fill”
- ▶ “jitter”

Visualisation :
8. Systèmes de
coordonnées

8. Système de coordonnées

Les systèmes de coordonnées sont probablement l'une des parties les plus compliquées de ggplot2.

Par défaut, le système de **coordonnées cartésiennes** est utilisé. Les axes x et y agissent indépendamment pour déterminer la position de chaque point.

D'autres systèmes existent. Examinons les plus utiles :

- ▶ `coord_cartesian()`
- ▶ `coord_flip()`
- ▶ `coord_map()`, `coord_quickmap()`
- ▶ `coord_polar()`

8. Système de coordonnées

coord_flip()

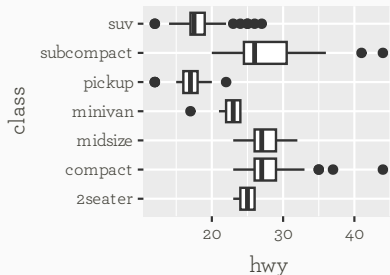
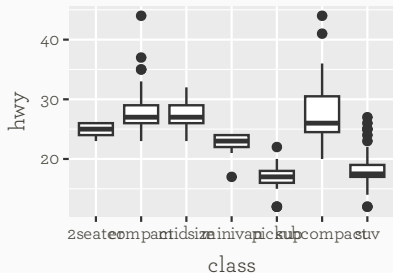
coord_flip() inverse la position des axes x et y :

Gauche

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

Droite

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



8. Système de coordonnées

`coord_map()`, `coord_quickmap()`

`coord_map()` ou `coord_quickmap()` sont utiles pour... faire des cartes!

```
fr <- map_data("france")
```

```
Error in `map_data()`:
```

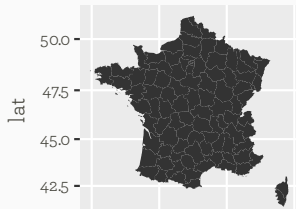
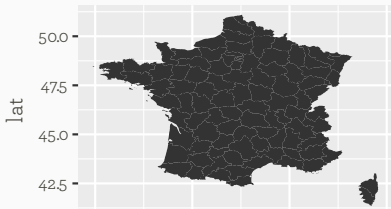
```
! The package "maps" is required for `map_data()`.
```

```
ggplot(fr, aes(long, lat, group = group)) + geom_polygon()
```

```
Error: objet 'fr' introuvable
```

```
ggplot(fr, aes(long, lat, group = group)) + geom_polygon() + coord_map()
```

```
Error: objet 'fr' introuvable
```



8. Système de coordonnées

coord_polar()

`coord_polar()` permet d'utiliser des coordonnées polaires. Il existe un lien entre **barplot** et **graphique de Coxcomb**.

Tapez les commandes suivantes pour visualiser ce lien :

```
bar <- ggplot(data = diamonds) +  
  geom_bar(  
    mapping = aes(x = cut, fill = cut),  
    show.legend = FALSE,  
    width = 1  
  ) +  
  theme(aspect.ratio = 1) +  
  labs(x = NULL, y = NULL)
```

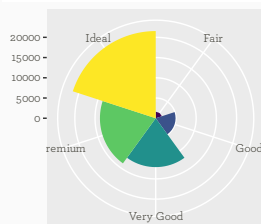
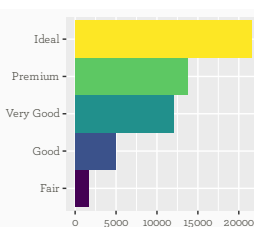
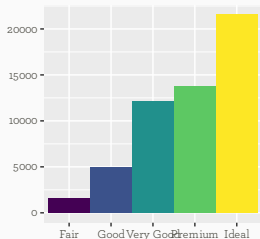
8. Système de coordonnées

coord_polar()

`coord_polar()` permet d'utiliser des coordonnées polaires. Il exise un lien entre **barplot** et **graphique de Coxcomb**.

Tapez les commandes suivantes pour visualiser ce lien :

```
bar # Gauche
bar + coord_flip() # Milieu
bar + coord_polar() # Droite
```



8. Système de coordonnées

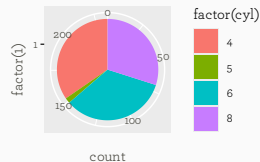
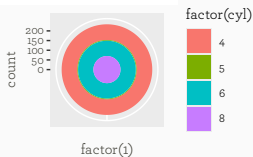
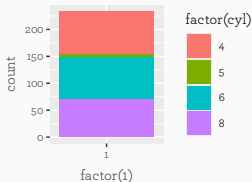
coord_polar()

`coord_polar()` permet d'utiliser des coordonnées polaires. Il exige aussi un lien entre barplot empilé et digramme camembert.

Tapez les commandes suivantes pour visualiser ce lien :

```
pie <- ggplot(mpg, aes(x = factor(1), fill = factor(cyl))) +  
  geom_bar(width=1)
```

```
pie                                     # Gauche  
pie + coord_polar()                   # Milieu  
pie + coord_polar(theta = "y")        # Droite
```

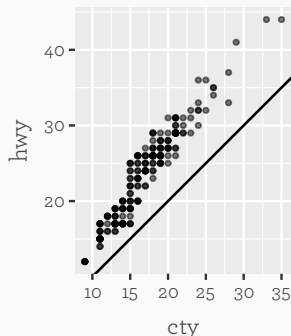


8. Système de coordonnées

Exercices

- ▶ Que nous apprend le graphique ci-dessous au sujet de la relation entre consommation en ville (cty) et consommation sur autoroute (hwy) ² ?
- ▶ Pourquoi est-il important d'utiliser `coord_fixed()` ?
- ▶ Que fait `geom_abline()` ?

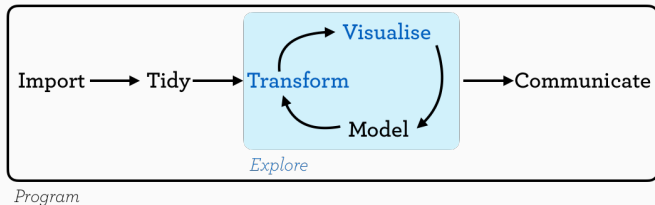
```
ggplot(data = mpg,  
       aes(x = cty, y = hwy)) +  
  geom_point(size = 0.7,  
            alpha = 0.5) +  
  geom_abline() +  
  coord_fixed()
```



2. Consultez l'aide de `mpg` pour savoir de quoi nous parlons.

Visualisation :
9. Des graphiques
présentables...

9. Des graphiques présentables



Jusqu'à maintenant, nous avons fait des graphiques pour **explorer** des jeux de données.

Il est tout aussi important d'être capable de réaliser des graphiques pour **communiquer** des résultats.

Pour cela, nous sommes souvent amenés à modifier l'**apparence** de nos graphiques. Nous aurons besoin des packages `ggrepel` et `viridis`. Installez-les dès maintenant.

9. Des graphiques présentables

Pour mettre en forme un graphique afin de communiquer clairement des résultats, il faut maîtriser au minimum les 3 éléments suivants :

1. Les labels avec la fonction `labs()`.
2. Les échelles avec notamment les fonctions `scale_XXX_YYY()`.
3. Les thèmes avec les fonctions `theme_XX()`.

Nous allons voir ensemble comment ça marche en reprenant des exemples de graphiques créés précédemment.