Une introduction à R

Marc-André Désautels

2017-10-05

Contents

Avant-propos

Part I

La présentation des données

Chapter 1

Introduction

1.1 Tibbles

Pour être en mesure d'effectuer des calculs statistiques, il nous faut une structure qui soit en mesure de garder en mémoire une base de données. Ces structures se nomment des "tibbles" dans R.

1.1.1 Prérequis

Pour être en mesure d'utiliser le paquetage **tibble**, nous devons charger le paquetage **tibble** et le paquetage **knitr**. Pour ce faire, il suffit d'utiliser la commande suivante:

```
library(tibble)
library(knitr)
```

Si vous exécutez ce code et vous recevez le message d'erreur suivant "there is no package called 'tibble' ", vous allez devoir installer le paquetage et ensuite charger la librairie.

```
install.packages("tibble")
library(tibble)
```

Vous faites la même chose pour le paquetage **knitr**.

Vous devez installer le paquetage une seule fois, mais vous devez le charger à chaque fois que vous démarrez une session en R.

1.1.2 Un exemple de "tibble"

Pour comprendre ce qu'est un "tibble", nous allons utiliser deux paquetages: "nycflights13" et "diamonds". Si ce n'est pas déjà fait, vous devez les installer et ensuite les charger.

```
library(nycflights13)
library(ggplot2)
```

Nous allons étudier le paquetage "nycflights13" qui contient 5 bases de données contenant des informations concernant les vols intérieurs en partance de New York en 2013, à partir des aéroports de Newark Liberty International (EWR), John F. Kennedy International (JFK) ou LaGuardia (LGA). Les 5 bases de données sont les suivantes:

- flights: information sur les 336,776 vols
- airlines: lien entre les codes IATA de deux lettres et les noms de compagnies d'aviation (16 au total)

- planes: information de construction sur les 3 322 avions utilisés
- weather: données météo à chaque heure (environ 8 710 observations) pour chacun des trois aéroports.
- airports: noms des aéroports et localisations

1.1.3 La base de données flights

Pour visualiser facilement une base de données sous forme "tibble", il suffit de taper son nom dans la console. Nous allons utiliser la base de données flights. Par exemple:

```
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
## # ... with 336,766 more rows, and 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 <dttm>
```

Nous allons décortiquer la sortie console:

- `A tibble: 336,776 x 19`: un "tibble" est une façon de représenter une base de données en R. Cette base de données possède:
  - `336 776` lignes
  - `19` colonnes correspondant aux 19 variables décrivant chacune des observations
- `year month day dep_time sched_dep_time dep_delay arr_time` sont différentes colonnes, en d'autres mots des variables, de cette base de données.
- Nous avons ensuite 10 lignes d'obervations correspondant à 10 vols
- `... with 336,766 more rows, and 12 more variables:` nous indique que 336 766 lignes et 12 autres variables ne pouvaient pas être affichées à l'écran.

Malheureusement cette sortie écran ne nous permet pas d'explorer les données correctement. Nous verrons à la section 1.1.5 comment explorer des `tibbles`.

1.1.4 La base de données `diamonds`

La base de données `diamonds` est composée des variables suivantes:

- `price` : prix en dollars US
- `carat` : poids du diamant en grammes
- `cut` : qualité de la coupe (Fair, Good, Very Good, Premium, Ideal)
- `color` : couleur du diamant (J (pire) jusqu'à D (meilleur))
- `clarity` : une mesure de la clarté du diamant (I1 (pire), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (meilleur))
- `x` : longueur en mm

- `y` : largeur en mm
- `z` : hauteur en mm
- `depth` : z / mean(x, y) = 2 * z / (x + y)
- `table` : largeur du dessus du diamant par rapport à son point le plus large

```
diamonds
```

```
## # A tibble: 53,940 x 10
##    carat       cut color clarity depth table price     x     y     z
##    <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23     Ideal     E     SI2  61.5    55   326  3.95  3.98  2.43
##  2  0.21   Premium     E     SI1  59.8    61   326  3.89  3.84  2.31
##  3  0.23      Good     E     VS1  56.9    65   327  4.05  4.07  2.31
##  4  0.29   Premium     I     VS2  62.4    58   334  4.20  4.23  2.63
##  5  0.31      Good     J     SI2  63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good     I    VVS1  62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good     H     SI1  61.9    55   337  4.07  4.11  2.53
##  9  0.22      Fair     E     VS2  65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good     H     VS1  59.4    61   338  4.00  4.05  2.39
## # ... with 53,930 more rows
```

1.1.5 Comment explorer des “tibbles”

Voici les façons les plus communes de comprendre les données se trouvant à l'intérieur d'un “tibble”:

```
1. En utilisant la fonction `View()` de RStudio.C'est la commande que nous utiliserons le plus fr?quemm
2. En utilisant la fonction `glimpse()` du paquetage knitr
3. En utilisant la fonction `kable()`
4. En utilisant l'opérateur `$` pour étudier une seule variable d'une base de données
```

1. `View()`:

Éxécutez `View(flights)` dans la console de RStudio et explorez la base de données obtenue.

Nous remarquons que chaque colonnes représentent une variable différente et que ces variables peuvent être de différents types. Certaines de ces variables, comme `distance`, `day` et `arr_delay` sont des variables dites quantitatives. Ces variables sont numériques par nature. D'autres variables sont dites qualitatives.

Si vous regardez la colonne à l'extrème-gauche de la sortie de `View(flights)`, vous verrez une colonne de nombres. Ces nombres représentent les numéros de ligne de la base de données. Si vous vous promenez sur une ligne de même nombre, par exemple la ligne 5, vous étudiez une unité statistique.

2. `glimpse`:

La seconde façon d'explorer une base de données est d'utiliser la fonction `glimpse()`. Cette fonction nous donne la majorité de l'information précédente et encore plus.

```
glimpse(flights)
```

```
## Observations: 336,776
## Variables: 19
## $ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013,...
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 55...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 60...
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2...
```

```
## $ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 7...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 7...
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -...
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV",...
## $ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79...
## $ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN...
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR"...
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL"...
## $ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138...
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 94...
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5,...
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013...
```

3. `kable()`:

La dernière façon d'étudier l'entièreté de la base de données est d'utiliser la fonction `kable()`. Nous allons explorer les codes des différentes compagnies d'aviation de deux façons.

```
airlines
```

```
## # A tibble: 16 x 2
##    carrier                         name
##      <chr>                        <chr>
##  1      9E           Endeavor Air Inc.
##  2      AA      American Airlines Inc.
##  3      AS        Alaska Airlines Inc.
##  4      B6             JetBlue Airways
##  5      DL        Delta Air Lines Inc.
##  6      EV    ExpressJet Airlines Inc.
##  7      F9      Frontier Airlines Inc.
##  8      FL AirTran Airways Corporation
##  9      HA      Hawaiian Airlines Inc.
## 10      MQ                   Envoy Air
## 11      OO       SkyWest Airlines Inc.
## 12      UA       United Air Lines Inc.
## 13      US             US Airways Inc.
## 14      VX              Virgin America
## 15      WN      Southwest Airlines Co.
## 16      YV          Mesa Airlines Inc.
```

```
kable(airlines)
```

carrier	name
9E	Endeavor Air Inc.
AA	American Airlines Inc.
AS	Alaska Airlines Inc.
B6	JetBlue Airways
DL	Delta Air Lines Inc.
EV	ExpressJet Airlines Inc.
F9	Frontier Airlines Inc.
FL	AirTran Airways Corporation
HA	Hawaiian Airlines Inc.
MQ	Envoy Air
OO	SkyWest Airlines Inc.
UA	United Air Lines Inc.
US	US Airways Inc.
VX	Virgin America
WN	Southwest Airlines Co.
YV	Mesa Airlines Inc.

À première vue, les deux sorties sont semblables sauf que la seconde est beaucoup plus agréable visuellement dans un document R Markdown.

4. L'opérateur `$`:

Finalement, l'opérateur `$` nous permet d'explorer une seule variable à l'intérieur d'une base de données. Par exemple, si nous désirons étudier la variable `name` de la base de données `airlines`, nous obtenons:

```
airlines$name
```

```
##  [1] "Endeavor Air Inc."           "American Airlines Inc."
##  [3] "Alaska Airlines Inc."        "JetBlue Airways"
##  [5] "Delta Air Lines Inc."        "ExpressJet Airlines Inc."
##  [7] "Frontier Airlines Inc."      "AirTran Airways Corporation"
##  [9] "Hawaiian Airlines Inc."      "Envoy Air"
## [11] "SkyWest Airlines Inc."       "United Air Lines Inc."
## [13] "US Airways Inc."             "Virgin America"
## [15] "Southwest Airlines Co."      "Mesa Airlines Inc."
```

1.2 Types de variables

Nous pouvons utiliser différents types de variables avec le langage `R`.

1.2.1 Variables qualitatives

Une variable qualitative est une variable dont les résultats possibles sont des **mots**. Les différents **mots** que peuvent prendre une telle variable sont appelées des **modalités**.

1.2.1.1 Variables qualitatives à échelle nominale

On observe ce type de variable lorsqu'il n'y a pas d'ordre croissant naturel dans les **modalités** de la variable. Par exemple, la variable *couleur des cheveux* est à échelle nominale. L'ordre « blonds, bruns, roux, noirs, autre » est un ordre aussi valable que « bruns, noirs, roux, blonds, autre ».

Dans la base de données `nycflights13`, la variable `origin` provenant des données `flights` est une variable qualitative nominale.

```
unique(flights$origin)
```

```
## [1] "EWR" "LGA" "JFK"
```

Autre que l'ordre alphabétique, nous n'avons pas d'autre ordre logique à imposer à l'aéroport d'origine des vols.

1.2.1.2 Variables qualitatives à échelle ordinale

On observe ce type de variable lorsqu'il existe un ordre croissant dans les modalités de la variable. Par exemple, la variable *degré de satisfaction* est à échelle ordinale. Il est possible de classer les modalités en ordre décroissant en écrivant : Très satisfait > Satisfait > Insatisfait > Très insatisfait.

Dans la base de données `diamonds`, la variable `cut` est une variable qualitative à échelle ordinale.

```
unique(diamonds$cut)
```

```
## [1] Ideal     Premium   Good      Very Good Fair
## Levels: Fair < Good < Very Good < Premium < Ideal
```

Nous remarquons que les modalités de cette variable possèdent un ordre. Cet ordre est indiqué par les symboles `<` dans la sortie `R`.

1.2.2 Variables quantitatives

Une variable quantitative est une variable dont les résultats possibles sont des **nombres**. Les différents nombres que peuvent prendre une telle variable sont appelées des **valeurs**.

1.2.2.1 Variables quantitatives discrètes

On observe ce type de variable lorsque les valeurs sont énumérables, c'est-à-dire lorsqu'il n'existe pas de valeur possible entre deux valeurs consécutives. Par exemple, la variable *nombre de cours suivis pendant cette session* est une variable quantitative discrète. Les valeurs de ces variables peuvent être : 3, 4, 5, 6, 7,... Il est impossible de suivre 4,6 cours durant une session.

Dans la base de données `nycflights13`, la variable `engines` provenant des données `planes` est une variable quantitative discrète. Cette variable représente le nombre de moteurs de l'avion en question.

```
unique(planes$engines)
```

```
## [1] 2 1 4 3
```

Dans la sortie `R` les valeurs ne sont pas en ordre croissant mais elles le seront lorsque nous les représenterons sous forme de tableau ou de graphique.

1.2.2.2 Variables quantitatives continues

On observe ce type de variable lorsqu'il existe une infinité de valeurs entre deux autres. Par exemple, la variable *masse d'un étudiant (en lbs)* est une variable quantitative continue. Entre 130 et 131 lbs, il existe une infinité de valeurs telles que 130,54 lbs.

Dans la base de données `diamonds`, nous allons observer la variable `carat`. Voici les 25 premiers éléments de ces valeurs.

```
diamonds$carat[1:25]
```

```
##  [1] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 0.30 0.23 0.22 0.31
## [15] 0.20 0.32 0.30 0.30 0.30 0.30 0.30 0.23 0.23 0.31 0.31
```

Chapter 2

Présentation des données

Pour débuter, nous allons charger les paquetages utiles:

```
library(dplyr)
library(ggplot2)
library(knitr)
```

Pour introduire la présentation des données, nous allons utiliser la base de données `mtcars` et la base de données `diamonds`, que nous avons utilisé à la section 1.1.4.

La base de données `mtcars` a été extraite du magazine Motor Trend de l'année 1974, et comprend la consommation d'essence et 10 autres aspects de design automobile pour 32 automobiles (modèles 1973-1974).

Les 11 variables de cette base de données sont:

- `mpg` : Miles/ (US) gallon
- `cyl` : Nombre de cylindres
- `disp` : Déplacement en pouces cube
- `hp` : Nombre de chevaux-vapeur
- `drat` : Ratio
- `wt` : Poids (1000 livres)
- `qsec` : Temps pour le quart de mile
- `vs` : V/S
- `am` : Transmission (0 = automatique, 1 = manuelle)
- `gear` : Nombre de vitesses
- `carb` : Nombre de carburateurs

```
mtcars
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
```

```
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

2.1 Variables qualitatives

2.1.1 Tableaux de fréquences

Nous pouvons représenter des variables qualitatives sous forme de tableau. Nous allons utiliser la commande `tabfreq`. Voici comment l'utiliser pour représenter la variable `cut` de la base de données `diamonds`.

```
tabfreq(diamonds$cut)
```

diamonds$cut	Fréquence	Fréquence relative	Fréquence relative cumulée
Fair	1610	0.030	0.030
Good	4906	0.091	0.121
Very Good	12082	0.224	0.345
Premium	13791	0.256	0.600
Ideal	21551	0.400	1.000
Total	53940	1.000	1.000

Nous pouvons également étudier la variable `clarity`.

```
tabfreq(diamonds$clarity)
```

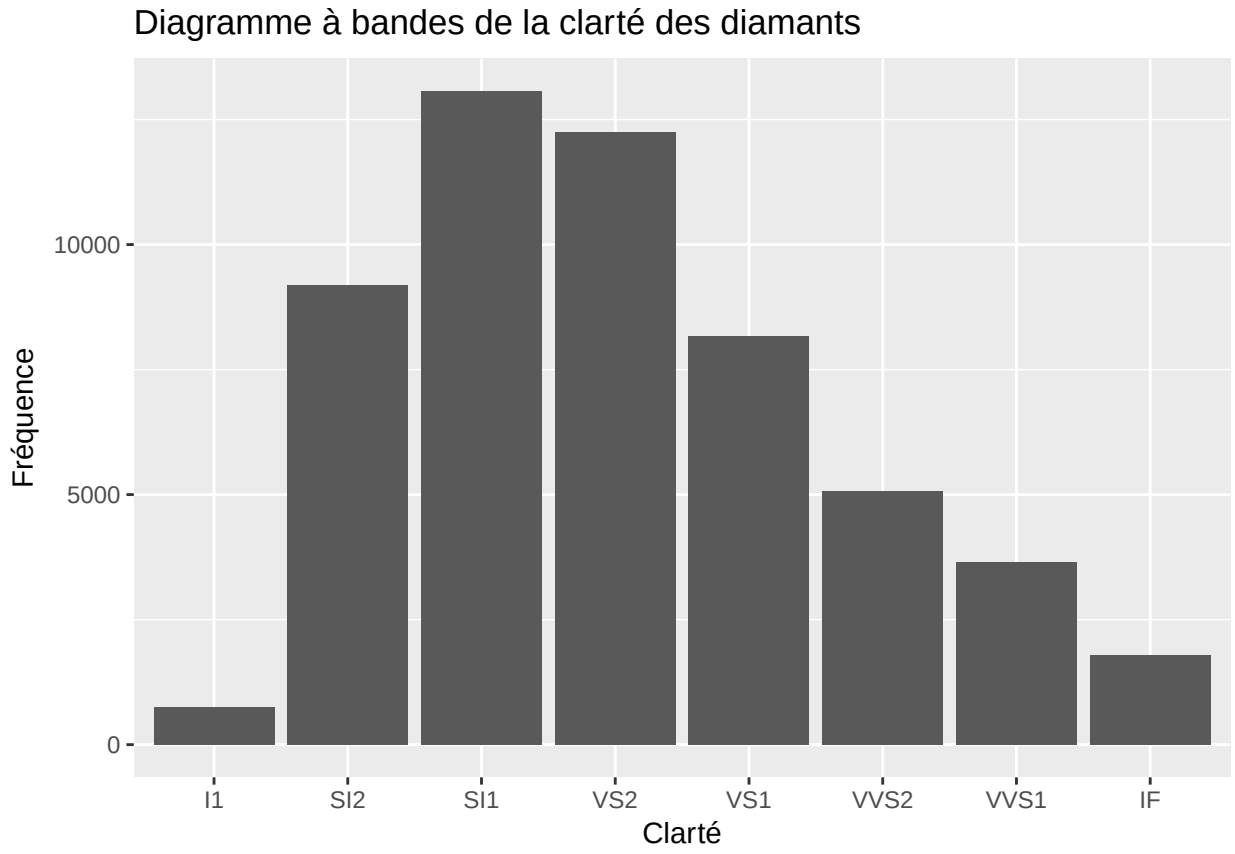
diamonds$clarity	Fréquence	Fréquence relative	Fréquence relative cumulée
I1	741	0.014	0.014
SI2	9194	0.170	0.184
SI1	13065	0.242	0.426
VS2	12258	0.227	0.654
VS1	8171	0.151	0.805
VVS2	5066	0.094	0.899
VVS1	3655	0.068	0.967
IF	1790	0.033	1.000
Total	53940	1.000	1.000

2.1.2 Diagramme à bandes

Pour les variables qualitatives, le diagramme à bandes est le graphique de choix.

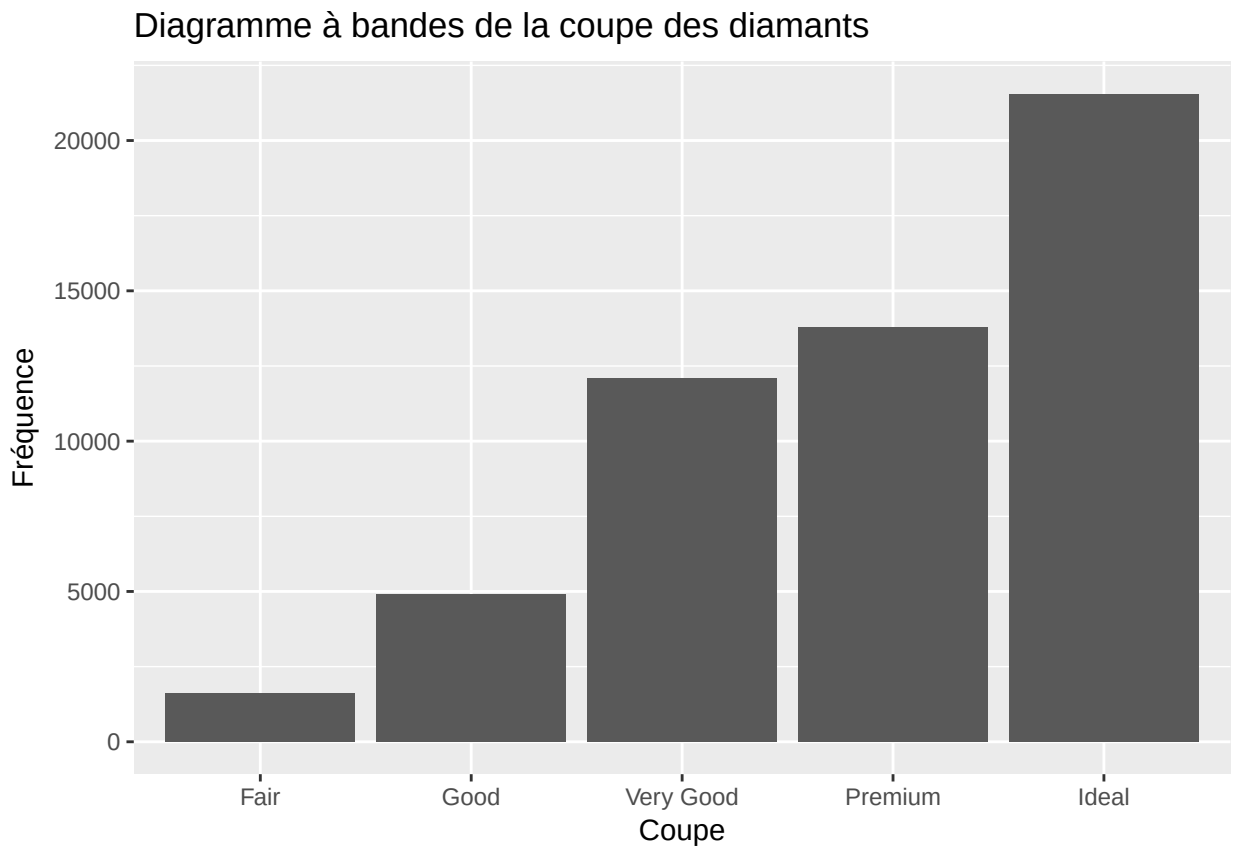
Pour la variable `clarity`.

```
ggplot(diamonds, aes(clarity)) + geom_bar() +
  labs(
    x = "Clarté",
    y = "Fréquence",
    title = "Diagramme à bandes de la clarté des diamants")
```



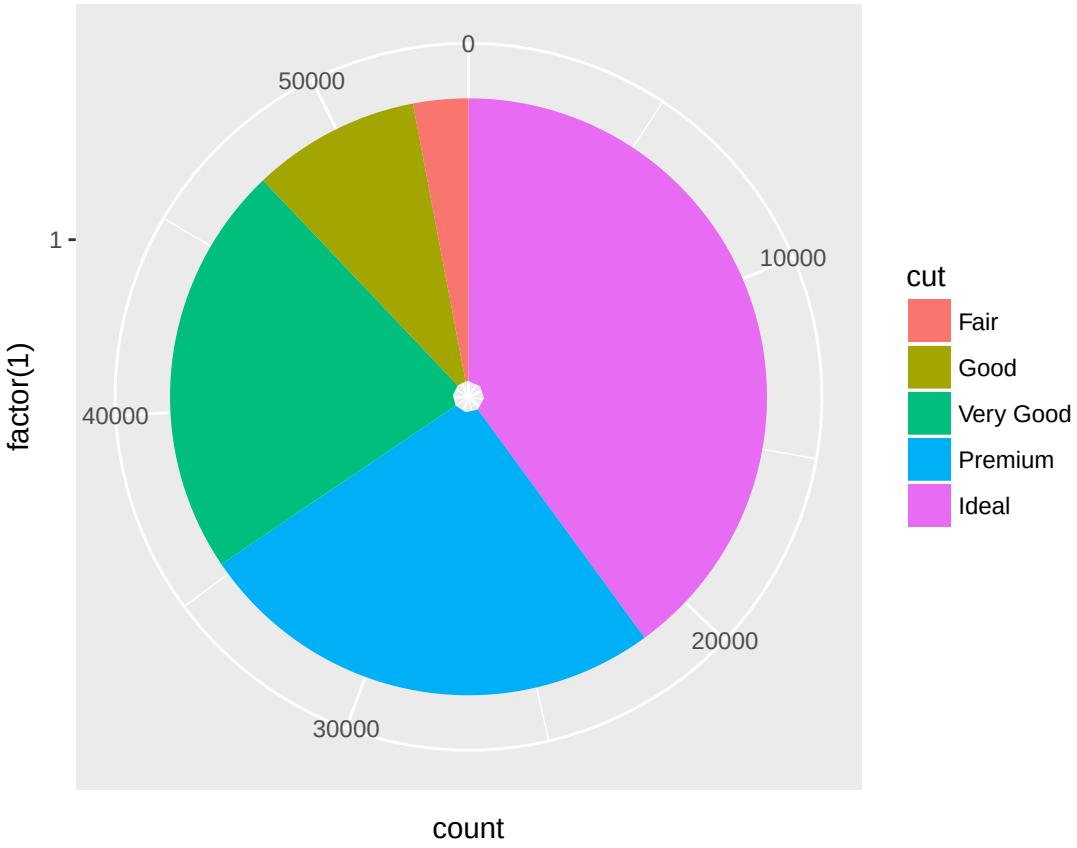
Pour la variable `cut`.

```
ggplot(diamonds, aes(cut)) + geom_bar() +
  labs(
    x = "Coupe",
    y = "Fréquence",
    title = "Diagramme à bandes de la coupe des diamants")
```



2.1.3 Diagramme circulaire

```
ggplot(diamonds, aes(x = factor(1), fill = cut)) +
  geom_bar() +
  coord_polar(theta = "y")
```



2.2 Variables quantitatives

2.2.1 Tableaux de fréquences

Pour une variable quantitative discrète, il suffit d'utiliser la fonction `tabfreq` pour représenter les données sous forme de tableau. Par exemple, pour la variable `cyl` de la base de données `mtcars`.

```
tabfreq(mtcars$cyl)
```

mtcars$cyl	Fréquence	Fréquence relative	Fréquence relative cumulée
4	11	0.344	0.344
6	7	0.219	0.562
8	14	0.438	1.000
Total	32	1.000	1.000

Pour représenter une variable quantitative continue sous forme de tableau, il faut effectuer un traitement préalable sur les données.

Étudions la variable `carat` de la base de données `diamonds`. Si nous tentons d'utiliser la commande `tabfreq` directement, nous allons obtenir une table beaucoup trop grande. En effet, la variable `carat` possède 273 valeurs différentes!

Pour représenter la variable correctement, nous allons débuter par observer l'étendue des valeurs possibles de cette variable en utilisant la commande `range`. Nous avons donc:

```
range(diamonds$carat)
```

```
## [1] 0.20 5.01
```

La sortie de R signifie que la valeur la plus petite de carat est 0.2, et que la plus grande est 5.01.

Nous voulons maintenant recoder notre variable carat pour obtenir des classes. Dans notre exemple, il semble adéquat de créer des classes de largeur 1 en débutant à 0 et en terminant à 6. L'option breaks permet de décider des classes et l'option right permet de fermer l'intervalle à gauche et de l'ouvrir à droite.

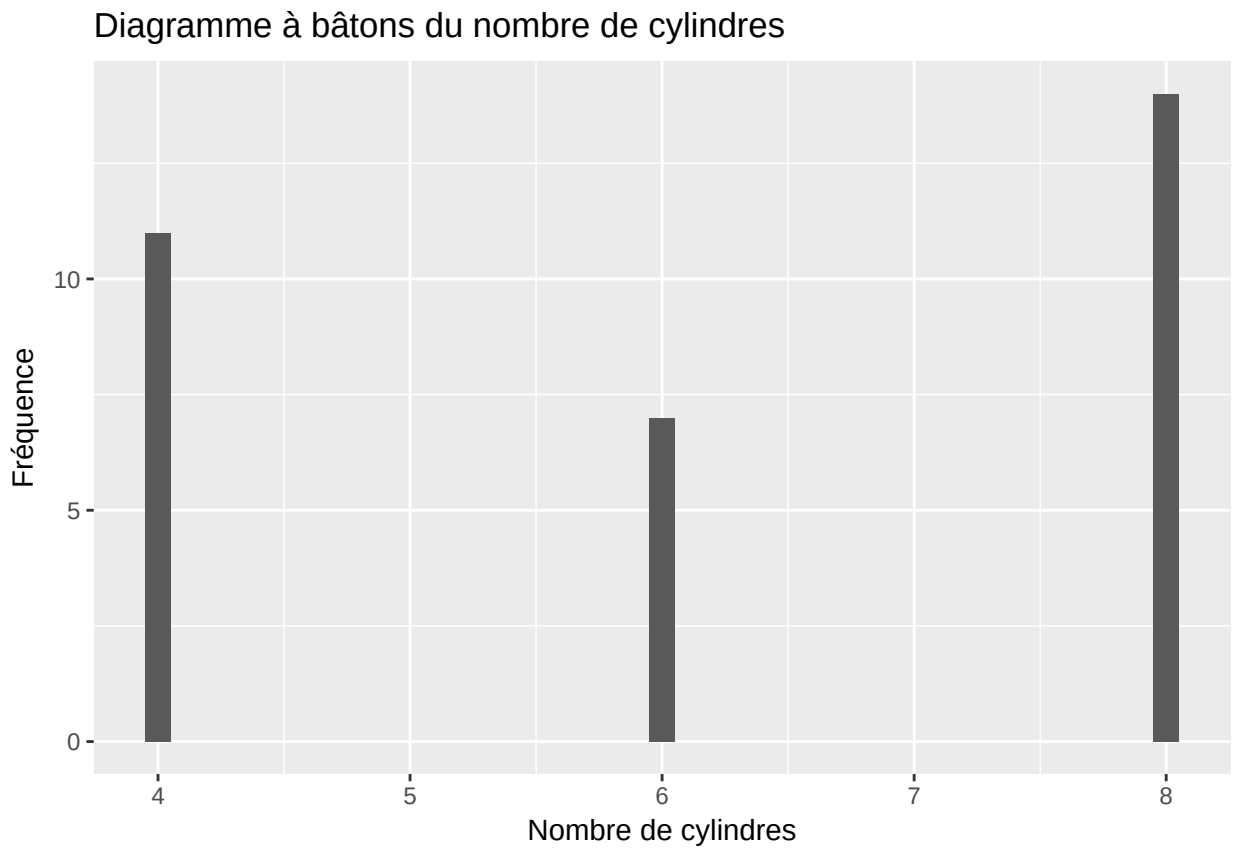
```
carat_class = cut(diamonds$carat,
                  breaks = seq(from = 0, to = 6, by = 1),
                  right = FALSE)
tabfreq(carat_class)
```

carat_class	Fréquence	Fréquence relative	Fréquence relative cumulée
[0,1)	34880	0.647	0.647
[1,2)	16906	0.313	0.960
[2,3)	2114	0.039	0.999
[3,4)	34	0.001	1.000
[4,5)	5	0.000	1.000
[5,6)	1	0.000	1.000
Total	53940	1.000	1.000

2.2.2 Diagramme à bâtons

Pour les variables quantitatives discrètes, le diagramme à bâtons est le graphique de choix.

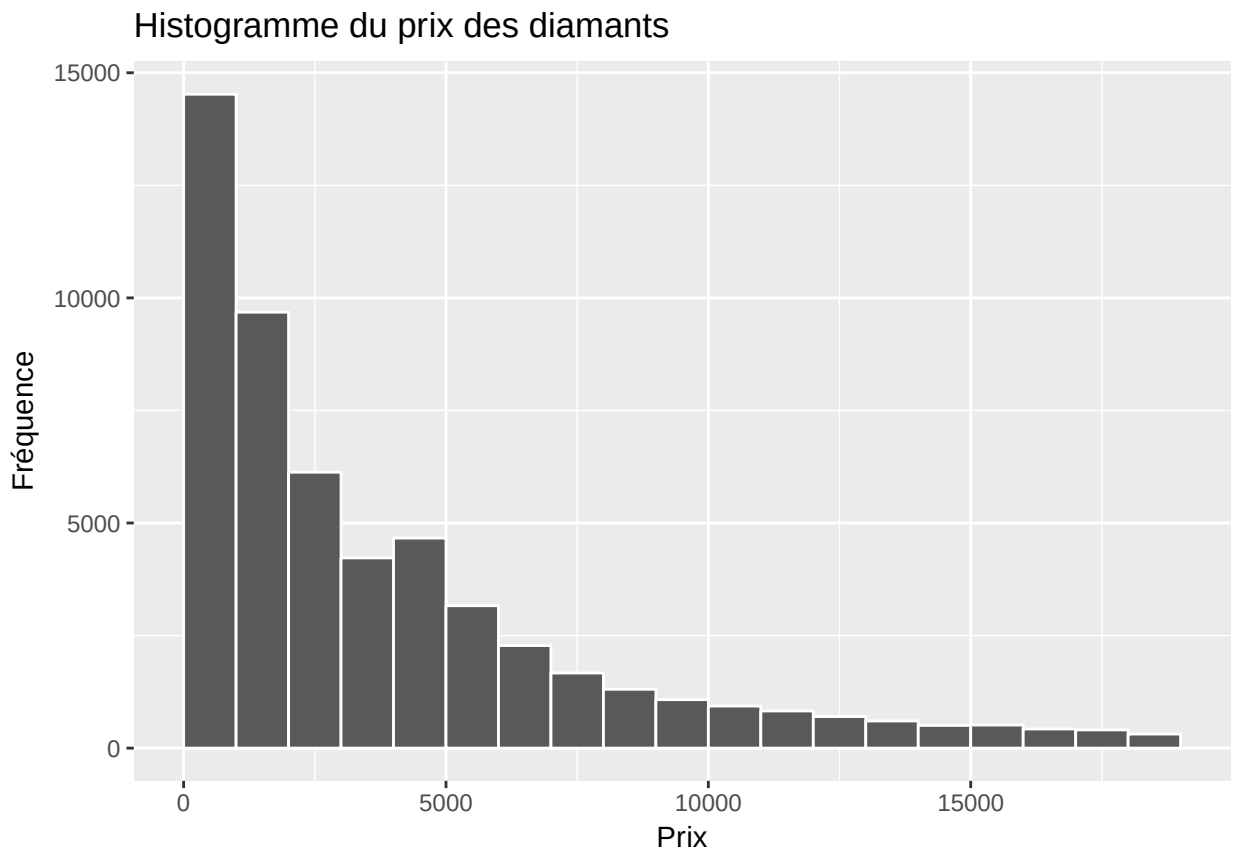
```
ggplot(mtcars, aes(cyl)) +
  geom_bar(width = 0.1) +
  labs(
    x = "Nombre de cylindres",
    y = "Fréquence",
    title = "Diagramme à bâtons du nombre de cylindres")
```



2.2.3 Histogramme

Pour les variables quantitatives discrètes, il est possible d'utiliser l'histogramme.

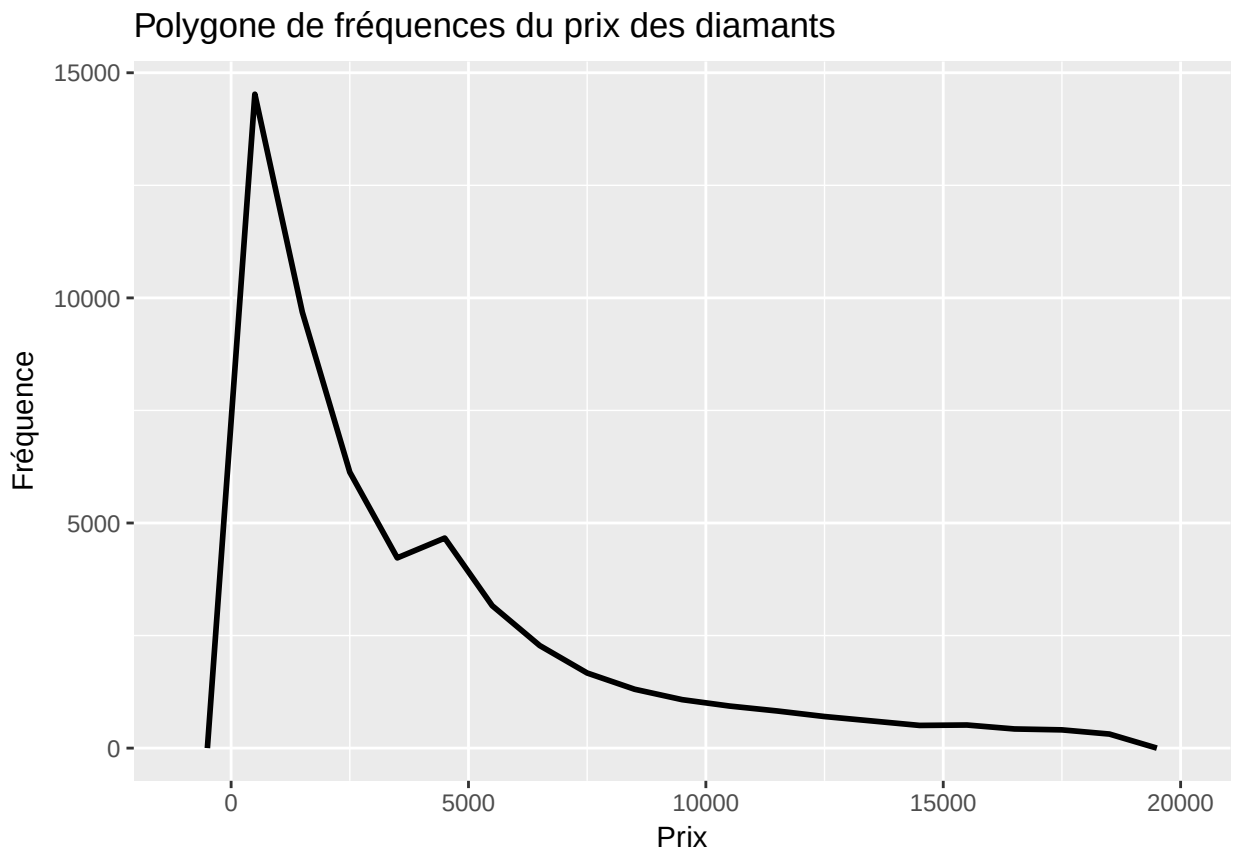
```
ggplot(diamonds, aes(price)) +
  geom_histogram(color = "white",binwidth = 1000, center = 500) +
  labs(
    x = "Prix",
    y = "Fréquence",
    title = "Histogramme du prix des diamants")
```



2.2.4 Polygone de fréquences

Pour les variables quantitatives discrètes, il est possible d'utiliser le polygone de fréquences.

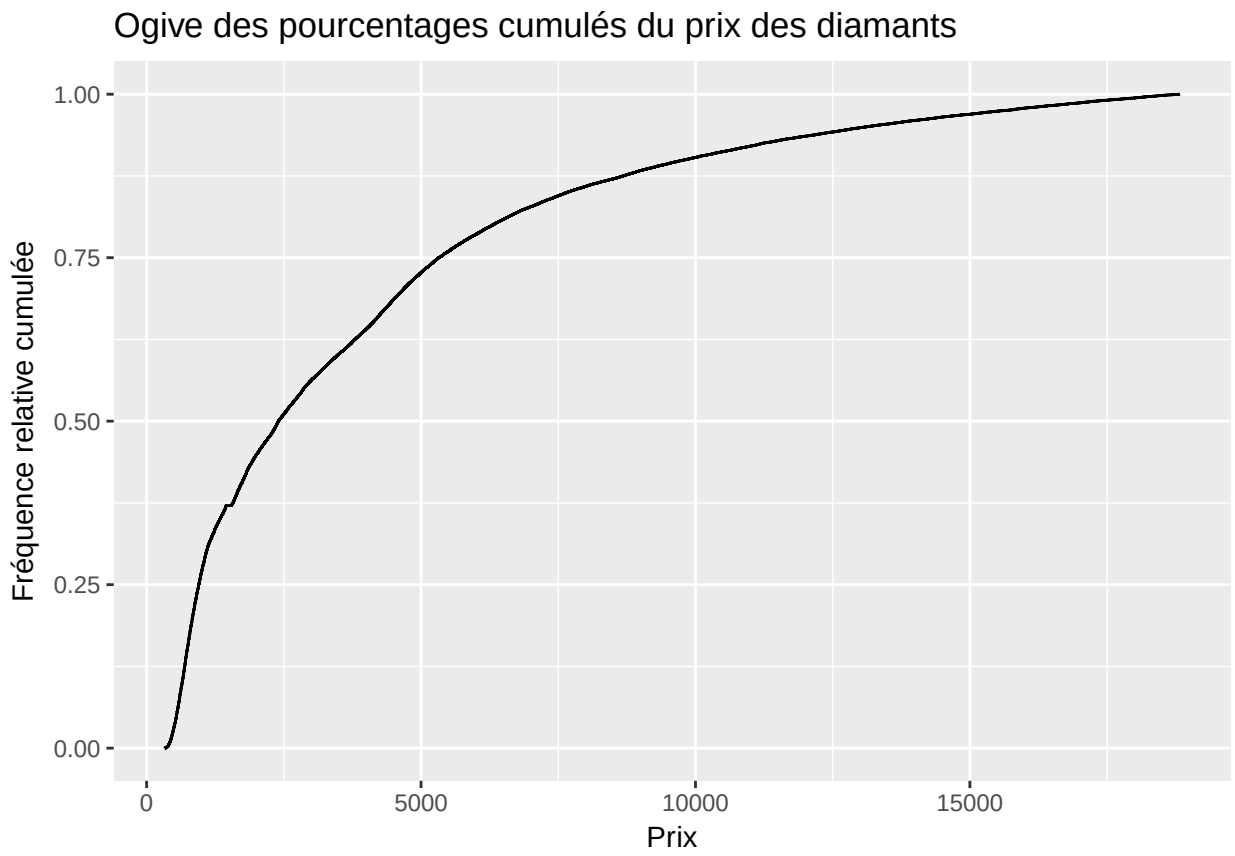
```
ggplot(diamonds, aes(price)) +
  geom_freqpoly(size = 1,binwidth = 1000, center = 500) +
  labs(
    x = "Prix",
    y = "Fréquence",
    title = "Polygone de fréquences du prix des diamants")
```



2.2.5 Ogive des pourcentages cumulés

Pour les variables quantitatives discrètes, il est possible d'utiliser l'ogive des pourcentages cumulés.

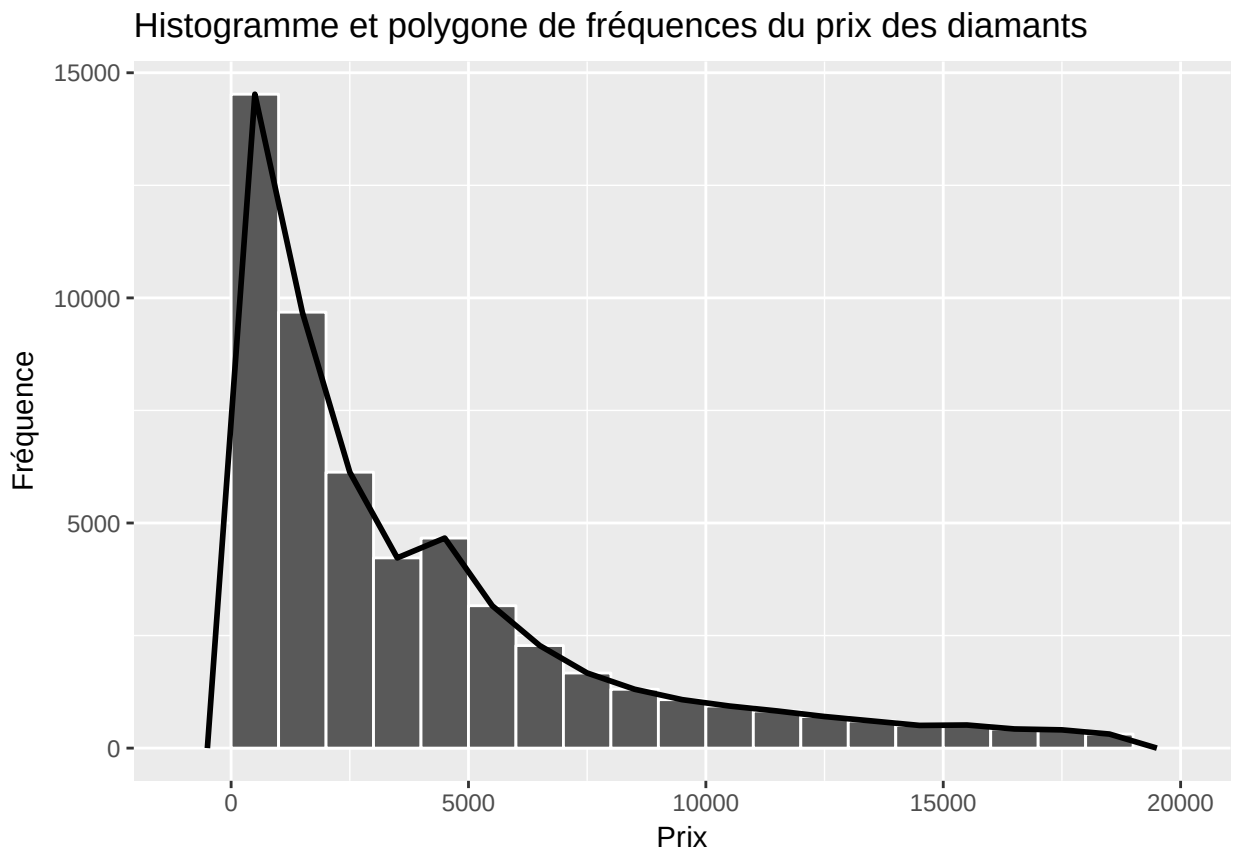
```
ggplot(diamonds, aes(price)) +
  stat_ecdf(pad = FALSE) +
  labs(
    x = "Prix",
    y = "Fréquence relative cumulée",
    title = "Ogive des pourcentages cumulés du prix des diamants")
```



2.2.6 Histogramme et polygone de fréquences

Pour les variables quantitatives discrètes, il est possible d'utiliser l'histogramme et le polygone de fréquences.

```
ggplot(diamonds, aes(price)) +
  geom_histogram(color = "white",,binwidth = 1000, center = 500) +
  geom_freqpoly(size = 1,,binwidth = 1000, center = 500) +
  labs(
    x = "Prix",
    y = "Fréquence",
    title = "Histogramme et polygone de fréquences du prix des diamants")
```


Histogramme et polygone de fréquences du prix des diamants
15000
10000
5000
0
Fréquence
0
5000
10000
15000
20000
Prix

Part II

Les mesures associées aux données

Chapter 3

Les différentes mesures

Dans ce chapitre, nous verrons comment utiliser `R` pour calculer les mesures importantes permettant de résumer des données.

Nous allons charger les paquetages que nous allons utiliser:

```
library(ggplot2)
library(nycflights13)
```

3.1 Les mesures de tendance centrale

Les mesures de tendance centrale permettent de déterminer où se situe le « centre » des données. Les trois mesures de tendance centrale sont le mode, la moyenne et la médiane.

3.1.1 Le mode

Le mode est la **modalité**, **valeur** ou **classe** possédant la plus grande fréquence. En d'autres mots, c'est la donnée la plus fréquente.

Puisque le mode se préoccupe seulement de la donnée la plus fréquente, il n'est pas influencé par les valeurs extrêmes.

Lorsque le mode est une classe, il est appelé **classe modale**.

Le mode est noté **Mo**.

Le langage `R` ne possède pas de fonction permettant de calculer le mode. La façon la plus simple de le calculer est d'utiliser la fonction `table` de `R`.

Par exemple, si nous voulons connaître le mode de la variable `cut` de la base de données `diamonds`:

```
table(diamonds$cut)
```

```
##
##      Fair      Good Very Good   Premium     Ideal
##      1610      4906     12082     13791     21551
```

Nous remarquons que le maximum est à la modalité *Ideal* avec une fréquence de 21551.

Si nous nous intéressons au mode d'une variable quantitative discrète comme `cyl` de la base de données `mtcars` nous obtenons:

```
table(mtcars$cyl)
```

```
##
##  4  6  8
## 11  7 14
```

Nous remarquons que le maximum est à la valeur *8* avec une fréquence de 14.

Dans le cas d'une variable quantitative continue, pour calculer le mode, il faut commencer par séparer les données en classes. Nous utiliserons les mêmes classes utilisées à la section 2.2.1

```
carat_class = cut(diamonds$carat,
                  breaks = seq(from = 0, to = 6, by = 1),
                  right = FALSE)
table(carat_class)
```

```
## carat_class
## [0,1) [1,2) [2,3) [3,4) [4,5) [5,6)
## 34880 16906  2114    34     5     1
```

La classe modale est donc la classe *[0,1)* avec une fréquence de 34880.

3.1.2 La médiane

La médiane, notée **Md**, est la valeur qui sépare une série de données classée en ordre croissant en deux parties égales.

La médiane étant la valeur du milieu, elle est la valeur où le pourcentage cumulé atteint 50%.

Puisque la médiane se préoccupe seulement de déterminer où se situe le centre des données, elle n'est pas influencée par les valeurs extrêmes. Elle est donc une mesure de tendance centrale plus fiable que la moyenne.

> Important : La médiane n'est définie que pour les variables quantitatives. En effet, si vous tentez d'utiliser la médiane pour des données autres que numériques, `R` vous donnera un message d'erreur.

La fonction `median` permet de calculer la médiane en langage `R`.

Par exemple, pour calculer la médiane de la variable `carat` de la base de données `diamonds`, nous avons:

```
median(diamonds$carat)
```

```
## [1] 0.7
```

Ceci signifie que 50% des diamants ont une valeur en carat inférieure ou égale à 0.7 et que 50% des diamants ont une valeur en carat supérieure ou égale à 0.7.

Nous pouvons aussi obtenir que la médiane de la variable `price` de la base de données `diamonds` est donnée par:

```
median(diamonds$price)
```

```
## [1] 2401
```

3.1.3 La moyenne

La moyenne est la valeur qui pourrait remplacer chacune des données d'une série pour que leur somme demeure identique. Intuitivement, elle représente le centre d'équilibre d'une série de données. La somme

des distances qui sépare les données plus petites que la moyenne devrait être la même que la somme des distances qui sépare les données plus grandes.

> Important : La moyenne n'est définie que pour les variables quantitatives. En effet, si vous tentez d'utiliser la moyenne pour des données autres que numériques, `R` vous donnera un message d'erreur.

La fonction `mean` permet de calculer la moyenne en langage `R`.

Par exemple, pour calculer la moyenne de la variable `carat` de la base de données `diamonds`, nous avons:

```
mean(diamonds$carat)
```

```
## [1] 0.7979397
```

Nous pouvons aussi obtenir que la moyenne de la variable `price` de la base de données `diamonds` est donnée par:

```
mean(diamonds$price)
```

```
## [1] 3932.8
```

3.2 Les mesures de dispersion

Les mesures de tendance centrale (mode, moyenne et médiane) ne permettent pas de déterminer si une série de données est principalement située autour de son centre, ou si au contraire elle est très dispersée.

Les mesures de dispersion, elles, permettent de déterminer si une série de données est centralisée autour de sa moyenne, ou si elle est au contraire très dispersée.

Les mesures de dispersion sont l'étendue, la variance, l'écart-type et le coefficient de variation.

3.2.1 L'étendue

La première mesure de dispersion, l'étendue, est la différence entre la valeur maximale et la valeur minimale.

L'étendue ne tenant compte que du maximum et du minimum, elle est grandement influencée par les valeurs extrêmes. Elle est donc une mesure de dispersion peu fiable.

La fonction `range` permet de calculer l'étendue d'une variable en langage `R`.

Par exemple, pour calculer l'étendue de la variable `carat` de la base de données `diamonds`, nous avons:

```
range(diamonds$carat)
```

```
## [1] 0.20 5.01
```

Nous pouvons donc calculer l'étendue de la variable `carat` en soustrayant les deux valeurs obtenues par la fonction `range`, c'est-à-dire que l'étendue est 5.01-0.2 = 4.81.

3.2.2 La variance

La variance sert principalement à calculer l'écart-type, la mesure de dispersion la plus connue.

> Attention : Les unités de la variance sont des unités^2.

La fonction `var` permet de calculer la variance d'une variable en langage `R`.

Par exemple, pour calculer la variance de la variable `carat` de la base de données `diamonds`, nous avons:

```
var(diamonds$carat)
```

```
## [1] 0.2246867
```

Ceci signifie que la variance de la variable `carat` est 0.2246867 carat^2.

3.2.3 L'écart-type

L'écart-type est la mesure de dispersion la plus couramment utilisée. Il peut être vu comme la « moyenne » des écarts entre les données et la moyenne.

Puisque l'écart-type tient compte de chacune des données, il est une mesure de dispersion beaucoup plus fiable que l'étendue.

Il est défini comme la racine carrée de la variance.

La fonction `sd` permet de calculer l"écart-type d'une variable en langage `R`.

Par exemple, pour calculer l'écart-type de la variable `carat` de la base de données `diamonds`, nous avons:

```
sd(diamonds$carat)
```

```
## [1] 0.4740112
```

Ceci signifie que l'écart-type de la variable `carat` est 0.4740112 carat.

3.2.4 Le coefficient de variation

Le coefficient de variation, noté C. V., est calculé comme suit :

$$C.V. = \frac{\text{ecart-type}}{\text{moyenne}} \times 100\% \tag{3.1}$$

Si le coefficient est inférieur à 15%, les données sont dites **homogènes**. Cela veut dire que les données sont situées près les unes des autres.

Dans le cas contraire, les données sont dites **hétérogènes**. Cela veut dire que les données sont très dispersées.

Important : Le coefficient de variation ne possède pas d'unité, outre le symbole de pourcentage.

Il n'existe pas de fonctions en `R` permettant de calculer directement le coefficient de variation. Par contre, nous pouvons utiliser en conjonction les fonctions `sd` et `mean` pour le calculer.

Par exemple, pour calculer le coefficient de variation de la variable `carat` de la base de données `diamonds`, nous avons:

```
sd(diamonds$carat)/mean(diamonds$carat)*100
```

```
## [1] 59.40439
```

Le C.V. de la variable `carat` est donc 59.4043906 %, ce qui signifie que les données sont hétérogènes, car le coefficient de variation est plus grand que 15%.

3.3 Les mesures de position

Les mesures de position permettent de situer une donnée par rapport aux autres. Les différentes mesures de position sont la cote Z, les quantiles et les rangs.

Tout comme les mesures de dispersion, celles-ci ne sont définies que pour une variable quantitative.

3.3.1 La cote z

Cette mesure de position se base sur la moyenne et l'écart-type.

La cote Z d'une donnée x est calculée comme suit :

$$Z = \frac{x - \text{moyenne}}{\text{ecart-type}} \tag{3.2}$$

Important : La cote z ne possède pas d'unités.

Une cote Z peut être positive, négative ou nulle.

Cote Z	Interprétation
Z>0	donnée supérieure à la moyenne
Z<0	donnée inférieure à la moyenne
Z=0	donnée égale à la moyenne

Il n'existe pas de fonctions en `R` permettant de calculer directement la cote Z. Par contre, nous pouvons utiliser en conjonction les fonctions `sd` et `mean` pour la calculer.

Par exemple, si nous voulons calculer la cote Z d'un diamant de 3 carats, nous avons:

```
(3-mean(diamonds$carat))/sd(diamonds$carat)
```

```
## [1] 4.645587
```

3.3.2 Les quantiles

Un quantile est une donnée qui correspond à un certain pourcentage cumulé.

Parmi les quantiles, on distingue les quartiles, les quintiles, les déciles et les centiles.

- Les quartiles Q_1, Q_2 et Q_3, séparent les données en quatre parties égales. Environ 25% des données sont inférieures ou égales à Q_1. Environ 50% des données sont inférieures ou égales à Q_2. Environ 75% des données sont inférieures ou égales à Q_3.
- Les quintiles V_1, V_2, V_3 et V_4, séparent les données en cinq parties égales. Environ 20% des données sont inférieures ou égales à V_1. Environ 40% des données sont inférieures ou égales à V_2. Etc.
- Les déciles D_1, D_2, ..., D_8 et D_9, séparent les données en dix parties égales. Environ 10% des données sont inférieures ou égales à D_1. Environ 20% des données sont inférieures ou égales à D_2. Etc.
- Les centiles C_1, C_2, ..., C_{98} et C_{99}, séparent les données en cent parties égales. Environ 1% des données sont inférieures ou égales à C_1. Environ 2% des données sont inférieures ou égales à C_2. Etc.

  Il est utile de noter que certains quantiles se recoupent.

La fonction `quantile` permet de calculer n'importe quel quantile d'une variable en langage `R`. Il suffit d'indiquer la variable étudiée ainsi que le pourcentage du quantile voulu.

Par exemple, si nous voulons calculer D_1 pour la variable `carat`, nous allons utiliser la fonction `quantile` avec une probabilité de 0,1.

```
quantile(diamonds$carat, 0.1)
```

```
##  10%
## 0.31
```

Ceci implique que 10% des diamants ont une valeur en carat inférieure ou égale à 0.31 carat.

Nous pouvons calculer le troisième quartile Q_3 de la variable `price` en utilisant la fonction `quantile` avec une probabilité de 0,75.

```
quantile(diamonds$price, 0.75)
```

```
##     75%
## 5324.25
```

Ceci implique que 75% des diamants ont un prix en dollars inférieur ou égal à 5324.25 $.

3.3.3 La commande `summary`

La commande `summary` produit un sommaire contenant six mesures importantes:

1. `Min` : le minimum de la variable
2. `1st Qu.`: Le premier quartile, Q_1, de la variable
3. `Median` : La médiane de la variable
4. `Mean` : La moyenne de la variable
5. `3rd Qu.` : Le troisième quartile, Q_3, de la variable
6. `Max` : Le maximum de la variable

Nous pouvons donc produire le sommaire de la variable `price` de la base de données `diamonds` de la façon suivante:

```
summary(diamonds$price)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     326     950    2401    3933    5324   18823
```

3.3.4 Le rang centile

Un rang centile représente le pourcentage cumulé, *exprimé en nombre entier*, qui correspond à une certaine donnée. Nous déterminerons les rangs centiles pour les variables continues seulement.

Les rangs centiles sont donc exactement l'inverse des centiles.

Il n'existe pas de fonctions dans `R` permettant de trouver directement le rang centile, mais il est facile d'utiliser la fonction `mean` pour le trouver.

Par exemple, si nous voulons trouver le rang centile d'un diamant qui coûte 500$, il suffit d'utiliser la commande suivante. La commande calcule la moyenne de toutes les valeurs en dollars des diamants coûtant 500$ ou moins.

```
mean(diamonds$price<=500)
```

```
## [1] 0.03242492
```

Ceci signifie que pour un diamant de 500$, il y a 3.2424917 % des diamants qui ont une valeur égale ou inférieure.

Chapter 4

Les séries chronologiques

Débutons par charger les paquetages qui nous seront utiles.

```
library(gapminder)
library(nycflights13)
library(ggplot2)
library(dplyr)
```

```
##
## Attachement du package : 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Une série chronologique est un ensemble de valeurs observées d'une variable quantitative. Elle permet d'analyser l'évolution de cette variable dans le temps dans le but éventuel de faire des prévisions.

4.1 Les graphiques

Nous allons débuter par utiliser la base de données `nycflights13`. Nous allons étudier la température au mois de janvier 2013 à l'aéroport Newark (code "EWR" dans la variable `origin`). La variable `weather` de la base de données contient ces informations mais nous devons tout d'abord filtrer les données pour ne conserver que celles qui correspondent à Newark et au mois de janvier.

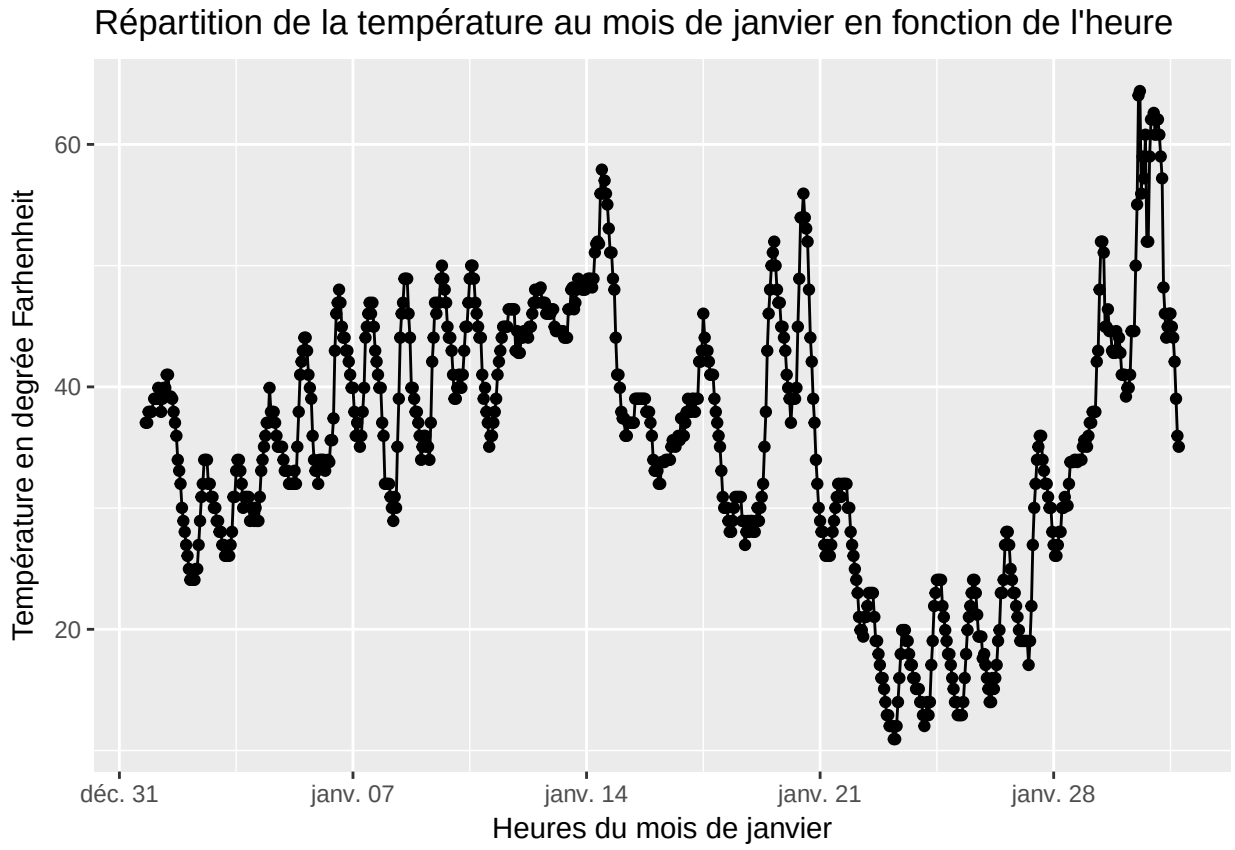
La commande suivante permet de faire ce filtrage. Vous n'avez pas besoin de comprendre la syntaxe.

```
meteo_janvier_ewr <- weather %>%
  filter(origin == "EWR" & month == 1 )
```

Nous pouvons maintenant tracer les données obtenues:

```
ggplot(meteo_janvier_ewr, aes(x = time_hour, y = temp)) +
  geom_line() +
  geom_point() +
  labs(
    x = "Heures du mois de janvier",
```

```
    y = "Température en degrée Farhenheit",
    title = "Répartition de la température au mois de janvier en fonction de l'heure"
  )
```

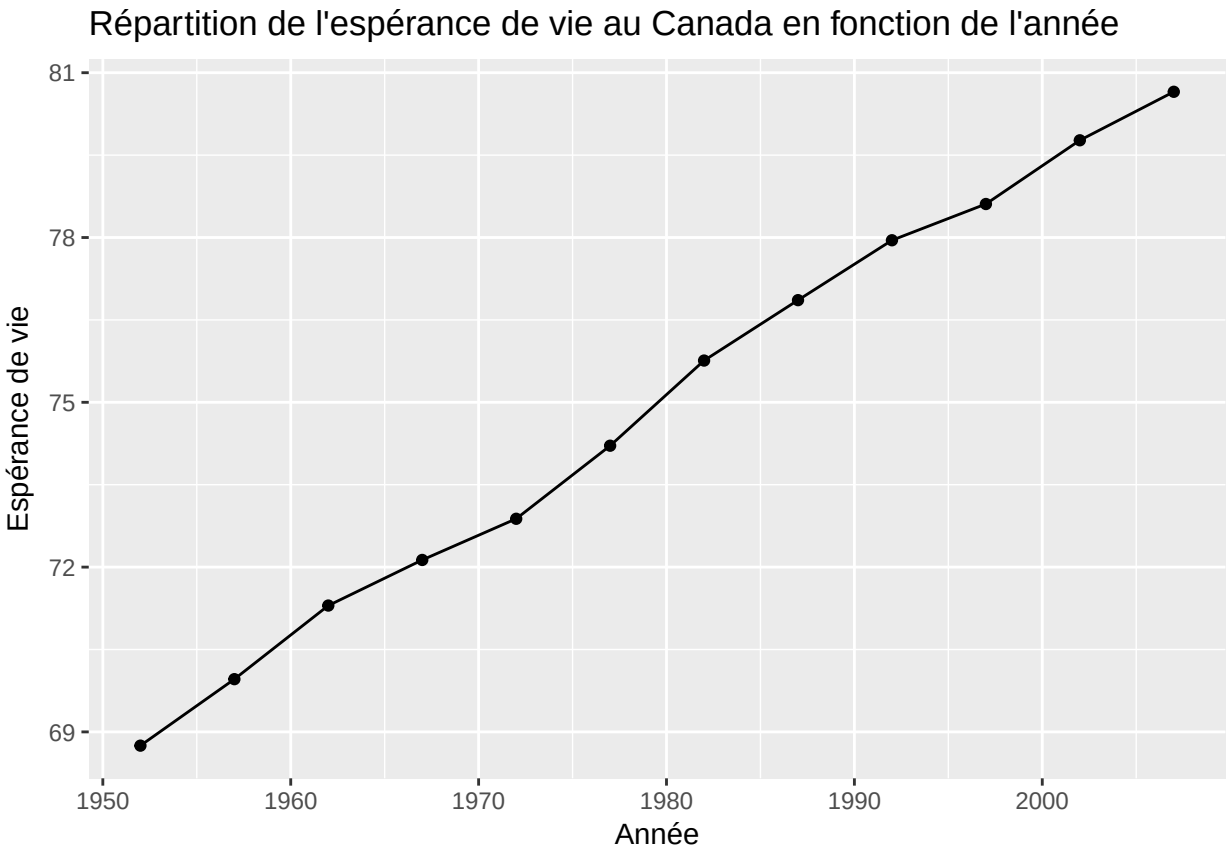


Nous pouvons aussi utiliser la paquetage `gapminder` qui contient des données sur l'espérance de vie. Comme précédemment, nous allons débuter par filtrer les données provenant uniquement du Canada.

```
gap_canada <- gapminder %>%
  filter(country == "Canada")
```

Nous pouvons maintenant tracer les données obtenues:

```
ggplot(gap_canada, aes(x = year, y = lifeExp)) +
  geom_line() + geom_point() +
  labs(
    x = "Année",
    y = "Espérance de vie",
    title = "Répartition de l'espérance de vie au Canada en fonction de l'année")
```



4.2 Les mesures

4.2.1 La variation absolue

La variation absolue mesure l'augmentation (ou la diminution) subie par une variable dans le temps. Pour calculer la variation absolue entre un moment A antérieur à un moment B, on utilise la formule ci-dessous :

$$\Delta V = V_B - V_A \tag{4.1}$$

où V_B est la valeur de la variable au temps B et V_A est la valeur de la variable au temps A.

Remarque : Les unités de la variation absolue sont les mêmes que celles de la variable étudiée.

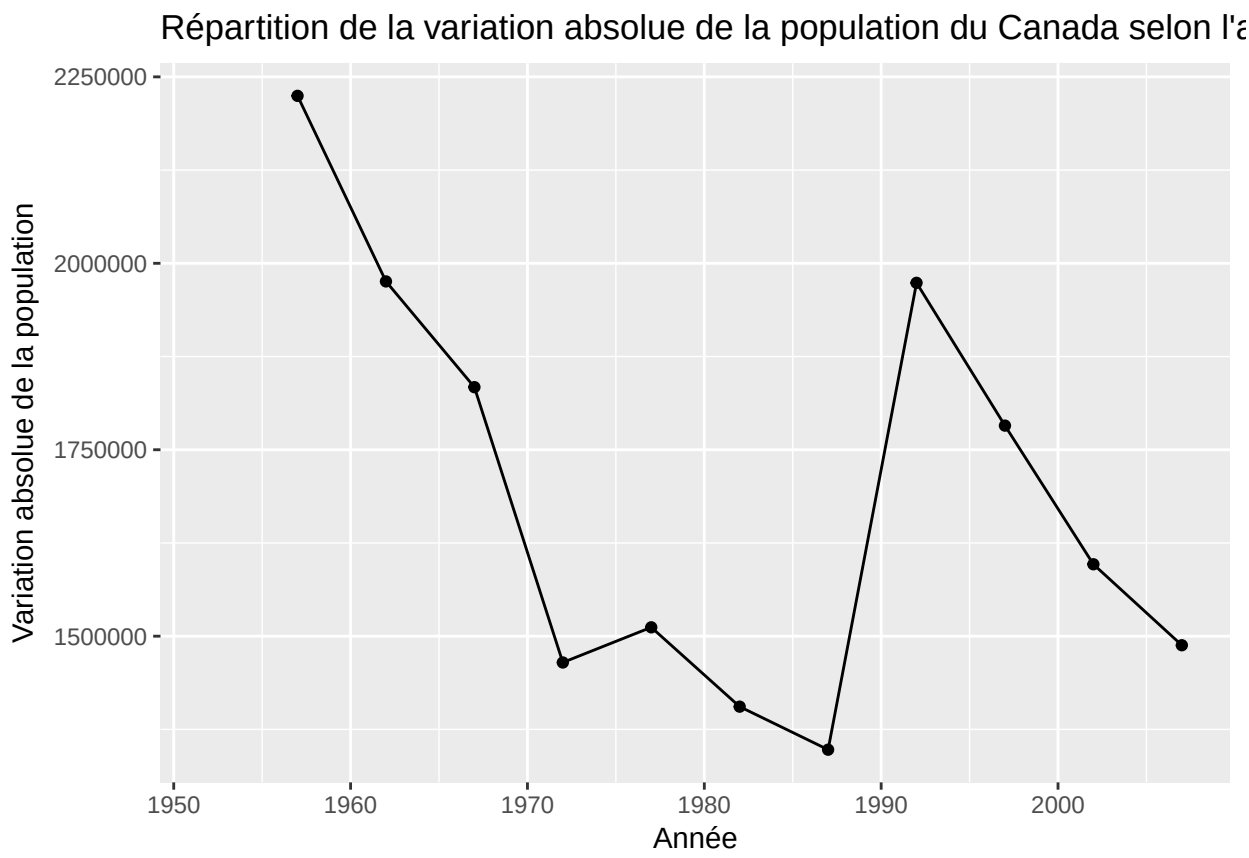
Si nous voulons connaître la variation absolue de la population du Canada, nous allons devoir ajouter une colonne à notre base de données `gap_canada`. Encore une fois, il n'est pas nécessaire de comprendre la syntaxe. Nous ajoutons une colonne variation absolue, notée `var_abs`, à notre base de données `gap_canada`.

```
gap_canada <- gap_canada %>%
  mutate(var_abs = pop - lag(pop))
```

Nous pouvons maintenant représenter la variable à l'aide d'un graphique.

```
ggplot(gap_canada, aes(x = year, y = var_abs)) +
  geom_line() +
  geom_point() +
  labs(
```

```
    x = "Année",
    y = "Variation absolue de la population",
    title = "Répartition de la variation absolue de la population du Canada selon l'année"
  )
```



4.2.2 La variation moyenne

La variation moyenne mesure l'augmentation (ou la diminution) moyenne subie par une variable par unité de temps. La variation moyenne entre les moments et est donnée par :

$$\Delta V_{moy} = \frac{V_B - V_A}{B - A} \tag{4.2}$$

Remarque : Les unités de la variation moyenne sont les unités de la variable étudiée par unité de temps.

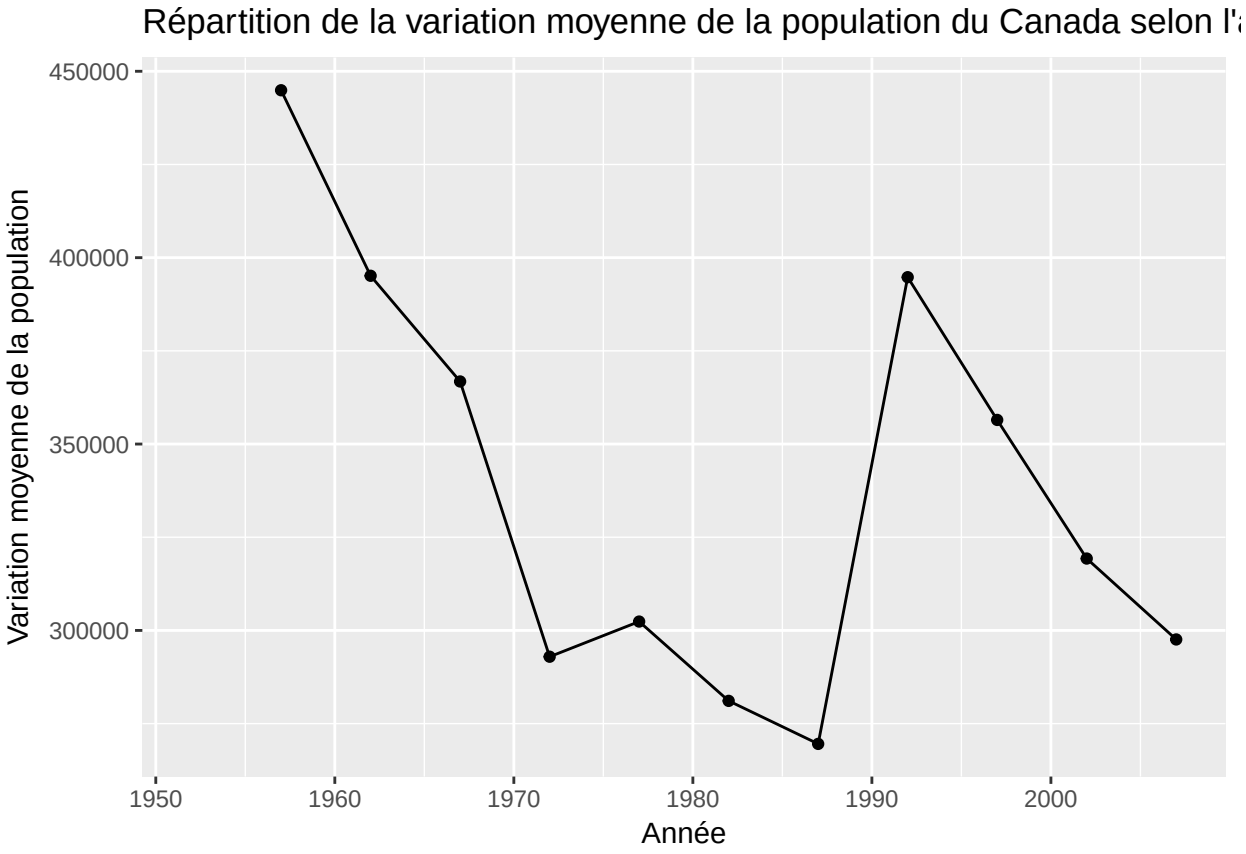
Si nous voulons connaître la variation moyenne de la population du Canada, nous allons devoir ajouter une colonne à notre base de données `gap_canada`. Encore une fois, il n'est pas nécessaire de comprendre la syntaxe. Nous ajoutons une colonne variation moyenne, notée `var_moy`, à notre base de données `gap_canada`.

```
gap_canada <- gap_canada %>%
  mutate(var_moy = (pop - lag(pop))/(year-lag(year)))
```

Nous pouvons maintenant représenter la variable à l'aide d'un graphique.

```
ggplot(gap_canada, aes(x = year, y = var_moy)) +
  geom_line() +
```

```
  geom_point() +
  labs(
    x = "Année",
    y = "Variation moyenne de la population",
    title = "Répartition de la variation moyenne de la population du Canada selon l'année"
  )
```



4.2.3 La variation relative (pourcentage de variation)

La variation relative exprime *en pourcentage* la variation subie par une variable entre les moments et . Le pourcentageest donné par:

$$\Delta V_{\%} = \frac{V_B - V_A}{V_A} \times 100 \tag{4.3}$$

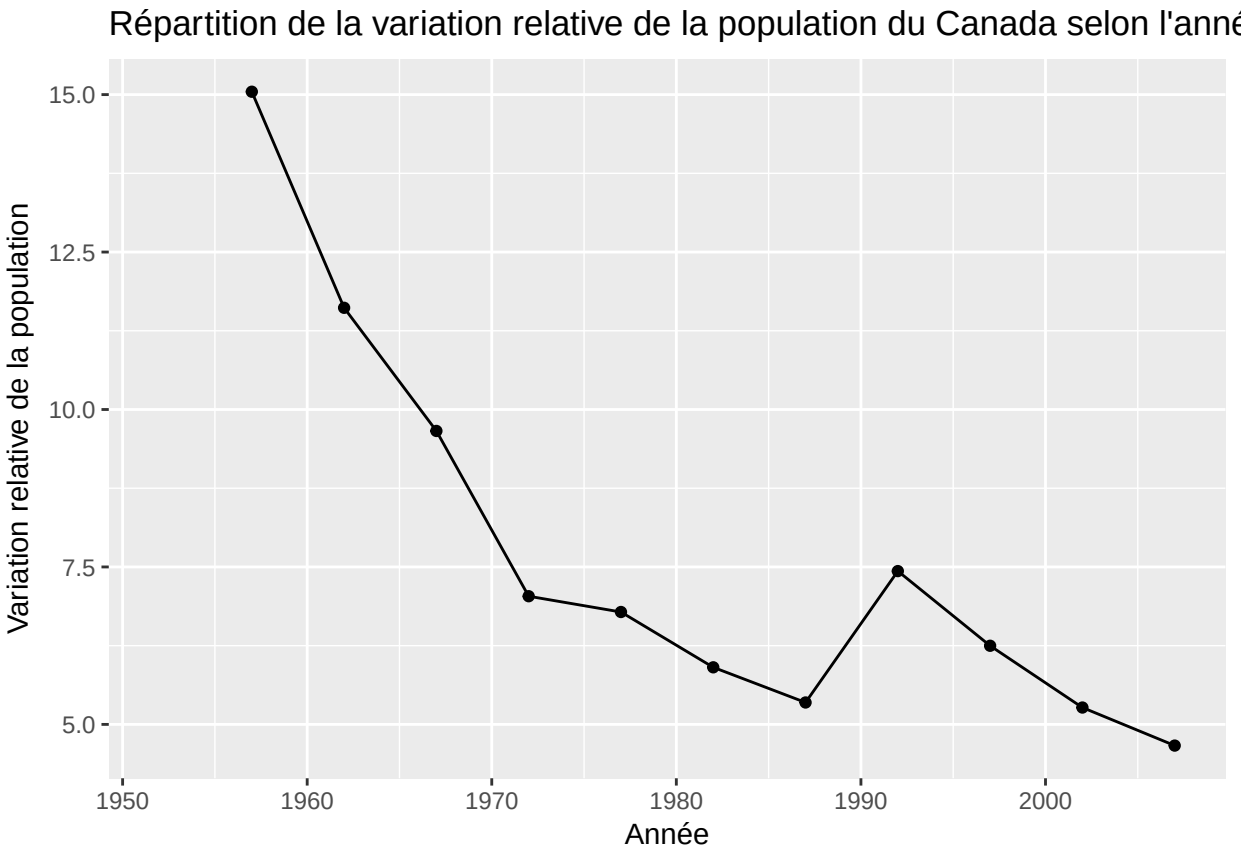
Remarque : Il n'y a pas d'unité autre que le symbole de pourcentage.

Si nous voulons connaître la variation relative de la population du Canada, nous allons devoir ajouter une colonne à notre base de données `gap_canada`. Encore une fois, il n'est pas nécessaire de comprendre la syntaxe. Nous ajoutons une colonne variation relative, notée `var_rel`, à notre base de données `gap_canada`.

```
gap_canada <- gap_canada %>%
  mutate(var_rel = (pop - lag(pop))/lag(pop) * 100)
```

Nous pouvons maintenant représenter la variable à l'aide d'un graphique.

```
ggplot(gap_canada, aes(x = year, y = var_rel)) +
  geom_line() +
  geom_point() +
  labs(
    x = "Année",
    y = "Variation relative de la population",
    title = "Répartition de la variation relative de la population du Canada selon l'année"
  )
```



4.3 Les données construites

Part III

La combinatoire et les probabilités

Chapter 5

La combinatoire

Chapter 6

Les lois de probabilités

Pour être en mesure d'utiliser les lois de probabilités en langage `R`, il faut charger le paquetage `stats`.

```
library(stats)
library(ggplot2)
```

Chaque distribution en `R` possède quatre fonctions qui lui sont associées. Premièrement, la fonction possède un *nom racine*, par exemple le *nom racine* pour la distribution *binomiale* est `binom`. Cette racine est précédée par une de ces quatre lettre:

- `p` pour *probabilité*, qui représente la fonction de répartition
- `q` pour *quantile*, l'inverse de la fonction de répartition
- `d` pour *densité*, la fonction de densité de la distribution
- `r` pour *random*, une variable aléatoire suivant la distribution spécifiée.

Pour la loi binomiale par exemple, ces fonctions sont `pbinom`, `qbinom`, `dbinom` et `rbinom`.

6.1 Les lois de probabilités discrètes

6.1.1 La loi binomiale

Le *nom racine* pour la loi binomiale est `binom`.

Soit X: le nombre de succès en n essais et $X \sim B(n,p)$. Voici la façon de calculer des probabilités pour la loi binomiale à l'aide de `R`:

Probabilités	Commande `R`
$P(X = k)$	`dbinom(k, n, p)`
$P(i \leq X \leq j)$	`sum(dbinom(i:j, n, p))`
$P(X \leq k)$	`pbinom(k, n, p)`
$P(X > k)$	`1-pbinom(k, n, p)`

Soit X la variable aléatoire comptant le nombre de face 2 que nous obtenons en lançant un dé à quatre reprises. Nous avons que $X \sim B(4, \frac{1}{6})$. Si nous voulons calculer $P(X = 3)$, nous aurons:

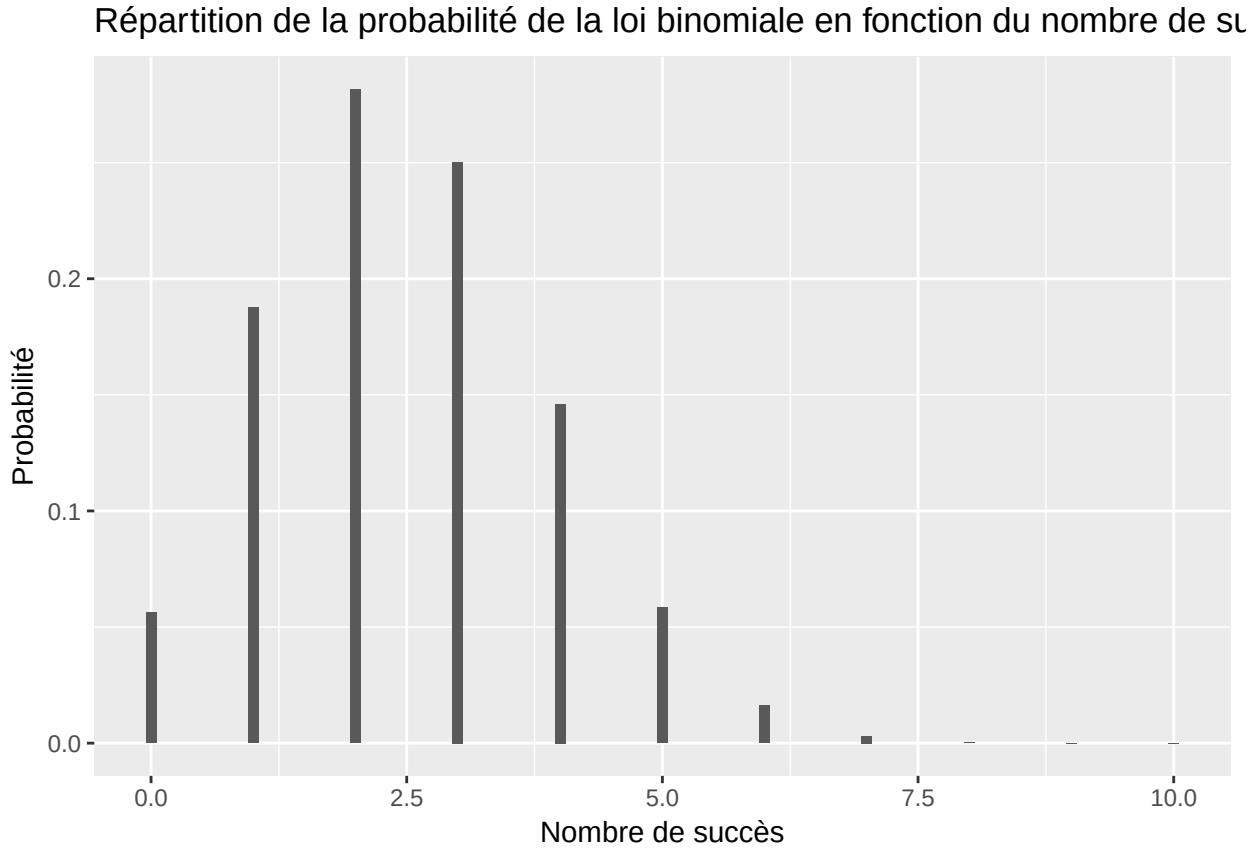
```
dbinom(3,4,1/6)
```

```
## [1] 0.0154321
```

Nous avons donc une probabilité de 1.5432099% d'obtenir 3 fois la face deux en lançant un dé à quatres reprises.

Nous pouvons représenter graphiquement la loi binomiale. Soit X $B(10, 1/4)$. Nous aurons:

```
fbinom <- data.frame(x = 0:10, y = dbinom(0:10, 10, 1/4))
ggplot(fbinom, aes(x = x, y = y)) +
  geom_bar(width = 0.1, stat = "identity") +
  labs(
    x = "Nombre de succès",
    y = "Probabilité",
    title = "Répartition de la probabilité de la loi binomiale en fonction du nombre de succès"
  )
```



6.1.2 La loi de Poisson

Le *nom racine* pour la loi de Poisson est `pois`.

Soit X: le nombre d'événements dans un intervalle fixé et $X \sim Po(\lambda)$. Voici la façon de calculer des probabilités pour la loi de Poisson à l'aide de `R`:

Probabilités	Commande `R`
$P(X = k)$	`dpois(k, lambda)`
$P(i \leq X \leq j)$	`sum(dpois(i:j, lambda))`
$P(X \leq k)$	`ppois(k, lambda)`
$P(X > k)$	`1-ppois(k, lambda)`

Soit X le nombre d'erreurs dans une page. Si une page contient en moyenne une demie erreur alors $X \sim Po(1/2)$. Si nous voulons calculer $P(X = 2)$, nous aurons:

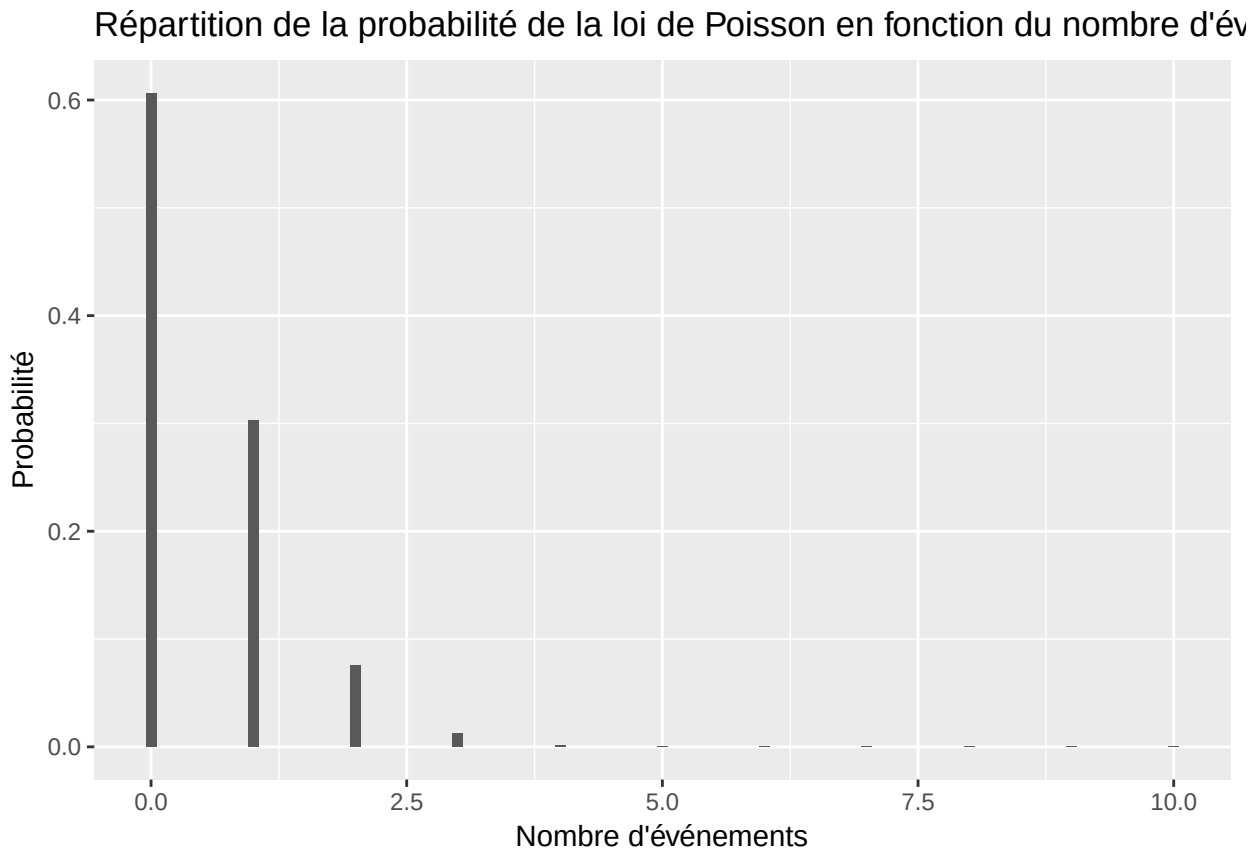
```
dpois(2, 1/2)
```

```
## [1] 0.07581633
```

Nous avons donc une probabilité de 7.5816332% d'obtenir deux erreurs sur une page.

Nous pouvons représenter graphiquement la loi de Poisson. Soit $X \sim Po(1/2)$. Nous aurons:

```
fpois <- data.frame(x = 0:10, y = dpois(0:10, 1/2))
ggplot(fpois, aes(x = x, y = y)) +
  geom_bar(width = 0.1, stat = "identity") +
  labs(
    x = "Nombre d'événements",
    y = "Probabilité",
    title = "Répartition de la probabilité de la loi de Poisson en fonction du nombre d'événements"
  )
```



6.1.3 La loi géométrique

Le *nom racine* pour la loi géométrique est `geom`.

Soit X: le nombre d'échecs avant d'obtenir un succès et $X \sim G(p)$. Voici la façon de calculer des probabilités pour la loi géométrique à l'aide de `R`:

Probabilités	Commande R
$P(X = k)$	`dgeom(k, p)`
$P(i \leq X \leq j)$	`sum(dgeom(i:j, p))`
$P(X \leq k)$	`pgeom(k, p)`
$P(X > k)$	`1-pgeom(k, p)`

Soit X le nombre d'échecs avant d'avoir un premier succès. Si la probabilité de succès est $\frac{1}{5}$ alors $X \sim G(1/5)$. Si nous voulons calculer $P(X = 6)$, nous aurons:

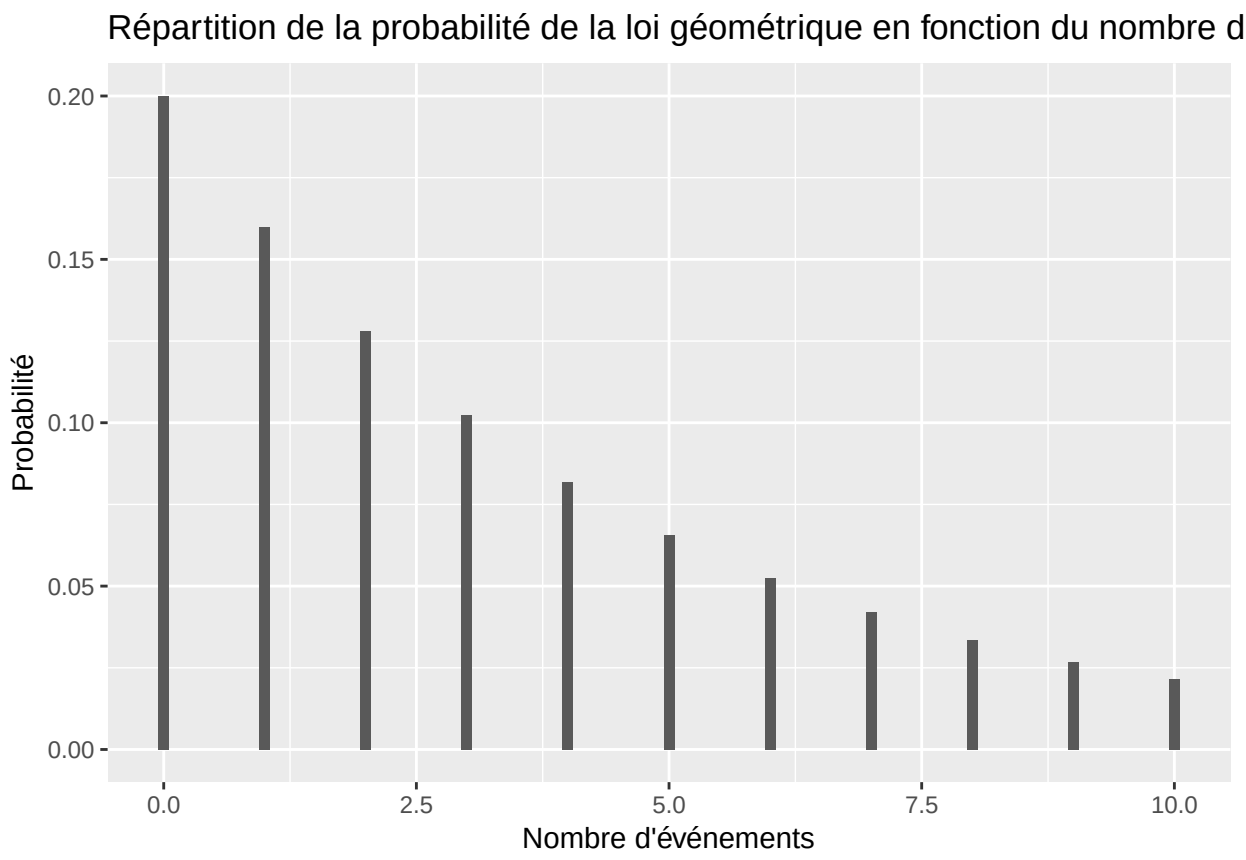
```
dgeom(6, 1/5)
```

```
## [1] 0.0524288
```

Nous avons donc une probabilité de 5.24288% d'obtenir 6 échecs avant un premier succès.

Nous pouvons représenter graphiquement la loi géométrique. Soit $X \sim G(1/5)$. Nous aurons:

```
fgeom <- data.frame(x = 0:10, y = dgeom(0:10, 1/5))
ggplot(fgeom, aes(x = x, y = y)) +
  geom_bar(width = 0.1, stat = "identity") +
  labs(
    x = "Nombre d'événements",
    y = "Probabilité",
    title = "Répartition de la probabilité de la loi géométrique en fonction du nombre d'échecs avant l
  )
```



Remarque : Pour la loi géométrique, on rencontre parfois cette définition : la probabilité p'(k) est la probabilité, lors d'une succession d'épreuves de Bernoulli indépendantes, d'obtenir k échecs

avant un succès. On remarque qu'il ne s'agit que d'un décalage de la précédente loi géométrique. Si X suit la loi p, alors $X + 1$ suit la loi p'.

6.1.4 La loi hypergéométrique

Le *nom racine* pour la loi hypergéométrique est `hyper`.

On tire sans remise n objets d'un ensemble de N objets dont A possèdent une caractéristique particulière (et les autres $B = N - A$ ne la possèdent pas). Soit X le nombre d'objets de l'échantillon qui possèdent la caractéristique. Nous avons que $X \sim H(N, A, n)$.

Voici la façon de calculer des probabilités pour la loi hypergéométrique à l'aide de `R`:

Probabilités	Commande `R`
$P(X = k)$	`dhyper(k, A, B, n)`
$P(i \leq X \leq j)$	`sum(dhyper(i:j, A, B, n))`
$P(X \leq k)$	`phyper(k, A, B, n)`
$P(X > k)$	`1-phyper(k, A, B, n)`

Soit X le nombre de boules blanches de l'échantillon de taille 4. Si l'urne contient 5 boules blanches et 8 boules noires, nous avons $X \sim H(13, 5, 4)$. Si nous voulons calculer $P(X = 2)$, nous aurons:

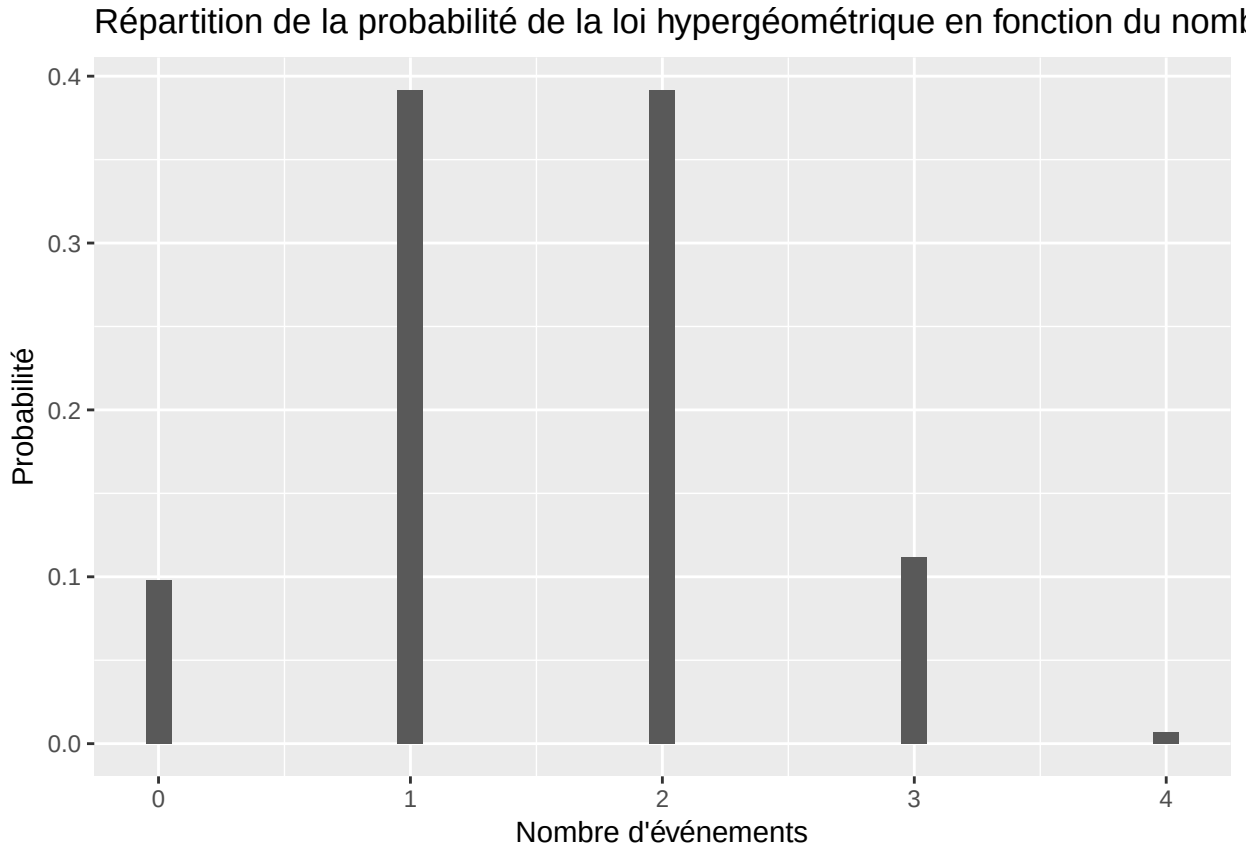
```
dhyper(2, 5, 8, 4)
```

```
## [1] 0.3916084
```

Nous avons donc une probabilité de 39.1608392% de piger 2 boules blanches dans un échantillon de taille 4.

Nous pouvons représenter graphiquement la loi hypergéométrique. Soit $X \sim H(13, 5, 4)$. Nous aurons:

```
fhyper <- data.frame(x = 0:4, y = dhyper(0:4, 5, 8, 4))
ggplot(fhyper, aes(x = x, y = y)) +
  geom_bar(width = 0.1, stat = "identity") +
  labs(
    x = "Nombre d'événements",
    y = "Probabilité",
    title = "Répartition de la probabilité de la loi hypergéométrique en fonction du nombre de boules b
  )
```



6.2 Les lois de probabilités continues

6.2.1 La loi normale

Le *nom racine* pour la loi normale est `norm`.

Si X suit une loi normale de moyenne μ et de variance σ^2, nous avons $X \sim N(\mu, \sigma^2)$.

Voici la façon de calculer des probabilités pour la loi normale à l'aide de `R`:

Probabilités	Commande `R`
$P(i \leq X \leq j)$	`pnorm(j, mu, sigma)-pnorm(i, mu, sigma)`
$P(X \leq k)$	`pnorm(k, mu, sigma)`
$P(X > k)$	`1-pnorm(k, mu, sigma)`

Soit $X \sim N(3, 25)$ une variable aléatoire suivant une loi normale de moyenne 3 et de variance 25. Si nous voulons calculer la probabilité $P(1.25 < X < 3.6)$ en `R`, nous pouvons utiliser la commande suivante:

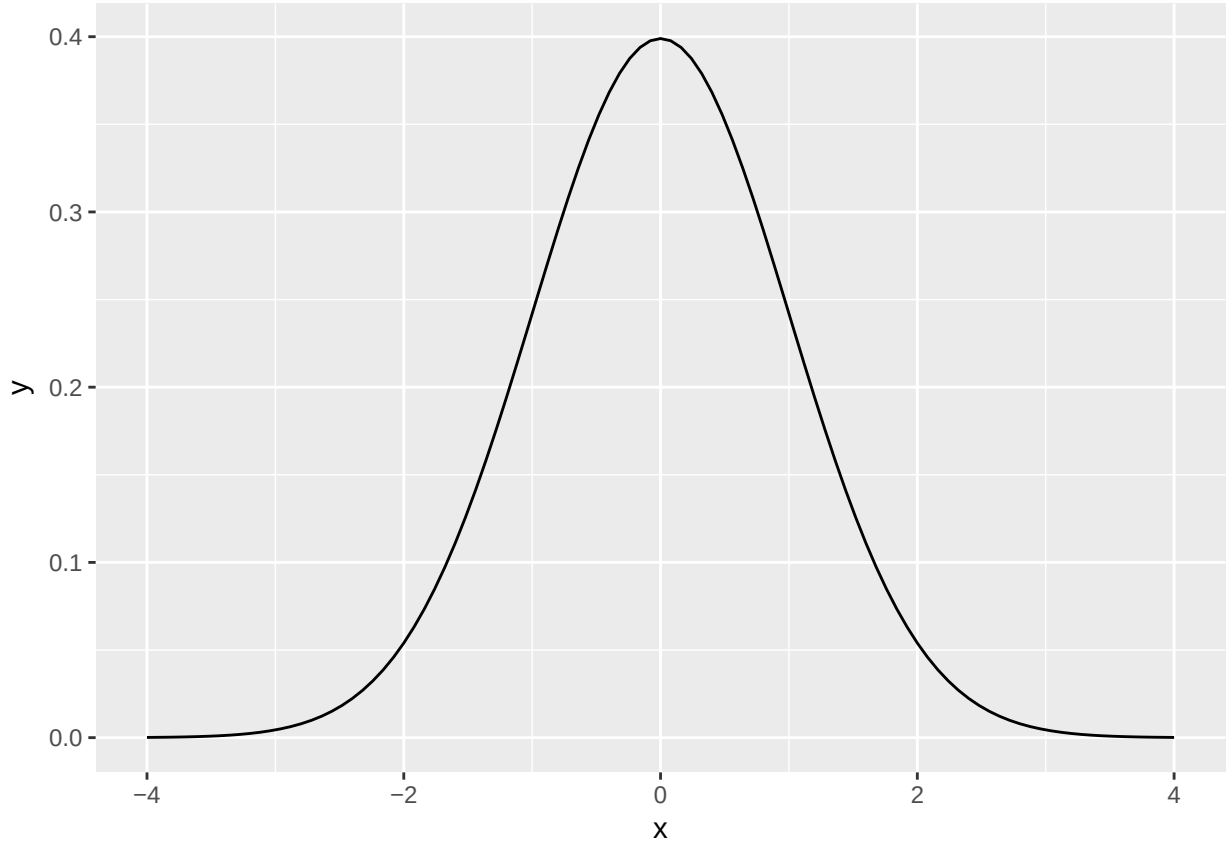
```
pnorm(3.6, 3, 5) - pnorm(1.25, 3, 5)
```

```
## [1] 0.1845891
```

La probabilité que notre variable aléatoire se trouve entre 1.25 et 3.6 est donc 18.4589077 %.

Nous pouvons représenter graphiquement la loi normale. Soit $X \sim N(0, 1)$. Nous aurons:

```
ggplot(data = data.frame(x = c(-4, 4)), aes(x)) +
  stat_function(fun = dnorm, args = list(mean = 0, sd = 1))
```



6.2.2 La loi de Student

Le *nom racine* pour la loi de Student est `t`.

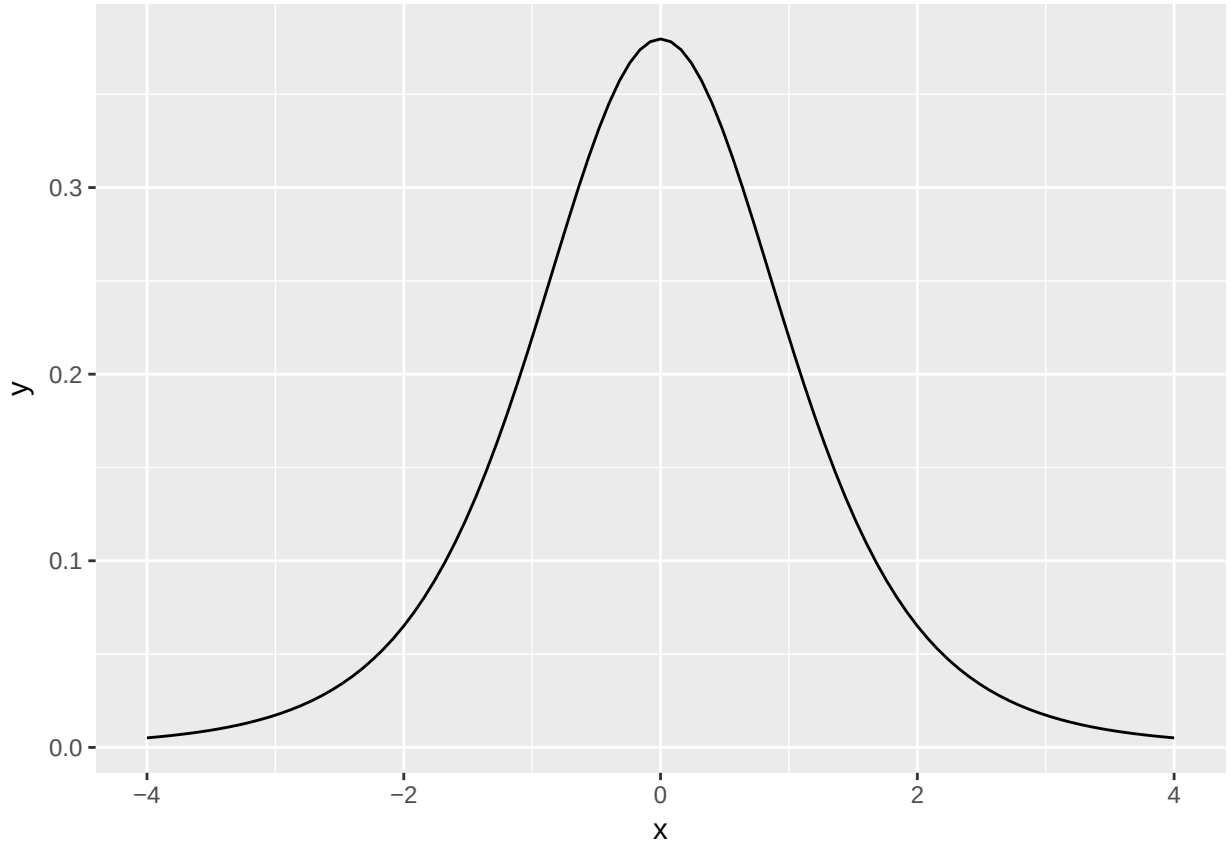
Si X suit une loi de Student à ν degrés de liberté, nous avons $X \sim T_\nu$.

Voici la façon de calculer des probabilités pour la loi de Student à l'aide de `R`:

Probabilités	Commande `R`
$P(i \leq X \leq j)$	`pt(j, nu)-pt(i, nu)`
$P(X \leq k)$	`pt(k, nu)`
$P(X > k)$	`1-pt(k, nu)`

Nous pouvons représenter graphiquement la loi de Student. Soit $X \sim T_5$. Nous aurons:

```
ggplot(data = data.frame(x = c(-4, 4)), aes(x)) +
  stat_function(fun = dt, args = list(df = 5))
```


0.3
0.2
0.1
0.0
y
−4
−2
0
2
4
x

Bibliography