1_1_cours

October 15, 2018

Table of Contents

Dans ce cours, nous allons utiliser la console RStudio que nous avons installé. Nous allons présenter la console, et découvrir les principales sections que nous utiliserons tout au long des prochains cours sur la programmation en R.

1 Ouvrir Rstudio

1.1 Sous Linux

vous ouvrez le terminal en appuyant sur les touches `Ctrl+Alt+T` et vous tapez `RStudio`

1.2 Sous macOS

Vous appuyez sur la touche `cmd+space` vous tapez ensuite sur `RSdtudio` en `Enter` ensuite

1.3 Sous windows,

Cherchez l'application RStudio et vous cliquez là-dessus pour la lancer

2 Présentation de RStudio

Lorsque nous ouvrons RStudion, nous avons alors une fenêtre avec trois sections dans lesquels nous allons travailler.

2.1 Les sous-section

1. La console est là où le code R est exécuté, cette console est la même que ce que nous voyons si nous ouvrons R
2. Dans la deuxième section, nous retrouvons;
    1. *Environment*, dans ce dernier nous retrouvons la liste des données ou objets à notre disposition. Par exemple, lorsque nous avons assigné la valeur 2 à la variable y, nous remarquons alors que RStudio garde en mémoire la valeur de cette variable.
    2. History: cette sous-section contient l'historique des commandes que nous avons exécutées. Si nous cliquons sur une ligne quelconque, nous verrons alors cette ligne s'écrire dans la console. Lorsque nous appuyons sur `Enter`, la ligne s'exécute.
3. La troisième section contient;
    1. Files: ici, nous pouvons naviguer directement dans le dossier dans lequel nous voulons; exécuter du code, créer des données, importer des données... etc. Nous verrons un peu plus loin plus en détail cette sous-section.
    2. Plots: dans cette sous-section, nous retrouvons nos graphiques que nous avons exécutés, nous pouvons les exporter directement à partir de là
    3. Packages: Ici, nous retrouvons les packages qui nous sont disponibles à télécharger ou qui le sont déjà (coché ou pas).
    4. Help: Cette sous-section est très importante, car elle nous permet trouver la documentation du langage R. Nous avons qu'à écrire ce que nous cherchons.

2.2 Créer un script

Afin de sauvegarder notre code R, nous pouvons ouvrir un nouveau script en cliquant sur le petit signe plus vert en haut à gauche. Ou simplement `Ctrl+Shift+N`

Un script est simplement un éditeur de code R, dans lequel nous pouvons exécuter notre code ligne par ligne en appuyant sur `Ctrl+Enter` ou en cliquant sur le bouton `Run`

Une fois votre script est créé, vous pouvez le sauvegarder à l'endroit que vous voulez.

3 Packages

Les packages R sont une partie importante, ce sont une collection de fonctions et de données créées par des individus (chercheurs, étudiants, des geeks. . . etc.) Au sein de la communauté open source. Lorsque nous installons R, un ensemble de packages est déjà inclus.

Afin de savoir quels packages sont installés, il suffit de taper `library()`

```
In [1]: library()
```

Lorsqu'on exécute une ligne de code, comme ce qu'on vient de faire avec `library()`, on demande à `R` de trouver la fonction et de l'exécuter. Habituellement, les fonctions requirent un argument, cette fonction est une exception et ne requiert aucun argument. Ainsi, il suffit de taper `library` sans les parenthèses.

```
In [2]: # library
```

3.1 *autocompletion*

Dans RStudio, et dans la majorité des IDE récents, il existe une option très utile appelée *autocompletion*. Elle devient très utile lorsque nous nous rappelons plus comment s'écrit exactement une option ou quels en sont les arguments.

Il suffit de taper sur la touche `tab`

Exemple: si nous voulons écrire la fonction `library()`, il suffit d'écrire `lib` et taper sur la touche `tab`. RStudio nous donne plusieurs choix de fonction que sont nom commence par les trois lettres `lib`

```
In [3]: lib
```

```
    Error in eval(expr, envir, enclos): object 'lib' not found
Traceback:
```

Une fois que le mot complet est saisi, en ouvrant des parenthèses, on peut encore taper sur la touche `tab` afin d'avoir la liste des arguments obligatoires ou optionnels à saisir.

```
In [4]: library()
```

3.2 Quels packages sont chargé dans l'environnement?

Afin de voir quels _packagers sont chargés dans l'environnement, il suffit de taper la commande `search()`

```
In [5]: search()
```

1. '.GlobalEnv' 2. 'jupyter:irkernel' 3. 'package:stats' 4. 'package:graphics' 5. 'package:grDevices' 6. 'package:utils' 7. 'package:datasets' 8. 'package:methods' 9. 'Autoloads' 10. 'package:base'

Un *package* que nous utiliserons beaucoup au début est le _package `MASS`. Afin que nous soyons sûrs de l'avoir, nous l'installons à nouveau avec la commande suivante:

```
In [ ]: # install.packages("MASS")
```

Nous pouvons voir dans l'onglet *packages* dans RSudio que nous l'avons et qu'il prêt à charger. Nous procédons au chargement de ce *package* on le cochant ou simplement (il faut vraiment s'habituer à travailler avec les commandes dans la console) avec la commande suivante:

On peut aussi installer un *package* avec l'IDE (Integrated development environment) de RStudio

```
In [ ]: # require(MASS)
```

Attention: R est sensible aux caractères *case sensitive*. Donc si j'écris:

```
In [ ]: # require(mass)
```

J'ai alors un message d'erreur ‘‘`there is no package called ‘mass’’’`

4 Assignation des variables

Comment peut-on assigner des valeurs à des variables? R reconnait les valeurs numériques telles qu'elles sont.

```
In [6]: 3
```

3

```
In [7]: 2
```

2

Toutesfois, il existe d'autres variables numériques écrites en caractère. Par exemple, π. lorsqu'on saisit `pi` dans R, il nous redonne la valeur numérique (arrondie) de π

```
In [8]: pi
```

3.14159265358979

Ce sont des variables déjà existantes dans `R`. Si l'on voulait chercher des valeurs non existantes dans R, ce dernier nous retourne un message d'erreur;

```
In [9]: x
```

```
    Error in eval(expr, envir, enclos): object 'x' not found
Traceback:
```

On peut assigner une valeur à une variable de différentes manières;

4.1 avec le signe =

```
In [10]: x=2
         x
```

2

```
In [11]: y=3
         y
```

3

4.2 La flèche vers la gauche <-

Par convention, nous utilison cette méthode

```
In [12]: x<-2
         x
```

2

```
In [13]: y<-3
```

Nous avons donné la valeur 2 à x et la valeur 3 à y. Ces valeurs sont gardées en mémoire, on peut d'ailleurs faire des opérations mathématiques sur ces valeurs. Par exemple on veut

$$x + y$$

```
In [14]: x+y
```

5

Si l'on voulait appliquer, un calcule sur une variable que nous n'y avons pas assigner une valeur auparavant, cela ne fonctionnerait pas, puisque R ne l'a pas gardé en mémoire.

```
In [15]: z+1
```

```
    Error in eval(expr, envir, enclos): object 'z' not found
Traceback:
```

Rappelons-nous que R est *case sensitive*, par exemple:

```
In [16]: X
```

```
    Error in eval(expr, envir, enclos): object 'X' not found
Traceback:
```

Nous avons essayé d'appeler X, mais il nous retourne un message d'erreur. Cela est dû à cause la majuscule.

nous avions donné à x la valeur `x<-2` et `y<-3`

```
In [17]: print(x)
         print(y)

[1] 2
[1] 3
```

Nous pouvons écraser la valeur de x en lui assignant la valeur de y;

```
In [18]: x<-y

In [19]: print(x)
         print(y)

[1] 3
[1] 3
```

Soit maintenant $z = 9$, on peut aussi faire ceci:

```
In [20]: x<-y<-z<-9

In [21]: print(x)
         print(y)
         print(z)

[1] 9
[1] 9
[1] 9
```

Remarquez que R commence toujours par la fin, la valeur 9 a été assignée à toutes les variables.

4.3 La flèche vers la droite ->

On peut aussi utiliser l'autre sens de la flèche (vers la droite) pour assigner des valeurs à des variables

```
In [22]: 15 ->p
```

Toutefois, nous restons dans la convention et utilisons la flèche vers la gauche

4.4 La fonction `assign()`

Nous pouvons aussi utliliser la fonction `assign()`

```
assign(x, value, pos = -1, envir = as.environment(pos),
       inherits = FALSE, immediate = TRUE)
```

```
In [23]: assign("q",30)
```

```
In [24]: q
```

30

Cette fonction est souvent utilisée à l'intérieur d'une boucle où l'on voudrait assigner une valeur quelconque à une variable qui peut changer lors des itérations

5 Les nombres, les caractères et les booléens

```
In [25]: num<-25
         string<-"bonjour"
         booleen<-TRUE
```

On peut appeler la variable `string` et elle nous retourne ceci:

```
In [27]: string
```

'bonjour'

Afin de donner une valeur de type *string* à une variable, on peut utiliser les doubles 'apostrophes', ou des "guillemets"

On peut assigner une valeur booléenne à une variable par `TRUE` ou `FALSE`, mais aussi par simplement `T` ou `F`

```
In [28]: booleen2 <-T
```

```
In [29]: booleen2
```

TRUE

Il est possible de savoir quel type possède une variable gardée en mémoire

```
In [30]: class(string)
```

'character'

```
In [31]: class(booleen)
```

'logical'

```
In [32]: class(num)
```

'numeric'

On peut aussi faire un test booléen sur le type d'une variable par
* `is.numeric(variable)` * `is.logical(variable)` * `is.character(variable)`

```
In [33]: is.logical(num)
```

FALSE

```
In [34]: is.numeric(num)
```

TRUE

5.1 Changement du type de variable

On peut aussi changer le type d'une variable

```
In [35]: as.character(num)
```

'25'

Remarquons les apostrophes. Toutefois, il faut faire attention avec les conversions; essayons de convertir un *string* en *numeric*

```
In [36]: # as.numeric(string)
```

Nous obtenons `NA` (not available). Car R ne sait pas comment traduire cette variable de type *string* en *numeric*.

Toutefois, il est possible de changer des booléens vers numérique. * TRUE=T=1 * FALSE=F=0

```
In [37]: as.numeric(booleen)
```

1

5.2 Tester le type de variable

On peut aussi faire un test booléen sur deux valeurs. Par exemple, on peut demander si une valeur est plus petite ou égale (ou supérieure ou égale) à une autre variable.

```
In [38]: num<100
```

TRUE

```
In [39]: num<=100
```

TRUE

```
In [40]: num>=100
```

FALSE

5.2.1 Test d'égalité

Le test sur l'égalité se fait par un double ==

```
In [41]: x==y
```

TRUE

```
In [42]: x==num
```

FALSE

5.2.2 Test d'inégalité

Pour ce qui est du test d'inégalité, on utilise `!=`

```
In [43]: x!=y
```

FALSE

```
In [44]: x!=num
```

TRUE

On peut aussi faire des tests logiques sur les valeurs de type string

```
In [45]: string2<-"bonjours"

In [46]: string==string2
```

FALSE

```
In [47]: string!=string2
```

TRUE

6 Les opérations sur le `workspace`

6.1 Répertoire courant

Afin de savoir dans quel répertoire nous travaillons, on peut utiliser `getwd()`

```
In [48]: getwd()
```

'/Users/nour/MEGA/Studies/ACT3035/AUT_2018'

On peut également voir dans l'onglet *files* à droite de l'écran dans RStudio le répertoire dans lequel on travaille. Pour ceux qui sont dans la version Linux, il suffit de taper `pwd` dans l'onglet *terminal*

On remarque que la fonction `getwd()` nous retourne une valeur de type string, on peut alors assigner cette valeur à une variable, par exemple:

```
In [49]: dir<-getwd()

In [50]: dir
```

'/Users/nour/MEGA/Studies/ACT3035/AUT_2018'

6.2 Changer le répertoire courant

On peut aussi changer le répertoire courant avec la fonction `setwd("repertoire/sous-repertoire")`

```
In [53]: setwd('/Users/nour/MEGA/Studies/ACT3035/AUT_2018')
```

```
In [54]: getwd()
```

'/Users/nour/MEGA/Studies/ACT3035/AUT_2018'

On retourne à notre répertoire original, on se rappelle que la variable `dir` contenant justement une valeur string du premier répertoire. On peut réassigner une nouvelle valeur à notre répertoire courant avec cette variable;

```
In [55]: setwd(dir)
```

```
In [56]: getwd()
```

'/Users/nour/MEGA/Studies/ACT3035/AUT_2018'

Toutefois, il est aussi possible de le faire via l'IDE avec **Session–> Set Working Directory –> Choose Directory**.
OU avec le raccourci: **Ctrl+Shift+H**

6.3 Lister les objets dans la mémoire

On peut avoir la liste des objets dans le répertoire courant avec la fonction `ls()`. Comme dans les commandes Linux dans le terminal (sans les parenthèses dans le cas de Linux)

```
In [57]: ls()
```

1. 'booleen' 2. 'booleen2' 3. 'dir' 4. 'num' 5. 'p' 6. 'q' 7. 'string' 8. 'string2' 9. 'x' 10. 'y' 11. 'z'

6.4 Supprimer un objets de la mémoire

Si l'on veut supprimer une variable, il suffit d'utiliser la fonction `rm(variable)`

```
In [58]: rm(x)
```

Regardons si x est encore là?

```
In [59]: ls()
```

1. 'booleen' 2. 'booleen2' 3. 'dir' 4. 'num' 5. 'p' 6. 'q' 7. 'string' 8. 'string2' 9. 'y' 10. 'z'

6.5 Vider complètement le *workspace*

Maintenant, on peut aussi vider tout le *workspace* avec;

```
In [60]: rm(list=ls())
```

Il est aussi possible de le faire avec le "ballet" dans l'onglet *Environment*. Pour résumer, une fonction est simplement un appel à un script créé auparavant. Certaines fonctions nécessitent des arguments, et d'autres pas.

7 Help

7.1 Aide sur les fonctions

Lorsqu'on ne se rappelle plus ce qu'une fonction fait, on peut appeler cette fonction avec le caractère?précède le nom de la fonction. Ça nous donne la documentation sur cette fonction.

```
In [61]: ?sqrt
```

L'autre façon d'avoir la documentation d'une fonction c'est de simplement écrire `help(nomFonction)`

```
In [62]: help(sqrt)
```

Si l'on ne se rappelle plus du nom exact de la fonction, on peut taper ce qu'on pense être le nom de la fonction précédée de `??`

```
In [63]: ??remove
```

Ça nous conduit vers l'onglet *help* en faisant une recherche avec le mot que nous avons tapé
Le mot `base::` qui précède le nom de la fonction veut dire de quel *package* provient cette fonction. Dans notre exemple on peut lire `base::rm`

7.2 La fonction `example`

Une autre fonction très utile `example(NomFonction)` qui nous donne un exemple de la fonction que nous recherchons. En plus de nous décrire les *packages* nécessaires (qui sont chargés).

```
In [64]: example(sqrt)
```

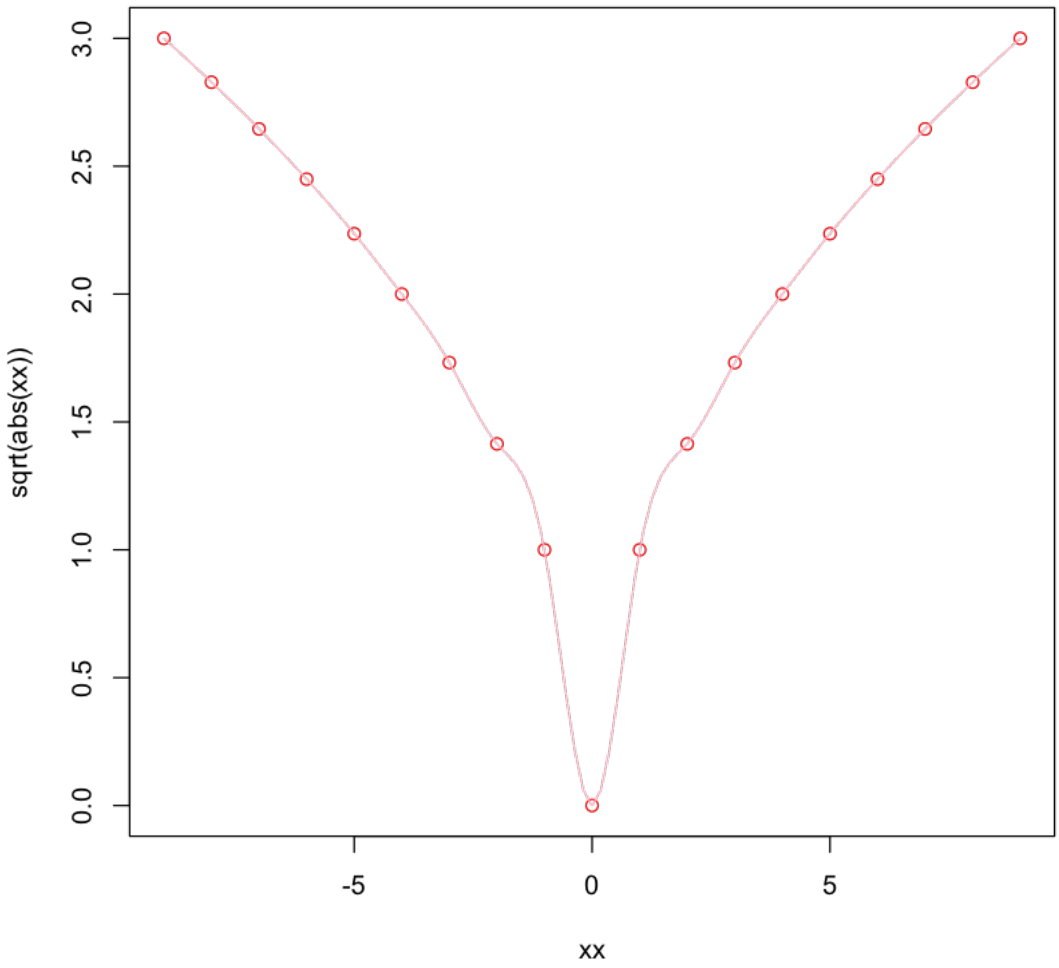
```
sqrt> require(stats) # for spline

sqrt> require(graphics)

sqrt> xx <- -9:9

sqrt> plot(xx, sqrt(abs(xx)),  col = "red")

sqrt> lines(spline(xx, sqrt(abs(xx)), n=101), col = "pink")
```




```
In [65]: example(exp)
```

```
exp> log(exp(3))
[1] 3

exp> log10(1e7) # = 7
[1] 7

exp> x <- 10^-(1+2*1:9)

exp> cbind(x, log(1+x), log1p(x), exp(x)-1, expm1(x))
          x
```

```
[1,] 1e-03 9.995003e-04 9.995003e-04 1.000500e-03 1.000500e-03
[2,] 1e-05 9.999950e-06 9.999950e-06 1.000005e-05 1.000005e-05
[3,] 1e-07 1.000000e-07 1.000000e-07 1.000000e-07 1.000000e-07
[4,] 1e-09 1.000000e-09 1.000000e-09 1.000000e-09 1.000000e-09
[5,] 1e-11 1.000000e-11 1.000000e-11 1.000000e-11 1.000000e-11
[6,] 1e-13 9.992007e-14 1.000000e-13 9.992007e-14 1.000000e-13
[7,] 1e-15 1.110223e-15 1.000000e-15 1.110223e-15 1.000000e-15
[8,] 1e-17 0.000000e+00 1.000000e-17 0.000000e+00 1.000000e-17
[9,] 1e-19 0.000000e+00 1.000000e-19 0.000000e+00 1.000000e-19
```

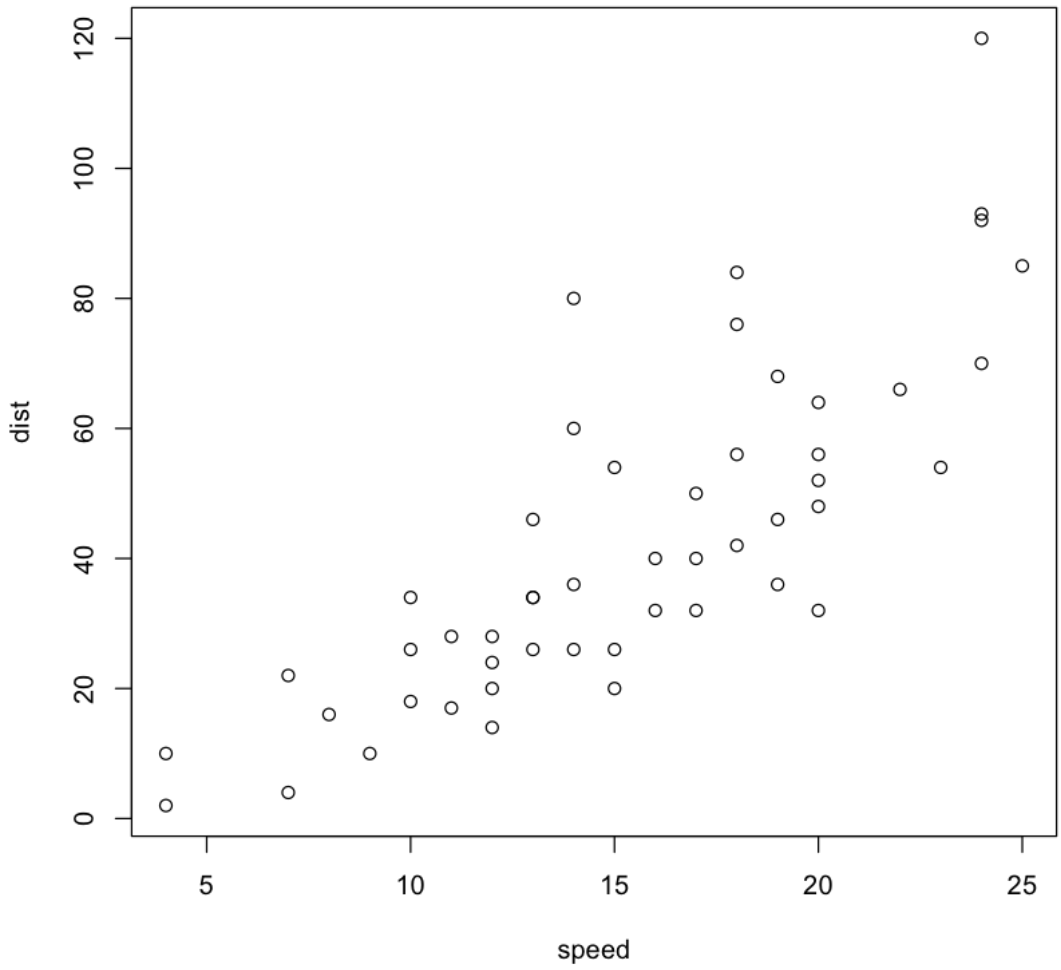
7.3 Aide sur les données

Il est aussi possible d'avoir plus d'informations sur les données préchargées.

```
In [66]: data()
```

Ce sont des bases de données de R qui sont disponibles par défaut

```
In [67]: plot(cars)
```



On peut aussi charger des données déjà disponibles dans un *package*

```
In [68]: require(MASS)
```

```
Loading required package: MASS
```

```
In [69]: data(Cars93)
```

Une fonction très utile afin d'avoir un sommaire rapide sur les données est `summary`

```
In [70]: summary(Cars93)
```

```
    Manufacturer      Model          Type       Min.Price          Price
 Chevrolet: 8     100     : 1   Compact:16   Min.   : 6.70   Min.   : 7.40
```

```
Ford     : 8    190E   : 1   Large  :11   1st Qu.:10.80   1st Qu.:12.20
Dodge    : 6    240    : 1   Midsize:22   Median :14.70   Median :17.70
Mazda    : 5    300E   : 1   Small  :21   Mean   :17.13   Mean   :19.51
Pontiac  : 5    323    : 1   Sporty :14   3rd Qu.:20.30   3rd Qu.:23.30
Buick    : 4    535i   : 1   Van    : 9   Max.   :45.40   Max.   :61.90
(Other)  :57    (Other):87
  Max.Price       MPG.city      MPG.highway                  AirBags
Min.   : 7.9   Min.   :15.00   Min.   :20.00   Driver & Passenger:16
1st Qu.:14.7   1st Qu.:18.00   1st Qu.:26.00   Driver only       :43
Median :19.6   Median :21.00   Median :28.00   None              :34
Mean   :21.9   Mean   :22.37   Mean   :29.09
3rd Qu.:25.3   3rd Qu.:25.00   3rd Qu.:31.00
Max.   :80.0   Max.   :46.00   Max.   :50.00

DriveTrain  Cylinders    EngineSize      Horsepower         RPM
4WD  :10   3     : 3   Min.   :1.000   Min.   : 55.0   Min.   :3800
Front:67   4     :49   1st Qu.:1.800   1st Qu.:103.0   1st Qu.:4800
Rear :16   5     : 2   Median :2.400   Median :140.0   Median :5200
           6     :31   Mean   :2.668   Mean   :143.8   Mean   :5281
           8     : 7   3rd Qu.:3.300   3rd Qu.:170.0   3rd Qu.:5750
           rotary: 1   Max.   :5.700   Max.   :300.0   Max.   :6500

 Rev.per.mile  Man.trans.avail Fuel.tank.capacity   Passengers
Min.   :1320   No :32          Min.   : 9.20      Min.   :2.000
1st Qu.:1985   Yes:61          1st Qu.:14.50      1st Qu.:4.000
Median :2340                   Median :16.40      Median :5.000
Mean   :2332                   Mean   :16.66      Mean   :5.086
3rd Qu.:2565                   3rd Qu.:18.80      3rd Qu.:6.000
Max.   :3755                   Max.   :27.00      Max.   :8.000

    Length        Wheelbase         Width        Turn.circle
Min.   :141.0   Min.   : 90.0   Min.   :60.00   Min.   :32.00
1st Qu.:174.0   1st Qu.: 98.0   1st Qu.:67.00   1st Qu.:37.00
Median :183.0   Median :103.0   Median :69.00   Median :39.00
Mean   :183.2   Mean   :103.9   Mean   :69.38   Mean   :38.96
3rd Qu.:192.0   3rd Qu.:110.0   3rd Qu.:72.00   3rd Qu.:41.00
Max.   :219.0   Max.   :119.0   Max.   :78.00   Max.   :45.00

Rear.seat.room   Luggage.room      Weight         Origin              Make
Min.   :19.00   Min.   : 6.00   Min.   :1695   USA    :48   Acura Integra: 1
1st Qu.:26.00   1st Qu.:12.00   1st Qu.:2620   non-USA:45   Acura Legend : 1
Median :27.50   Median :14.00   Median :3040                Audi 100     : 1
Mean   :27.83   Mean   :13.89   Mean   :3073                Audi 90      : 1
3rd Qu.:30.00   3rd Qu.:15.00   3rd Qu.:3525                BMW 535i     : 1
Max.   :36.00   Max.   :22.00   Max.   :4105                Buick Century: 1
NA's   :2       NA's   :11                                  (Other)      :87
```

Ça nous donne les variables trouvées dans cette base de données ainsi qu'une statistique descriptive sur chacune des variables

Min.
1st Qu.
Median
Mean
3rd Qu.
Max.

```
In [71]: head(Cars93)
```

Manufacturer	Model	Type	Min.Price	Price	Max.Price	MPG.city	MPG.highway	AirBags
Acura	Integra	Small	12.9	15.9	18.8	25	31	None
Acura	Legend	Midsize	29.2	33.9	38.7	18	25	Driver &
Audi	90	Compact	25.9	29.1	32.3	20	26	Driver c
Audi	100	Midsize	30.8	37.7	44.6	19	26	Driver &
BMW	535i	Midsize	23.7	30.0	36.2	22	30	Driver c
Buick	Century	Midsize	14.2	15.7	17.3	22	31	Driver c

8 Plus de resources

- **The R Project for Statistical Computing**: (http://www.r-project.org/) Premier lieu où.
- **The Comprehensive R Archive Network**: (http://cran.r-project.org/) C,est là ou se trouve le logiciel R, avec des miliers de *packages*, il s,y trouve aussi des exemples et même des livres!
- **R-Forge**: (http://r-forge.r-project.org/) Une autre place où des *packages* sont sauvegardé, on y trouve aussi des *packages* tout récemment développés
- **Rlanguage reddit**: (https://www.reddit.com/r/Rlanguage) On y trouve toutes sortes d'informations ou question exemple

1_2_cours

October 15, 2018

Table of Contents

1 les opérateurs de base

R est d'abord une calculatrice mathématique:

1.1 Arithmetic

D'abord créons quelques variables;

```
In [1]: x <- 12
        y <- 10
        z<--1 #ou  z <- -1
```

```
In [2]: x+y+z
```

21

```
In [3]: x-y-z
```

3

1.2 multiplication / division

```
In [4]: x*y
```

120

```
In [5]: x/y
```

1.2

1.3 Puissance

```
In [6]: x^2
```

144

```
In [7]: x**2
```

144

1.4 Arrondir

1.4.1 round()

```
In [8]: pi
```

3.14159265358979

```
In [9]: round(pi,2)
```

3.14

1.4.2 floor

```
In [10]: floor(pi)
```

3

1.4.3 ceiling

```
In [11]: ceiling(pi)
```

4

1.4.4 valeur absolue

```
In [12]: z
```

-1

```
In [13]: abs(z)
```

1

On peut aussi effectuer un calcul à l'intérieur d'un autre calcul. Essayons de calculer la racine carrée de z

```
In [14]: # sqrt(z)

In [15]: sqrt(abs(z))
```

1

1.4.5 les parenthèses

```
In [16]: x^(1/2)
```

3.46410161513775

```
In [17]: x^1/2
```

6

1.5 logarithme

```
In [18]: log(x)
```

2.484906649788

```
In [19]: log10(x)
```

1.07918124604762

```
In [20]: exp(1)
```

2.71828182845905

1.6 Factoriel

```
In [21]: factorial(4)
```

24

```
In [22]: 4*3*2*1
```

24

2 Les vecteurs

Nous avons vu que nous avons créé des variables, nous leur avons donnée des valeurs et nous avons effectué des opérations mathématiques sur ces variables. Nous avons vu que nous avons chargé des *packages*, nous avons également appelé des fonctions des données venant de *packages* de base ou incluses dans des *packages* que nous avons chargés.

On commence à comprendre alors que dans R, tous les éléments sont des **objets**. Ces derniers sont créés et interagissent entre elles par des instructions (des lignes de code) du programmeur. Par exemple si nous créons une variable de x de valeur 21, nous venons alors de créer un seul objet x ayant une valeur 21

```
In [23]: x<-21
```

Nous pouvons en tout moment faire appel à ce vecteur par son attribut;

```
In [24]: x
```

21

On peut aussi créer une collection de valeurs à l'intérieur d'**un seul** objet. C'est ce que nous appelons **un vecteur**

```
In [25]: vecteur <-c(1,2,3,4,5)
```

Notre vecteur est un ensemble d'entiers

$$vecteur = \{1, 2, 3, 4, 5\}$$

Regardons la valeur de cet objet que R a gardé en mémoire;

```
In [26]: vecteur
```

1. 1 2. 2 3. 3 4. 4 5. 5

Regardons maintenant le type de cet objet;

```
In [27]: class(vecteur)
```

'numeric'

Tel qu'attendu, R a automatiquement reconnu le type de ce vecteur sans avoir à le définir. Contrairement à d'autres langages de programmation, tel que `C++`, R a donné le type `numeric` à cet objet.

2.1 autres façons de créer des vecteurs

```
In [28]: vecteur <-1:5
```

```
In [29]: vecteur
```

1. 1 2. 2 3. 3 4. 4 5. 5

```
In [30]: vecteur <-10:17
```

```
In [31]: vecteur
```

1. 10 2. 11 3. 12 4. 13 5. 14 6. 15 7. 16 8. 17

Ceci fonctionne aussi pour les décimales

```
In [32]: vecteur <- 17.5:22.5
```

```
In [33]: vecteur
```

1. 17.5 2. 18.5 3. 19.5 4. 20.5 5. 21.5 6. 22.5

On remarque que les deux façons qu'on vient de voir créent une suite de données avec un saut de 1 entre chaque valeur. On peut utiliser la fonction `seq(from = ..., to = ..., by = ...)` afin de changer le saut.

```
In [34]: # ?seq
```

```
In [35]: vecteur <-seq(from = 17.5,to = 29.5,by = 2)
```

```
In [36]: vecteur
```

1. 17.5 2. 19.5 3. 21.5 4. 23.5 5. 25.5 6. 27.5 7. 29.5

2.2 Extraction des valeurs

Supposons qu'on voudrait extraire la première valeur se trouvant dans le vecteur;

```
In [37]: vecteur # ce vecteur avait comme valeurs
```

1. 17.5 2. 19.5 3. 21.5 4. 23.5 5. 25.5 6. 27.5 7. 29.5

```
In [38]: vecteur[1] #le premier objet
```

17.5

Le troisième objet maintenant;

```
In [39]: vecteur[3]
```

21.5

```
In [40]: vecteur[1:3] # du premier au troisième objet
```

1. 17.5 2. 19.5 3. 21.5

Si l'on voulait tous sauf le nième élément, on utilise le signe (-)

```
In [41]: vecteur [-1] # tout sauf le premier
```

1. 19.5 2. 21.5 3. 23.5 4. 25.5 5. 27.5 6. 29.5

```
In [42]: vecteur[-(1:3)] # tous sauf les trois premiers objets
```

1. 23.5 2. 25.5 3. 27.5 4. 29.5

2.3 Combinaison de vecteurs

Puisque les vecteurs sont des objets ou une collection d'objets, on peut combiner plusieurs vecteurs et effectuer des opérations sur ces derniers.

Créons trois vecteurs;

```
In [43]: x<-c(1,2,3)
         y <- c(.17, .66, .44)
         z<-11
```

```
In [44]: vecteur <-c(x,y,z)
```

```
In [45]: vecteur
```

1. 1 2. 2 3. 3 4. 0.17 5. 0.66 6. 0.44 7. 11

```
In [46]: vecteur <-c(x,y,z, -123:-126)
```

```
In [47]: vecteur
```

1. 1 2. 2 3. 3 4. 0.17 5. 0.66 6. 0.44 7. 11 8. -123 9. -124 10. -125 11. -126

2.4 Taille des vecteurs

```
In [48]: length(vecteur)
```

11

2.5 Plusieurs types

Lorsque nous créons un vecteur, R se charge de donner un type à ce dernier. Toutefois, il est possible de mélanger les données de types différents.

2.5.1 Numeric

Nous avons vu plusieurs exemples de ce type auparavant

2.5.2 Character

```
In [49]: vect_stri <-c("bonjour", "Hi", Buongiorno")
```

```
In [50]: vect_stri
```

1. 'bonjour' 2. 'Hi'

Regardons maintenant la `class` de cette variable;

```
In [51]: class(vect_stri)
```

'character'

2.5.3 logical

```
In [52]: vect_bool <- c(T,F,F,F,T,F)
```

```
In [53]: vect_bool
```

1. TRUE 2. FALSE 3. FALSE 4. FALSE 5. TRUE 6. FALSE

```
In [54]: class(vect_bool)
```

'logical'

2.5.4 Mélange

On peut aussi avoir de multiples class dans un seul vecteur;

```
In [55]: vect_melange_type<-c(123,"bonjour", T)
```

```
In [56]: vect_melange_type
```

1. '123' 2. 'bonjour' 3. 'TRUE'

On remarque que R s'est chargé de tout transformer en type caractère, car "bonjour" ne peut pas prendre le numérique

```
In [57]: class(vect_melange_type)
```

'character'

2.6 Opértation sur les vecteurs

Créons notre vecteur qui nous sert d'exemple'

```
In [58]: vecteur <- c(1:5)
         vecteur
```

1. 1 2. 2 3. 3 4. 4 5. 5

Multiplions ce vecteur par deux, donc toutes les valeurs contenues à l'intérieur de ce dernier;

```
In [59]: vecteur*2
```

1. 2 2. 4 3. 6 4. 8 5. 10

Ou appliquons une simple adition à lui même;

```
In [60]: vecteur+vecteur
```

1. 2 2. 4 3. 6 4. 8 5. 10

Si l'on ajoute 5 au vecteur, R se charge d'ajouter 5 à chaque élément du vecteur

```
In [61]: vecteur+5
```

1. 6 2. 7 3. 8 4. 9 5. 10

```
In [62]: vecteur**2
```

1. 1 2. 4 3. 9 4. 16 5. 25

```
In [63]: vecteur^2
```

1. 1 2. 4 3. 9 4. 16 5. 25

Un autre façon de créer le même vecteur;

```
In [64]: vecteur2 <- 2*(1:5)
         vecteur2
```

1. 2 2. 4 3. 6 4. 8 5. 10

```
In [65]: (vecteur*3+vecteur2^2+17)/2
```

1. 12 2. 19.5 3. 31 4. 46.5 5. 66

2.6.1 Attention!

Note Ces opérations fonctionnent sur des vecteurs de même taille. Essayons par exemple d'additionner deux vecteurs de différente taille; le premier contient cinq éléments et le second en contient six.

```
In [66]: vecteur5 <-c(1:5)
         vecteur6 <-c(1:6)
```

```
In [67]: vecteur5+vecteur6
```

```
Warning message in vecteur5 + vecteur6:
''longer object length is not a multiple of shorter object length''
```

1. 2 2. 4 3. 6 4. 8 5. 10 6. 7

Nous avons quand même un résultat, mais nous avons un avertissement que la taille des deux vecteurs est différente.

On voit que le résultat contient six éléments, `R` s'est chargé de réappliquer le même calcul (adition) entre le sixième élément du vecteur6 et le premier élément du vecteur5 puisque le sixième de ce dernier est manquant.

3 Sequences

```
In [68]: ?seq
```

Cette fonction sert à générer une série de séquence régulière. Les arguments sont; * `from`: qui est le début de la séquence qu'on voudrait générer. * `to`: la fin de la séquence * `by`: l'incrément avec lequel on veut augmenter notre séquence

```
In [69]: seq(from = 1, to = 5,by=.3)
```

1. 1 2. 1.3 3. 1.6 4. 1.9 5. 2.2 6. 2.5 7. 2.8 8. 3.1 9. 3.4 10. 3.7 11. 4 12. 4.3 13. 4.6 14. 4.9

Un autre exemple où l'on crée un incrément de `pi` dans une série de 1 à 10;

```
In [70]: seq(1, 10, pi)
```

1. 1 2. 4.14159265358979 3. 7.28318530717959

Si l'on omet de mettre tous les arguments, l'incrément devient alors 1 par défaut.

```
In [71]: seq(5)
```

1. 1 2. 2 3. 3 4. 4 5. 5

Cela fonctionne aussi si l'on veut générer des nombres négatifs

```
In [72]: seq(-1,-5, -.3)
```

1. -1 2. -1.3 3. -1.6 4. -1.9 5. -2.2 6. -2.5 7. -2.8 8. -3.1 9. -3.4 10. -3.7 11. -4 12. -4.3 13. -4.6 14. -4.9

Cette fonction nous sert plus souvent lorsqu'on veut générer des vecteurs. D'ailleurs, vérifions si une séquence générée possède les mêmes valeurs qu'un vecteur.

```
In [73]: seq(5)
```

1. 1 2. 2 3. 3 4. 4 5. 5

```
In [74]: 1:5
```

1. 1 2. 2 3. 3 4. 4 5. 5

```
In [75]: seq(5)==1:5
```

1. TRUE 2. TRUE 3. TRUE 4. TRUE 5. TRUE

Dans la ligne de code ci-haut on s'aperçoit que chaque valeur de la séquence est égale à chaque valeur du vecteur. Si l'on veut savoir par une seule réponse booléenne si toutes les valeurs du vecteur sont égales.

```
In [76]: all(seq(5)==1:5)
```

TRUE

Ou le contraire, est-ce toutes les valeurs sont différentes?

```
In [77]: !all(seq(5)==1:5)
```

FALSE

4 Plus de resources

- **The R Project for Statistical Computing**: (http://www.r-project.org/) Premier lieu où.
- **The Comprehensive R Archive Network**: (http://cran.r-project.org/) C,est là ou se trouve le logiciel R, avec des miliers de *packages*, il s,y trouve aussi des exemples et même des livres!
- **R-Forge**: (http://r-forge.r-project.org/) Une autre place où des *packages* sont sauvegardé, on y trouve aussi des *packages* tout récemment développés
- **Rlanguage reddit**: (https://www.reddit.com/r/Rlanguage) On y trouve toutes sortes d'informations ou question exemple

1_4_Exercices_vecteurs-solutions

October 15, 2018

1 Question #1

Dans R, la valeur de la variable du nombre π est nommée « pi ». Afficher cette valeur à l'écran avec une précision de 3 puis de 5 décimales.

```
In [1]: pi
        round(pi,3)
        round(pi, 5)
```

3.14159265358979
3.142
3.14159

2 Question #2

Créer le vecteur « vec1 » contenant la suite des entiers de 1 à 12. Ajouter à la fin de ce vecteur les valeurs 16, 17, 18.

```
In [26]: vec1<-1:12
         vec1
```

1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12

```
In [27]: vec1<-c(vec1, 16:18)
         vec1
```

1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 16 14. 17 15. 18

3 Question #3

Créer le vecteur « vec2 » contenant les valeurs suivantes : 0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0.

```
In [79]: seq(0,5,.5)
```

1. 0 2. 0.5 3. 1 4. 1.5 5. 2 6. 2.5 7. 3 8. 3.5 9. 4 10. 4.5 11. 5

4 Question #4

Créer le vecteur « vec3 » contenant tous les multiples de 2 compris entre 1 et 50.

```
In [1]: vec3<-seq(2,50, 2)
        vec3
```

1. 2 2. 4 3. 6 4. 8 5. 10 6. 12 7. 14 8. 16 9. 18 10. 20 11. 22 12. 24 13. 26 14. 28 15. 30 16. 32 17. 34 18. 36 19. 38 20. 40 21. 42 22. 44 23. 46 24. 48 25. 50

5 Question #5

Soit les deux vecteurs suivants;

```
In [7]: x=c(0.069, 0.0813, 0.0753, 0.0856, 0.0646)
        y=c(1341.05, 1393.88, 1324.88, 1186.97, 1051.55)
```

Calculer la valeur de z qui simplement le résultat de la multiplication de ces deux vecteurs;

```
In [8]: z=y*y
        print(z)
```

```
[1] 1798415 1942901 1755307 1408898 1105757
```

6 Question #6

Vous avez une liste de courriels `A` de l'année 2017 et une autre liste de courriel `B` de l'année 2018 suivante;

```
In [3]: A=c("bantonignetti0@bloomberg.com",
        "dgillogley1@cam.ac.uk",
        "stabart2@gmpg.org",
        "fchatenet3@digg.com",
        "hmattiussi4@cdc.gov",
        "rlafont5@spiegel.de",
        "blivingston6@bloglovin.com",
        "cdevuyst7@twitter.com",
        "jhuddleston8@cornell.edu",
        "kearry9@wp.com")
```

```
In [4]: B=c("bantonignetti0@bloomberg.com",
        "dgillogley1@cam.ac.uk",
        "stabart2@gmpg.org",
        "fchatenet3@digg.com",
        "hmatiussi4@cdc.gov",
        "rlafont5@spiegel.de",
        "blivingston6@bloglovin.com",
        "cdevuyst7@twitter.com",
        "jhuddleston8@cornell.edu",
        "kearry9@wp.com")
```

Vous voulez savoir si un changement a eu lieu entre les listes, affichez un vecteur booléen qui vous informe par `TRUE` si le i'ème élément du vecteur A est égale au i'ème élément du vecteur B;

```
In [5]: A==B
```

1. TRUE 2. TRUE 3. TRUE 4. TRUE 5. FALSE 6. TRUE 7. TRUE 8. TRUE 9. TRUE 10. TRUE

7 Question 7

Vous voulez savoir quel a été le rendement quotidien du S&P 500 (^GSPC) du 27 au 31 août de l'année courante. Vous pouvez allez sur ce site afin d'extraire les prix du marché Open et Close

Ensuite, créer un vecteur `R` qui tout simplement le rendement quotidien en utilisant la formule suivante:

$$R = \frac{\text{Prix}_{open} - \text{Prix}_{Close}}{\text{Prix}_{open}}$$

Mais attention, on veut seulement voir 4 décimales!

```
In [6]: # Attention! ces prix sont ceux de l'année 2018
        open=c(2884.69, 2901.45, 2900.62, 2908.94, 2898.37)
        close=c(2896.74, 2897.52, 2914.04, 2901.13, 2901.52)
        R=(open-close)/open
        print(round(R,4))
```

```
[1] -0.0042  0.0014 -0.0046  0.0027 -0.0011
```

2_1_cours

October 15, 2018

Table of Contents

0.1 seq (suite)

Cela fonctionne aussi si l'on veut générer des nombres négatifs

```
In [70]: seq(-1,-5, -1)
```

1. -1 2. -2 3. -3 4. -4 5. -5

Cette fonction nous sert plus souvent lorsqu'on veut générer des vecteurs. D'ailleurs, vérifions si une séquence générée possède les mêmes valeurs qu'un vecteur.

```
In [57]: vec
         seq(5)
```

1. 1 2. 2 3. 3 4. 4 5. 5

```
In [58]: 1:5
```

1. 1 2. 2 3. 3 4. 4 5. 5

```
In [59]: seq(5)==1:5
```

1. TRUE 2. TRUE 3. TRUE 4. TRUE 5. TRUE

Dans la ligne de code ci-haut on s'aperçoit que chaque valeur de la séquence est égale à chaque valeur du vecteur. Si l'on veut savoir par une seule réponse booléenne si toutes les valeurs du vecteur sont égales.

```
In [60]: all(seq(5)==1:5)
```

TRUE
Ou le contraire, est-ce toutes les valeurs sont différentes?

```
In [61]: !all(seq(5)==1:5)
```

FALSE

0.2 Replicate

Replicate Elements of Vectors and Lists

```
In [71]: ?rep
```

```
In [1]: rep(1, 5)
```

1. 1 2. 1 3. 1 4. 1 5. 1

```
In [2]: nb_fois=5
        rep("qqch", nb_fois)
```

1. 'qqch' 2. 'qqch' 3. 'qqch' 4. 'qqch' 5. 'qqch'
On peut aussi répéter un vecteur *n* nombre de fois

```
In [3]: rep(c(1:5), 3)
```

1. 1 2. 2 3. 3 4. 4 5. 5 6. 1 7. 2 8. 3 9. 4 10. 5 11. 1 12. 2 13. 3 14. 4 15. 5

```
In [4]: rep(c('bonjour', 'hello', "Hola"), 4)
```

1. 'bonjour' 2. 'hello' 3. 'Hola' 4. 'bonjour' 5. 'hello' 6. 'Hola' 7. 'bonjour' 8. 'hello' 9. 'Hola' 10. 'bonjour' 11. 'hello' 12. 'Hola'

0.3 Statistiques descriptives

Générons quelques vecteurs

```
In [64]: x_v<-c(1,5,7,3,2,3,6,8,7,3)
```

```
In [65]: y_v <- seq(1.1,2,.1)
```

```
In [66]: z_v <- x_v^2/-.5
```

0.3.1 sum

```
In [67]: sum(x_v)
```

45

```
In [9]: sum(y_v)
```

15.5

```
In [10]: sum(z_v)
```

-510

0.3.2 lenght

Combien d'objets y'a-t-il à l'intérieur du vecteur

```
In [11]: length(x_v)
```

10

0.3.3 min/max

Le minimum ou le maximum à l'intérieur d'un vecteur

```
In [12]: min(x_v)
```

1

```
In [13]: max(x_v)
```

8
Si l'on veut la valeur minimale à l'intérieur de plusieur vecteurs

```
In [14]: min(x_v, z_v)
```

-128

0.3.4 range

Si l'on veut le minimum et le maximum en même temps

```
In [15]: range(x_v, z_v)
```

1. -128 2. 8

0.3.5 average

la fonction `mean` nous donne la moyenne d'un vecteur

```
In [16]: mean(x_v)
```

4.5

0.3.6 median

```
In [17]: median(x_v)
```

4

0.3.7 Standard Deviation

```
In [18]: sd(x_v)
```

2.41522945769824

0.3.8 Variance

```
In [19]: var(x_v)
```

5.83333333333333

- **The R Project for Statistical Computing**: (http://www.r-project.org/) Premier lieu où.
- **The Comprehensive R Archive Network**: (http://cran.r-project.org/) C,est là ou se trouve le logiciel R, avec des miliers de *packages*, il s,y trouve aussi des exemples et même des livres!
- **R-Forge**: (http://r-forge.r-project.org/) Une autre place où des *packages* sont sauvegardé, on y trouve aussi des *packages* tout récemment développés
- **Rlanguage reddit**: (https://www.reddit.com/r/Rlanguage) On y trouve toutes sortes d'informations ou question exemple

2_2_cours

October 15, 2018

Table of Contents

1 Matrices

Une matrice est un objet constitué de données en deux dimensions, soit des lignes et des colonnes. Chaque élément de la matrice est situé à l'intersection d'une ligne et d'une colonne.

```
In [56]: A<- matrix(c(6,8,1,1,4,2), nrow = 2, ncol = 3)
```

```
In [57]: A
```

```
6  1  4
8  1  2
```

Il arrive souvent qu'on veuille transposer une matrice. Pour ce faire, il suffit de l'inclure à l'intérieur de `t(matrice)`

```
In [58]: t(A)
```

6 8
1 1
4 2

Note Lorsqu'on transpose un vecteur, R transforme ce vecteur en une matrice à une seule dimension:

```
In [59]: vec<-1:5
```

```
In [61]: t(vec)
```

1 2 3 4 5

Note la fonction `dim()` donne les dimensions d'une matrice. Si l'on vérifie la dimension du vecteur.

```
In [62]: dim(vec)
```

```
NULL
```

Bien évidemment il nous retourne une valeur nulle. Mais lorsqu'on transforme ce vecteur en matrice, on obtient;

```
In [63]: dim(t(vec))
```

1. 1 2. 5

Ce qui veut dire que notre matrice est composée d'une seule ligne et cinq colonnes

1.1 Extraction d'un élément d'une matrice

Si l'on veut extraire un élément d'une matrice, il suffit d'indiquer ses coordonnées `[ligne, colonne]`

```
In [64]: A[1,3]
```

4

```
In [65]: A
```

6 1 4
8 1 2

Lorsqu'on veut extraire un élément qui n'existe pas dans la matrice, on obtien alors le message d'erreur `subscript out of bounds`. Un message d'erreur que nous verrons souvent!

```
In [67]: A[1,4]
```

```
Error in A[1, 4]: subscript out of bounds
Traceback:
```

Si l'on omet de mettre une valeur au numéro de colonne ou de ligne, on obtient la ligne ou la colonne complète

```
In [68]: A[,1]
```

1. 6 2. 8

Lorsqu'on crée une matrice, nous ne sommes pas obligés d'indiquer le nombre de colonnes ou de lignes en même temps. Un seul argument suffit.

```
In [81]: B<-matrix(seq(1,9.5,.5), 3)
```

```
In [82]: B
```

1.0	2.5	4.0	5.5	7.0	8.5
1.5	3.0	4.5	6.0	7.5	9.0
2.0	3.5	5.0	6.5	8.0	9.5

Si l'on veut extraire la deuxième et la quatrième colonne

```
In [83]: B[,c(2,4)]
```

2.5	5.5
3.0	6.0
3.5	6.5

```
In [78]: B<-t(B)
```

```
In [79]: B
```

1.0	1.5	2.0
2.5	3.0	3.5
4.0	4.5	5.0
5.5	6.0	6.5
7.0	7.5	8.0
8.5	9.0	9.5

Si l'on veut extraire la troisième et la cinquième ligne;

```
In [80]: B[c(3,5),]
```

4	4.5	5
7	7.5	8

1.2 diag

Cette fonction crée une matrice identité, c'est une matrice carrée avec des 1 sur la diagonale et des 0 partout ailleurs.

```
In [84]: diag(5)
```

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

1.3 Opértaion sur les matrices

On peut aussi appliquer des fonctions mathématiques sur des matrices comme nous l'avons fait avec des vecteurs

```
In [85]: A**2
```

```
    36   1   16
    64   1   4
```

```
In [86]: B/2
```

```
    0.50   1.25   2.00   2.75   3.50   4.25
    0.75   1.50   2.25   3.00   3.75   4.50
    1.00   1.75   2.50   3.25   4.00   4.75
```

```
In [89]: C<-B*2
```

```
In [90]: B+C
```

```
    3.0   7.5    12.0   16.5   21.0   25.5
    4.5   9.0    13.5   18.0   22.5   27.0
    6.0   10.5   15.0   19.5   24.0   28.5
```

Créons une matrice `A=5X3`. Cette matrice contient les températures en Fahrenheit des trois villes "Fairbanks","San Francisco" et "Chicago" (nom de colonnes). Les lignes sont les données du mois de mars 2012 au mois de mars 2016.

```
In [6]: A<-matrix(c(30,32,31,27,36,72,60,78,67,71,55,57,56,55,49),ncol=3)
        A
```

```
    30   72   55
    32   60   57
    31   78   56
    27   67   55
    36   71   49
```

Convertissons ces données en Celsius avec la formule suivante;

$$C = \frac{F - 32}{1.8000} \tag{1}$$

```
In [7]: A<-round((A-32)/1.8,0)
        A
```

```
    -1   22   13
    0    16   14
    -1   26   13
    -3   19   13
    2    22   9
```

On peut donner des noms à chacune des colonnes avec la fonction `colnames()`

```
In [8]: colnames(A)<-c("Fairbanks","San Francisco","Chicago")
```

et des nom aux lignes avec la fonction `rownames()`

```
In [9]: rownames(A)<-paste("3/",12:16,sep='')
```

La fonction `paste` ci-haut permet de concatener des caractères

```
In [10]: paste("3/",12:16,sep='___')
```

1. '3/___12' 2. '3/___13' 3. '3/___14' 4. '3/___15' 5. '3/___16'

```
In [11]: A
```

	Fairbanks	San Francisco	Chicago
3/12	-1	22	13
3/13	0	16	14
3/14	-1	26	13
3/15	-3	19	13
3/16	2	22	9

Créons une autre matrice `B`

```
In [12]: B<-matrix(c(88,85,83,81,78,62,61,54,60,65,90,92,91,89,90),ncol=3)
         colnames(B)<-c("Los Angeles","Seattle","Honolulu")
         rownames(B)<-paste("3/",12:16,sep='')
```

```
In [13]: B<-round((B-32)/1.8,0)
         B
```

	Los Angeles	Seattle	Honolulu
3/12	31	17	32
3/13	29	16	33
3/14	28	12	33
3/15	27	16	32
3/16	26	18	32

1.4 cbind

La fonction `cbind` permet de concaténer deux matrices ensemble en colonne

```
In [14]: cbind(A,B)
```

	Fairbanks	San Francisco	Chicago	Los Angeles	Seattle	Honolulu
3/12	-1	22	13	31	17	32
3/13	0	16	14	29	16	33
3/14	-1	26	13	28	12	33
3/15	-3	19	13	27	16	32
3/16	2	22	9	26	18	32

1.5 rbind

La fonction `rbind` permet de concaténer deux matrices ensemble une par-dessus l'autre

```
In [15]: rbind(A,B)
```

	Fairbanks	San Francisco	Chicago
3/12	-1	22	13
3/13	0	16	14
3/14	-1	26	13
3/15	-3	19	13
3/16	2	22	9
3/12	31	17	32
3/13	29	16	33
3/14	28	12	33
3/15	27	16	32
3/16	26	18	32

1.6 matrcice en vecteur

On peut aussi transformer une matrice en un vecteur;

Reprenons la matrice que nous avons créée avec la fonction `rbind`. On lui donne le nom "mat_comb"

```
In [16]: mat_comb<-cbind(A,B)
```

On la transforme en vecteur avec `c(nomMatrice)`

```
In [17]: c(mat_comb)
```

1. -1 2. 0 3. -1 4. -3 5. 2 6. 22 7. 16 8. 26 9. 19 10. 22 11. 13 12. 14 13. 13 14. 13 15. 9 16. 31 17. 29 18. 28 19. 27 20. 26 21. 17 22. 16 23. 12 24. 16 25. 18 26. 32 27. 33 28. 33 29. 32 30. 32

1.7 Quelques fonctions statistiques sur les matrices

```
In [18]: mat_comb
```

	Fairbanks	San Francisco	Chicago	Los Angeles	Seattle	Honolulu
3/12	-1	22	13	31	17	32
3/13	0	16	14	29	16	33
3/14	-1	26	13	28	12	33
3/15	-3	19	13	27	16	32
3/16	2	22	9	26	18	32

Lorsqu'on applique la fonction `min`, on obtient alors la valeur minimale de toutes les valeurs contenues dans la matrice

```
In [19]: min(mat_comb)
```

-3

```
In [20]: max(mat_comb)
```

33

```
In [21]: range(mat_comb)
```

1. -3 2. 33

```
In [22]: sd(mat_comb)
```

11.1923619275487

Les statistiques que nous venons d'obtenir, sont applquées à toutes les valeurs de la matrice. Et si nous voulions des statistiques par ligne ou par colonne

```
In [23]: rowMeans(mat_comb)
```

3/12 19 **3/13** 18 **3/14** 18.5 **3/15** 17.3333333333333 **3/16** 18.1666666666667

```
In [24]: colMeans(mat_comb)
```

Fairbanks -0.6 **San Francisco** 21 **Chicago** 12.4 **Los Angeles** 28.2 **Seattle** 15.8 **Honolulu** 32.4

1.8 Corrélation

```
In [33]: cor(mat_comb)
```

	Fairbanks	San Francisco	Chicago	Los Angeles	Seattle	Honolulu
Fairbanks	1.00000000	0.07356124	-0.6918586	-0.24325462	0.38624364	0.05025189
San Francisco	0.07356124	1.00000000	-0.3084798	-0.06947125	-0.49810768	0.00000000
Chicago	-0.69185856	-0.30847978	1.0000000	0.64005690	-0.48366537	0.51512220
Los Angeles	-0.24325462	-0.06947125	0.6400569	1.00000000	-0.04559608	0.14237370
Seattle	0.38624364	-0.49810768	-0.4836654	-0.04559608	1.00000000	-0.72057669
Honolulu	0.05025189	0.00000000	0.5151222	0.14237370	-0.72057669	1.00000000

1.9 summary

```
In [34]: summary(mat_comb)
```

```
   Fairbanks    San Francisco    Chicago      Los Angeles      Seattle
 Min.   :-3.0   Min.   :16    Min.   : 9.0   Min.   :26.0   Min.   :12.0
 1st Qu.:-1.0   1st Qu.:19    1st Qu.:13.0   1st Qu.:27.0   1st Qu.:16.0
 Median :-1.0   Median :22    Median :13.0   Median :28.0   Median :16.0
 Mean   :-0.6   Mean   :21    Mean   :12.4   Mean   :28.2   Mean   :15.8
 3rd Qu.: 0.0   3rd Qu.:22    3rd Qu.:13.0   3rd Qu.:29.0   3rd Qu.:17.0
 Max.   : 2.0   Max.   :26    Max.   :14.0   Max.   :31.0   Max.   :18.0
    Honolulu
 Min.   :32.0
 1st Qu.:32.0
 Median :32.0
 Mean   :32.4
 3rd Qu.:33.0
 Max.   :33.0
```

Si on veut par ligne, rappelons-nous que nous avons appris à transposer les matrices!

```
In [35]: summary(t(mat_comb))
```

```
      3/12            3/13            3/14            3/15
 Min.   :-1.00   Min.   : 0.00   Min.   :-1.00   Min.   :-3.00
 1st Qu.:14.00   1st Qu.:14.50   1st Qu.:12.25   1st Qu.:13.75
 Median :19.50   Median :16.00   Median :19.50   Median :17.50
 Mean   :19.00   Mean   :18.00   Mean   :18.50   Mean   :17.33
 3rd Qu.:28.75   3rd Qu.:25.75   3rd Qu.:27.50   3rd Qu.:25.00
 Max.   :32.00   Max.   :33.00   Max.   :33.00   Max.   :32.00
      3/16
 Min.   : 2.00
 1st Qu.:11.25
 Median :20.00
 Mean   :18.17
 3rd Qu.:25.00
 Max.   :32.00
```

2 Générer des variables aléatoires

2.1 runif

La fonction `runif` permet de générer des pseudo-variables aléatoires indépendantes entre deux bornes `runif(n=combien, min, max)`

```
In [40]: runif(1,0,10)
```

3.35023581283167

```
In [41]: runif(10,0,1)
```

1. 0.762836940586567 2. 0.604817938059568 3. 0.0636072147171944 4. 0.052814619615674 5. 0.429516698000953 6. 0.327015534741804 7. 0.425390120828524 8. 0.093060856917873 9. 0.203973314259201 10. 0.0420023950282484

On peut insérer maintenant les valeurs générées à l'intérieur d'un vecteur

```
In [44]: x<-runif(100,0,1)
         x
```

1. 0.539611001266167 2. 0.969200171763077 3. 0.161860105348751 4. 0.446521448437124 5. 0.0071074718143791 6. 0.963198963319883 7. 0.579066600417718 8. 0.165705290157348 9. 0.594905791571364 10. 0.375249444739893 11. 0.437035385984927 12. 0.98869122331962 13. 0.446837736526504 14. 0.547483163885772 15. 0.0244561785366386 16. 0.402985491789877 17. 0.349131921306252 18. 0.740115433465689 19. 0.277191531611606 20. 0.843555369414389 21. 0.85709911887534 22. 0.45878880051896 23. 0.0530683281831443 24. 0.92775225196965 25. 0.353458900935948 26. 0.568380535580218 27. 0.230667703552172 28. 0.337636061245576 29. 0.375225966563448 30. 0.387821002630517 31. 0.459963123081252 32. 0.272585999919102

33. 0.377333411248401 34. 0.998247936135158 35. 0.445947563275695 36. 0.815888306358829
37. 0.205926696537063 38. 0.145684423623607 39. 0.829736989922822 40. 0.863744982285425
41. 0.974668300244957 42. 0.588302413932979 43. 0.755799307022244 44. 0.54972411529161
45. 0.779031443409622 46. 0.0310876257717609 47. 0.412300328025594 48. 0.566727907396853
49. 0.498380966950208 50. 0.709585281088948 51. 0.915065908571705 52. 0.78087642788887
53. 0.178426813567057 54. 0.639498160919175 55. 0.439221660373732 56. 0.797969023464248
57. 0.70820996677503 58. 0.21974349534139 59. 0.38840561453253 60. 0.33431780571118
61. 0.125142666976899 62. 0.205675624776632 63. 0.305508880876005 64. 0.686442974256352
65. 0.864465568447486 66. 0.608728708233684 67. 0.288030886324123 68. 0.158349109115079
69. 0.323984490707517 70. 0.389953877544031 71. 0.180025292327628 72. 0.644314110744745
73. 0.414720708504319 74. 0.986791230970994 75. 0.340648488840088 76. 0.574771377490833
77. 0.933746692258865 78. 0.227020582649857 79. 0.105411230353639 80. 0.695686528459191
81. 0.632969576399773 82. 0.820697318995371 83. 0.438879428664222 84. 0.0309128267690539
85. 0.560534957563505 86. 0.079159309156239 87. 0.775976540520787 88. 0.0613414319232106
89. 0.803953293012455 90. 0.90822087507695 91. 0.127778963884339 92. 0.199427006300539
93. 0.604085020720959 94. 0.580892456928268 95. 0.463377881562337 96. 0.437255924800411
97. 0.885360276792198 98. 0.293950659455732 99. 0.168015816481784 100. 0.805729570332915

Ou les insérer à l'intérieur d'une matrice

```
In [45]: x<-matrix(x, 10)
         x
```

0.539611001	0.43703539	0.85709912	0.4599631	0.97466830	0.9150659	0.1251427	0.1800253	0.632
0.969200172	0.98869122	0.45878880	0.2725860	0.58830241	0.7808764	0.2056756	0.6443141	0.820
0.161860105	0.44683774	0.05306833	0.3773334	0.75579931	0.1784268	0.3055089	0.4147207	0.438
0.446521448	0.54748316	0.92775225	0.9982479	0.54972412	0.6394982	0.6864430	0.9867912	0.030
0.007107472	0.02445618	0.35345890	0.4459476	0.77903144	0.4392217	0.8644656	0.3406485	0.560
0.963198963	0.40298549	0.56838054	0.8158883	0.03108763	0.7979690	0.6087287	0.5747714	0.079
0.579066600	0.34913192	0.23066770	0.2059267	0.41230033	0.7082100	0.2880309	0.9337467	0.775
0.165705290	0.74011543	0.33763606	0.1456844	0.56672791	0.2197435	0.1583491	0.2270206	0.061
0.594905792	0.27719153	0.37522597	0.8297370	0.49838097	0.3884056	0.3239845	0.1054112	0.803
0.375249445	0.84355537	0.38782100	0.8637450	0.70958528	0.3343178	0.3899539	0.6956865	0.908

Nous avons maintenant obtenu une matrice de dimension 10X10

2.2 Seed

Comme dans SAS, nous avons appris comment générer les mêmes variables aléatoires dans un contexte de cumulation par exemple, avec la fonction seed

```
In [151]: set.seed(2)
```

```
In [152]: runif(2,0,1)
```

1. 0.18488225992769 2. 0.702374035958201

```
In [153]: runif(2,0,1)
```

1. 0.573326334822923 2. 0.168051920365542

```
In [154]: runif(2,0,1)
```

1. 0.943839338840917 2. 0.943474958650768

Si on remet le seed=2, on obtient alors les mêmes valeurs que nous avons obtenues auparavant

```
In [159]: set.seed(2)
```

```
In [160]: runif(2,0,1)
```

1. 0.18488225992769 2. 0.702374035958201

```
In [161]: runif(2,0,1)
```

1. 0.573326334822923 2. 0.168051920365542

```
In [162]: runif(2,0,1)
```

1. 0.943839338840917 2. 0.943474958650768

2.3 Sample (échantillonage)

La fonction `sample` tire un échantillon aléatoire de n variables à partir d'un ensemble de données allant de $\{1, \ldots, N\}$

```
In [168]: sample(1:10,5)
```

1. 2 2. 8 3. 7 4. 4 5. 9

```
In [169]: sample(x,5)
```

1. 0.467615901259705 2. 0.585986070567742 3. 0.318405627040192 4. 0.117259352467954 5. 0.808955513406545

```
In [170]: sample(x,5)
```

1. 0.0845668376423419 2. 0.693119892384857 3. 0.849038900341839 4. 0.410260857315734 5. 0.884574939031154

$n \leq N$

```
In [172]: sample(x,length(x)+1)
```

```
    Error in sample.int(length(x), size, replace, prob): cannot take a sample larger than th
 Traceback:


    1. sample(x, length(x) + 1)

    2. sample.int(length(x), size, replace, prob)
```

Lorsque notre échantillon tiré est plus grand que notre ensemble de données, on peut utiliser un tirage avec remise:

```
In [175]: sample(0:1, 100, replace = T)
```

1. 1 2. 0 3. 0 4. 0 5. 0 6. 0 7. 1 8. 0 9. 1 10. 0 11. 1 12. 0 13. 0 14. 0 15. 1 16. 1 17. 0 18. 1 19. 0 20. 1 21. 0 22. 1 23. 0 24. 1 25. 1 26. 0 27. 0 28. 1 29. 0 30. 1 31. 0 32. 0 33. 1 34. 1 35. 1 36. 1 37. 1 38. 1 39. 0 40. 1 41. 0 42. 1 43. 0 44. 0 45. 0 46. 0 47. 1 48. 0 49. 1 50. 0 51. 0 52. 0 53. 0 54. 0 55. 1 56. 1 57. 1 58. 1 59. 0 60. 1 61. 0 62. 0 63. 0 64. 0 65. 1 66. 1 67. 1 68. 1 69. 1 70. 0 71. 0 72. 0 73. 0 74. 0 75. 0 76. 1 77. 0 78. 0 79. 1 80. 1 81. 1 82. 0 83. 1 84. 1 85. 1 86. 0 87. 1 88. 1 89. 1 90. 0 91. 1 92. 0 93. 0 94. 0 95. 1 96. 0 97. 1 98. 1 99. 1 100. 1

Cette fonction fonctionne aussi sur un tirage de variable de type caractères

```
In [176]: sample(state.name, 5)
```

1. 'Kentucky' 2. 'Montana' 3. 'Delaware' 4. 'Wyoming' 5. 'North Carolina'

La fonction `sample` donne des probabilités égales à tous les éléments tirés d'un ensemble de données. Toutesfois, il est possible de préciser la probabilité de chaque élément tiré.

```
In [1]: s<-sample(1:5, 1000, replace=T, prob=c(.2,.2,.2,.2,.2))
        s
```

1. 5 2. 4 3. 4 4. 5 5. 1 6. 1 7. 1 8. 1 9. 5 10. 1 11. 1 12. 1 13. 1 14. 3 15. 5 16. 2 17. 5 18. 4 19. 1 20. 3 21. 3 22. 5 23. 4 24. 2 25. 3 26. 4 27. 4 28. 5 29. 2 30. 4 31. 5 32. 2 33. 4 34. 3 35. 1 36. 5 37. 5 38. 3 39. 5 40. 5 41. 3 42. 5 43. 1 44. 5 45. 1 46. 4 47. 1 48. 5 49. 5 50. 2 51. 4 52. 4 53. 5 54. 2 55. 5 56. 5 57. 4 58. 3 59. 4 60. 5 61. 2 62. 1 63. 3 64. 1 65. 3 66. 2 67. 3 68. 4 69. 5 70. 2 71. 5 72. 4 73. 4 74. 4 75. 4 76. 2 77. 5 78. 3 79. 2 80. 2 81. 1 82. 2 83. 4 84. 1 85. 4 86. 2 87. 1 88. 1 89. 4 90. 2 91. 1 92. 3 93. 5 94. 1 95. 4 96. 4 97. 5 98. 2 99. 2 100. 1 101. 1 102. 4 103. 1 104. 5 105. 4 106. 2 107. 5 108. 2 109. 5 110. 1 111. 4 112. 4 113. 1 114. 2 115. 3 116. 4 117. 4 118. 5 119. 2 120. 3 121. 5 122. 3 123. 5 124. 3 125. 3 126. 3 127. 5 128. 2 129. 2 130. 5 131. 1 132. 2 133. 5 134. 2 135. 4 136. 3 137. 2 138. 3 139. 4 140. 5 141. 3 142. 1 143. 2 144. 2 145. 3 146. 1 147. 3 148. 1 149. 5 150. 4 151. 4 152. 1 153. 2 154. 4 155. 4 156. 2 157. 1 158. 3 159. 1 160. 4 161. 1 162. 2 163. 5 164. 4 165. 3 166. 3 167. 1 168. 4 169. 2 170. 5 171. 3 172. 1 173. 5 174. 1 175. 5 176. 4 177. 5 178. 5 179. 4 180. 1 181. 1 182. 2 183. 2 184. 1 185. 3 186. 3 187. 3 188. 1 189. 3 190. 5 191. 2 192. 2 193. 4 194. 3 195. 4 196. 5 197. 4 198. 3 199. 5 200. 5 201. 3 202. 2 203. 5 204. 5 205. 2 206. 2 207. 2 208. 1 209. 2 210. 2 211. 4 212. 3 213. 2 214. 2 215. 4 216. 1 217. 1 218. 1 219. 3 220. 1 221. 4 222. 3 223. 4 224. 1 225. 3 226. 1 227. 2 228. 3 229. 4 230. 4 231. 4 232. 5 233. 2 234. 1 235. 1 236. 5 237. 1 238. 3 239. 5 240. 4 241. 4 242. 3 243. 4 244. 3 245. 2 246. 5 247. 2 248. 2 249. 2 250. 3 251. 4 252. 1 253. 1 254. 1 255. 3 256. 4 257. 3 258. 1 259. 3 260. 1 261. 1 262. 3 263. 1 264. 5 265. 3 266. 1 267. 2 268. 5 269. 5 270. 3 271. 5 272. 3 273. 5 274. 5 275. 3 276. 3 277. 5 278. 5 279. 1 280. 2 281. 4 282. 1 283. 2 284. 4 285. 1 286. 4 287. 3 288. 2 289. 3 290. 1 291. 2 292. 3 293. 1 294. 3 295. 3 296. 3 297. 1 298. 3 299. 5 300. 3 301. 5 302. 2 303. 2 304. 5 305. 5 306. 5 307. 1 308. 1 309. 4 310. 1 311. 2 312. 3 313. 5 314. 4 315. 4 316. 5 317. 4 318. 2 319. 2 320. 2 321. 1 322. 1 323. 2 324. 2 325. 3 326. 1 327. 2 328. 5 329. 3 330. 3 331. 4 332. 4 333. 5 334. 3 335. 5 336. 3 337. 3 338. 3 339. 2 340. 2 341. 2 342. 2 343. 4 344. 3 345. 1 346. 4 347. 2 348. 3 349. 2 350. 3 351. 5 352. 5 353. 5 354. 3 355. 2 356. 5 357. 4 358. 3 359. 1 360. 3 361. 2 362. 2 363. 1 364. 5 365. 2 366. 3 367. 3 368. 1 369. 1 370. 2 371. 4 372. 5 373. 3 374. 4 375. 5 376. 5 377. 2 378. 1 379. 3 380. 2 381. 5 382. 4 383. 4 384. 5 385. 3 386. 3 387. 2 388. 4 389. 3 390. 3 391. 2 392. 1 393. 1 394. 5 395. 2 396. 2 397. 1 398. 5 399. 2 400. 2 401. 5 402. 4 403. 4 404. 3 405. 5 406. 4 407. 2 408. 5 409. 2 410. 1 411. 3 412. 2 413. 2 414. 3 415. 1 416. 3 417. 1 418. 3 419. 2 420. 1 421. 2 422. 5 423. 1 424. 4 425. 4 426. 4 427. 1 428. 2 429. 1 430. 1 431. 5 432. 4 433. 4 434. 1 435. 3 436. 3 437. 1 438. 3 439. 3 440. 2 441. 3 442. 2 443. 2 444. 2 445. 4 446. 2 447. 4 448. 4 449. 1 450. 3 451. 1 452. 2 453. 4 454. 5 455. 4 456. 5 457. 5 458. 2 459. 3 460. 3 461. 4 462. 3 463. 4 464. 1 465. 2 466. 2 467. 1 468. 2 469. 5 470. 4 471. 5 472. 5 473. 5 474. 4 475. 4

476. 4 477. 1 478. 4 479. 5 480. 3 481. 1 482. 2 483. 1 484. 1 485. 1 486. 4 487. 3 488. 4 489. 1 490. 4
491. 5 492. 3 493. 5 494. 5 495. 1 496. 4 497. 5 498. 5 499. 5 500. 3 501. 5 502. 5 503. 5 504. 2 505. 2
506. 1 507. 3 508. 3 509. 1 510. 4 511. 4 512. 1 513. 3 514. 5 515. 2 516. 5 517. 5 518. 3 519. 2 520. 2
521. 4 522. 3 523. 2 524. 4 525. 2 526. 1 527. 2 528. 2 529. 1 530. 2 531. 1 532. 3 533. 1 534. 1 535. 5
536. 1 537. 5 538. 2 539. 1 540. 4 541. 5 542. 1 543. 5 544. 3 545. 1 546. 3 547. 2 548. 5 549. 2 550. 4
551. 1 552. 4 553. 4 554. 1 555. 1 556. 5 557. 3 558. 4 559. 5 560. 3 561. 2 562. 1 563. 3 564. 4 565. 1
566. 3 567. 4 568. 1 569. 5 570. 3 571. 4 572. 1 573. 3 574. 1 575. 3 576. 1 577. 3 578. 1 579. 4 580. 4
581. 4 582. 1 583. 5 584. 4 585. 5 586. 1 587. 4 588. 2 589. 2 590. 5 591. 5 592. 3 593. 1 594. 2 595. 3
596. 5 597. 5 598. 4 599. 3 600. 3 601. 2 602. 3 603. 2 604. 1 605. 1 606. 4 607. 2 608. 1 609. 2 610. 5
611. 5 612. 2 613. 5 614. 3 615. 5 616. 3 617. 3 618. 5 619. 1 620. 1 621. 5 622. 3 623. 4 624. 4 625. 2
626. 4 627. 5 628. 5 629. 4 630. 3 631. 3 632. 2 633. 3 634. 4 635. 5 636. 1 637. 2 638. 3 639. 5 640. 1
641. 2 642. 1 643. 2 644. 2 645. 4 646. 4 647. 1 648. 5 649. 3 650. 2 651. 5 652. 2 653. 5 654. 3 655. 5
656. 5 657. 1 658. 4 659. 5 660. 1 661. 5 662. 3 663. 1 664. 2 665. 5 666. 1 667. 5 668. 5 669. 5 670. 4
671. 4 672. 3 673. 1 674. 1 675. 2 676. 3 677. 2 678. 2 679. 5 680. 5 681. 4 682. 5 683. 1 684. 2 685. 1
686. 4 687. 4 688. 1 689. 1 690. 1 691. 4 692. 4 693. 2 694. 2 695. 2 696. 1 697. 3 698. 5 699. 2 700. 2
701. 2 702. 4 703. 3 704. 3 705. 1 706. 3 707. 1 708. 5 709. 3 710. 3 711. 5 712. 4 713. 5 714. 1 715. 1
716. 1 717. 4 718. 4 719. 3 720. 4 721. 5 722. 3 723. 5 724. 3 725. 2 726. 3 727. 2 728. 3 729. 1 730. 4
731. 2 732. 2 733. 3 734. 5 735. 4 736. 5 737. 2 738. 5 739. 5 740. 1 741. 4 742. 3 743. 4 744. 3 745. 1
746. 3 747. 1 748. 4 749. 4 750. 5 751. 5 752. 1 753. 4 754. 3 755. 1 756. 4 757. 2 758. 1 759. 2 760. 4
761. 3 762. 5 763. 2 764. 5 765. 5 766. 5 767. 3 768. 5 769. 2 770. 3 771. 4 772. 1 773. 4 774. 1 775. 5
776. 1 777. 3 778. 2 779. 5 780. 1 781. 3 782. 4 783. 4 784. 2 785. 5 786. 1 787. 3 788. 4 789. 3 790. 2
791. 3 792. 5 793. 4 794. 2 795. 5 796. 3 797. 3 798. 5 799. 3 800. 1 801. 2 802. 1 803. 3 804. 1 805. 1
806. 1 807. 1 808. 5 809. 1 810. 3 811. 1 812. 5 813. 2 814. 1 815. 1 816. 2 817. 2 818. 2 819. 3 820. 2
821. 2 822. 3 823. 2 824. 5 825. 4 826. 4 827. 4 828. 1 829. 3 830. 4 831. 1 832. 3 833. 1 834. 2 835. 4
836. 2 837. 1 838. 5 839. 3 840. 2 841. 3 842. 2 843. 5 844. 4 845. 2 846. 2 847. 4 848. 1 849. 3 850. 4
851. 1 852. 2 853. 5 854. 3 855. 3 856. 5 857. 1 858. 5 859. 2 860. 2 861. 5 862. 3 863. 2 864. 1 865. 1
866. 3 867. 1 868. 3 869. 1 870. 1 871. 4 872. 3 873. 2 874. 4 875. 5 876. 5 877. 4 878. 1 879. 5 880. 5
881. 3 882. 2 883. 4 884. 4 885. 5 886. 5 887. 1 888. 4 889. 5 890. 1 891. 3 892. 5 893. 1 894. 2 895. 2
896. 2 897. 1 898. 1 899. 3 900. 1 901. 5 902. 3 903. 1 904. 2 905. 4 906. 5 907. 5 908. 1 909. 1 910. 1
911. 2 912. 1 913. 1 914. 5 915. 3 916. 5 917. 3 918. 4 919. 4 920. 5 921. 4 922. 3 923. 5 924. 1 925. 3
926. 2 927. 1 928. 3 929. 5 930. 3 931. 1 932. 4 933. 1 934. 4 935. 5 936. 4 937. 3 938. 1 939. 3 940. 4
941. 3 942. 3 943. 5 944. 5 945. 3 946. 3 947. 2 948. 4 949. 5 950. 5 951. 1 952. 4 953. 5 954. 5 955. 4
956. 4 957. 3 958. 1 959. 3 960. 4 961. 1 962. 3 963. 1 964. 3 965. 1 966. 4 967. 1 968. 1 969. 1 970. 5
971. 5 972. 2 973. 4 974. 3 975. 3 976. 2 977. 1 978. 1 979. 1 980. 4 981. 5 982. 4 983. 2 984. 1 985. 4
986. 1 987. 1 988. 5 989. 2 990. 3 991. 3 992. 1 993. 3 994. 4 995. 3 996. 2 997. 1 998. 1 999. 5 1000. 1

Si on utilise la fonction `table` afin de compter l'occurrence de chaque élément

```
In [7]: tableau <- table(s)
        tableau
```

```
s
  1   2   3   4   5
222 187 203 183 205
```

on remarque que chaque élément à été tiré à un taux d'environ 20%

```
In [18]: tableau[[1]]/sum(tableau)
```

0.222

Si on change les probabilités maintenant, mais **attention** la somme des probabilités doit être égale à 1

```
In [188]: s<-sample(1:5, 1000, replace=T, prob=c(.2,.5,.1,.1,.1))
          table(s)
```

```
s
  1   2   3   4   5
197 509 101  87 106
```

2.4 rnorm

La moyenne par défaut est égale à 0 et l'écart-type=1

```
In [190]: rnorm(1)
```

0.106821263122198

```
In [191]: rnorm(1, 0, 100)
```

-24.9130626933775

Créons un vecteur de 100 valeurs avec moyenne=100 et écart-type=100

```
In [192]: x<-rnorm(100, 0, 100)
```

Si on calcule la moyenne de ce vecteur;

```
In [193]: mean(x)
```

-5.83207843041791

Nous avons obtenu une moyenne proche de la moyenne de nos variables aléatoires générées par la fonction `rnorm`

La même chose maintenant pour l'écart-type

```
In [195]: sd(x)
```

93.1100361171955

Toutefois, si nous augmentons le nombre de variables aléatoires généré, nous sommes alors plus proches des arguments de la fonction `rnorm` que nous avons saisie

```
In [197]: x<-rnorm(10000, 0, 100)
          mean(x)
```

0.963181514782263

```
In [198]: sd(x)
```

101.228973820731

3 arrays

Les tableaux sont une généralisation des matrices. Le nombre de dimensions d'un tableau est égal à la longueur de l'attribut `dim`. Sa classe est "array"

```
In [22]: x<-array(1:24, dim=c(3,4,2))
```

```
In [23]: x
```

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

Les tableaux sont un cas spécial des matrices. Ils sont comme des vecteurs ou des matrices, à l'exception d'avoir des attributs additionnels.

4 Listes

Les listes sont un type de vecteur spécial qui peut être composé d'élément ayant n'importe laquelle classe (numérique, string ou booléen).

```
In [41]: (x <- list(taille = c(1, 5, 2), utilisateur = "Mike", new = TRUE))
```

$taille 1. 1 2. 5 3. 2

$utilisateur 'Mike'

$new TRUE

Puisque la liste un vecteur, on peut alors extraire avec les crochets `[]`

```
In [42]: x[1]
```

$taille = 1. 1 2. 5 3. 2

```
In [43]: x[[1]]
```

1. 1 2. 5 3. 2

```
In [44]: x$taille
```

1. 1 2. 5 3. 2

```
In [46]: x$utilisateur
```

'Mike'

5 Data Frames

Un *Data Frame* est une **liste** de vecteurs de même longueur. Conceptuellement, c'est une matrice dont les lignes correspondent aux variables explicatives, et les lignes sont les valeurs mesurées de ces variables.

```
In [46]: villes <-c("Montréal", "Québec", "Laval")
         Population <-c(1942044, 585485, 430077)
         village <-c(F,T,T)
```

```
In [47]: donnees_ville <-data.frame(villes, Population, village)
```

```
In [48]: donnees_ville
```

villes	Population	village
Montréal	1942044	FALSE
Québec	585485	TRUE
Laval	430077	TRUE

Si l'on vérifie les attributs de ce *data frame*

```
In [49]: attributes(donnees_ville)
```

$names 1. 'villes' 2. 'Population' 3. 'village'

$row.names 1. 1 2. 2 3. 3

$class 'data.frame'

ça nous donne les étiquettes des colonnes, le nom de colonnes (numéros) et la classe

```
In [51]: donnees_ville[,1]
```

1. Montréal 2. Québec 3. Laval

```
In [52]: donnees_ville[,2]
```

1. 1942044 2. 585485 3. 430077

```
In [53]: donnees_ville[2,1]
```

Québec
Ou par nom;

```
In [54]: donnees_ville$Population
```

1. 1942044 2. 585485 3. 430077

Remarquez que les villes ne s'affichent pas entre guillemets comme des strings, mais plutôt comme des `levels`, Si l'on voulait les avoir en strings, il faut ajouter l'argument `stringAsFactors=F`

```
In [50]: donnees_ville <-data.frame(villes, Population, village, stringsAsFactors=F)
```

```
In [51]: donnees_ville
```

villes	Population	village
Montréal	1942044	FALSE
Québec	585485	TRUE
Laval	430077	TRUE

```
In [52]: donnees_ville$villes
```

1. 'Montréal' 2. 'Québec' 3. 'Laval'

Une fonction très utile afin d'avoir un résumé sur les éléments du `df`

```
In [53]: str(donnees_ville)
```

```
'data.frame':        3 obs. of  3 variables:
 $ villes    : chr  "Montréal" "Québec" "Laval"
 $ Population: num  1942044 585485 430077
 $ village   : logi  FALSE TRUE TRUE
```

```
In [54]: df<-donnees_ville
```

```
In [55]: summary(df)
```

```
    villes            Population       village
 Length:3           Min.   : 430077   Mode :logical
 Class :character   1st Qu.: 507781   FALSE:1
 Mode  :character   Median : 585485   TRUE :2
                    Mean   : 985869   NA's :0
                    3rd Qu.:1263764
                    Max.   :1942044
```

On se rappelle des données `Cars93`, ce sont des données sous format df. Chargeons-les afin de travailler avec quelques exemples.

```
In [20]: require(MASS)
```

```
In [21]: data(Cars93)
```

```
In [66]: class(Cars93)
```

'data.frame'

```
In [67]: str(Cars93)
```

```
'data.frame':        93 obs. of  27 variables:
 $ Manufacturer      : Factor w/ 32 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5 ...
 $ Model             : Factor w/ 93 levels "100","190E","240",..: 49 56 9 1 6 24 54 74 73 35 ...
 $ Type              : Factor w/ 6 levels "Compact","Large",..: 4 3 1 3 3 3 2 2 3 2 ...
 $ Min.Price         : num  12.9 29.2 25.9 30.8 23.7 14.2 19.9 22.6 26.3 33 ...
```

```
$ Price             : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
$ Max.Price         : num  18.8 38.7 32.3 44.6 36.2 17.3 21.7 24.9 26.3 36.3 ...
$ MPG.city          : int  25 18 20 19 22 22 19 16 19 16 ...
$ MPG.highway       : int  31 25 26 26 30 31 28 25 27 25 ...
$ AirBags           : Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2 ...
$ DriveTrain        : Factor w/ 3 levels "4WD","Front",..: 2 2 2 2 3 2 2 3 2 2 ...
$ Cylinders         : Factor w/ 6 levels "3","4","5","6",..: 2 4 4 4 2 2 4 4 4 5 ...
$ EngineSize        : num  1.8 3.2 2.8 2.8 3.5 2.2 3.8 5.7 3.8 4.9 ...
$ Horsepower        : int  140 200 172 172 208 110 170 180 170 200 ...
$ RPM               : int  6300 5500 5500 5500 5700 5200 4800 4000 4800 4100 ...
$ Rev.per.mile      : int  2890 2335 2280 2535 2545 2565 1570 1320 1690 1510 ...
$ Man.trans.avail   : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 1 1 1 1 ...
$ Fuel.tank.capacity: num  13.2 18 16.9 21.1 21.1 16.4 18 23 18.8 18 ...
$ Passengers        : int  5 5 5 6 4 6 6 6 5 6 ...
$ Length            : int  177 195 180 193 186 189 200 216 198 206 ...
$ Wheelbase         : int  102 115 102 106 109 105 111 116 108 114 ...
$ Width             : int  68 71 67 70 69 69 74 78 73 73 ...
$ Turn.circle       : int  37 38 37 37 39 41 42 45 41 43 ...
$ Rear.seat.room    : num  26.5 30 28 31 27 28 30.5 30.5 26.5 35 ...
$ Luggage.room      : int  11 15 14 17 13 16 17 21 14 18 ...
$ Weight            : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
$ Origin            : Factor w/ 2 levels "USA","non-USA": 2 2 2 2 2 1 1 1 1 1 ...
$ Make              : Factor w/ 93 levels "Acura Integra",..: 1 2 4 3 5 6 7 9 8 10 ...
```

On voit que nous avons 93 observations avec 27 variables

2_4_Solutions

October 15, 2018

Liste des numéros

1 Question

Créer une matrice A au format 2X4 avec les valeurs suivantes: 1 2 3 4 5 6 7 8

```
In [93]: 1:8
```

1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8

```
In [101]: A<- matrix(c(1:8), nrow = 2, ncol = 4)
          A
```

1	3	5	7
2	4	6	8

Recréez cette matrice, mais cette fois les valeurs sont incrémentées par lignes

```
In [103]: B<- matrix(c(1:8), nrow = 2, ncol = 4, byrow = T)
          B
```

1	2	3	4
5	6	7	8

2 Question

Créez une matrice carrée avec les valeurs du vecteur vec3 créée auparavant:

```
In [108]: matrix(vec3, nrow = 5)
```

2	12	22	32	42
4	14	24	34	44
6	16	26	36	46
8	18	28	38	48
10	20	30	40	50

3 Question

Créer une matrice identité 7X7

```
In [30]: mat<-diag(7)
         mat
```

1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	1	0
0	0	0	0	0	0	1

Afin des fins de calcul de réserves actuarielles, vous devez renverser (*reverse*) la diagonale de sorte où les 1 sont dans les cases (7,1), (6,2),...,(1,7)

```
In [31]: apply(diag(7), 2, rev)
```

0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
1	0	0	0	0	0	0

3.1 autre façon de faire

```
In [32]: mat <- mat[ nrow(mat):1, ]
         mat
```

0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
1	0	0	0	0	0	0

4 Question

À partir des données suivantes, créez une matrice où vous avez la population des villes (Montréal, Québec, Laval, Gatineau) par ligne et les années 2013 à 2016 par colonnes

```
In [33]: pop<-matrix(c(1718241,1735096,1746940,1767753,
                       530474,533857,536013,538918,
                       417325,421959,425481,429413,
                       274180,276290,278050,281392), ncol=4, byrow = T)
```

```
In [34]: pop
```

1718241	1735096	1746940	1767753
530474	533857	536013	538918
417325	421959	425481	429413
274180	276290	278050	281392

```
In [35]: rownames(pop)<-c("Montréal", "Québec", "Laval", "Gatineau")
         colnames(pop)<-2013:2016
```

```
In [36]: pop
```

	2013	2014	2015	2016
Montréal	1718241	1735096	1746940	1767753
Québec	530474	533857	536013	538918
Laval	417325	421959	425481	429413
Gatineau	274180	276290	278050	281392

On vous dit que l'arrondissement Hochelaga-Maisonneuve (situé à Montréal) est devenu un quartier très aisé et veut maintenant avoir son indépendance. Aujourd'hui ce prestigieux quartier appelé HOMA, l'évolution de la population de ce quartier de 2013 à 2016 a été la suivante: 20000, 20500, 23000, 23800

Quelle aurait été la population de Montréal sans compter les habitants du pays très prospère pays HOMA

```
In [37]: vec_HOMA<-c(20000, 20500, 23000, 23800)
         HOMA<-matrix(vec_HOMA, ncol=4, )
         rownames(HOMA)<-c("HOMA")
         colnames(HOMA)<-2013:2016
         HOMA
```

	2013	2014	2015	2016
HOMA	20000	20500	23000	23800

```
In [38]: pop[1,]-vec_HOMA
```

2013 1698241 **2014** 1714596 **2015** 1723940 **2016** 1743953

Reconstruisez la nouvelle matrice avec les nouvelles données de Montréal.

```
In [39]: pop[1,]<-pop[1,]-vec_HOMA
         pop
```

	2013	2014	2015	2016
Montréal	1698241	1714596	1723940	1743953
Québec	530474	533857	536013	538918
Laval	417325	421959	425481	429413
Gatineau	274180	276290	278050	281392

Ajouter les données de HOMA à la matrice modifiée !

```
In [173]: rbind(pop,HOMA)
```

	2013	2014	2015	2016
Montréal	1698241	1714596	1723940	1743953
Québec	530474	533857	536013	538918
Laval	417325	421959	425481	429413
Gatineau	274180	276290	278050	281392
HOMA	20000	20500	23000	23800

5 Question

Créer deux vecteurs aléatoires nommés « x1 » et « x2 », contenant chacun 100 valeurs aléatoires compatibles 1. avec une distribution de loi normale centrée réduite et 2. avec une distribution de loi uniforme définie sur l'intervalle [0 ; 10].

```
In [42]: x1<-rnorm(100)
         x2<-runif(100, 0,10)
```

Créez une matrice 10X10 contient les valeur du vecteur x1 crée auparavant:

```
In [43]: matNorm<-matrix(x1,nrow=10)
```

Calculez la moyenne de cette dernière et la variance de cette dernière

```
In [44]: mean(matNorm)
```

0.0178254903830712

```
In [45]: (sd(matNorm))**2
```

0.823977976790721

6 Question

Créer un vecteur xx1 contenant un échantillon équiprobable de 4 variables à partir du vecteur x1 de la question précédente

```
In [46]: xx1<-sample(x1, 4)
         xx1
```

1. -0.163930968642975 2. 0.300279118120242 3. 0.596425059015276 4. -0.488922835294287

À partir du vecteur xx1, créez un autre vecteur xx2 qui possède 1000 variables de l'échantillon xx1. La dernière variable possède une probabilité de 70% qu'elle soit tirée alors que les trois premières ont chacune 10% de chance qu'elle soit tirée.

```
In [47]: xx2<-sample(xx1, 1000, replace = T, prob=c(.1,.1,.1,.7))
```

Donnez la fréquence de chacune des variables simulée

```
In [48]: table(xx2)
```

```
xx2
-0.488922835294287 -0.163930968642975  0.300279118120242  0.596425059015276
               705                 82                111                102
```

7 Question

On vous dit que les temps pour finir un demi-triathlon suivent une loi normale avec une moyenne (et les écarts types [ET]) pour les hommes et les femmes sont les suivantes: * Pour les **hommes** nager 1.9 km en 40 minutes (ET=3), pédaler 90 km en 2:45 (ET=8), et courir 21.1 km en 2:05 (ET=10). * Pour les **Femmes** nager 1.9 km en 50 minutes (ET=5), pédaler 90 km en 3:00 (ET=5), et courir 21.1 km en 2:15 (ET=12).

Créer les vecteurs {`swimH`, `bikeH`, `runH`, `swimF`, `bikeF`, `runF`} contenant le temps pour chacun des sports pour 1002 hommes et 1300 femmes, tirés aléatoirement selon les lois ci-dessus (on suppose que les trois sports sont indépendants même si en réalité ce n'est jamais vrai, car si on se blesse en vélo, on performe beaucoup moins en course).

```
In [49]: set.seed(123)
         n<-1002
         swimH<-round(rnorm(n, mean = 40, sd = 3),2)
         bikeH<-round(rnorm(n, mean = 165, sd = 8),2)
         runH<-round(rnorm(n, mean = 125, sd = 8),2)

         m<-1300
         swimF<-round(rnorm(m, mean = 50, sd = 5),2)
         bikeF<-round(rnorm(m, mean = 180, sd = 5),2)
         runF<-round(rnorm(m, mean = 135, sd = 12),2)
```

Avec les vecteurs crée précedemment, construisez une matrice pour les hommes et une autre pour les femmes

```
In [50]: resultatH<-matrix(c(swimH, bikeH, runH), ncol = 3)
         colnames(resultatH)<-c("Swim", "Bike", "Run")
         rownames(resultatH)<-paste("H",1:n,sep='')
```

```
In [51]: head(resultatH)
```

	Swim	Bike	Run
H1	38.32	164.86	126.39
H2	39.31	163.94	120.08
H3	44.68	144.61	110.54
H4	40.21	173.32	119.85
H5	40.39	167.00	141.37
H6	45.15	184.33	120.51

```
In [52]: tail(resultatH)
```

	Swim	Bike	Run
H997	43.21	169.29	123.80
H998	35.95	161.32	122.38
H999	38.43	160.91	113.41
H1000	39.25	166.90	119.42
H1001	37.01	160.67	145.79
H1002	36.88	174.75	124.70

```
In [53]: resultatF<-matrix(c(swimF, bikeF, runF), ncol = 3)
         colnames(resultatF)<-c("Swim", "Bike", "Run")
         rownames(resultatF)<-paste("F",n+1:m,sep='')
```

```
In [54]: head(resultatF)
```

	Swim	Bike	Run
F1003	54.57	180.26	117.59
F1004	49.08	184.04	130.86
F1005	53.05	175.30	115.72
F1006	49.74	180.20	149.05
F1007	56.82	170.01	131.69
F1008	47.48	180.69	132.66

```
In [55]: tail(resultatF)
```

	Swim	Bike	Run
F2297	57.17	172.19	114.79
F2298	54.56	174.28	132.75
F2299	51.91	176.39	132.54
F2300	52.76	182.63	126.28
F2301	50.72	168.28	124.29
F2302	58.54	179.21	134.37

Créez une matrice qui contient les résultats des femmes ensuite et le résultat des hommes

```
In [292]: résultat<-rbind(resultatH, resultatF)
```

```
In [293]: head(résultat)
```

	Swim	Bike	Run
H1	40.54	171.39	135.25
H2	41.78	157.40	130.89
H3	46.27	168.04	131.96
H4	39.01	157.34	133.89
H5	36.20	160.56	118.65
H6	42.51	163.69	123.67

```
In [294]: tail(résultat)
```

	Swim	Bike	Run
F2297	57.86	176.24	173.19
F2298	48.88	177.83	111.61
F2299	43.47	175.25	137.77
F2300	46.19	180.49	156.81
F2301	51.36	183.77	129.70
F2302	59.24	182.38	134.04

Quel est le numéro du dossard du participant/es qui a le meilleur temps en nage, et en combien de temps à accomplie cette discipline

```
In [273]: min(résultat[, 1])
```

31.34

```
In [272]: which(résultat[, 1]==min(résultat[, 1]))
```

H19: 19

Quel est le numéro du dossard du participant/es qui a le meilleur temps en vélo, et en combien de temps à accomplie cette discipline

```
In [271]: which(résultat[, 2]==min(résultat[, 2]))
```

H759: 759

```
In [275]: min(résultat[, 2])
```

142.31

Quel est le numéro du dossard du participant/es qui a le meilleur temps en course, et en combien de temps à accomplie cette discipline

```
In [276]: which(résultat[, 3]==min(résultat[, 3]))
```

F1719: 1719

```
In [277]: min(résultat[, 3])
```

102.02

Quel a été le meilleur temps chez les femmes?

```
In [298]: tempsF<-resultatF[, 1]+resultatF[, 2]+resultatF[, 3]
```

```
In [301]: bestF<-min(tempsF)
          bestF
```

322.77
Quel numéro de dossard?

```
In [302]: which(tempsF==min(tempsF))
```

F1871: 869
Qui a gagné la course et en combien de temps?

```
In [306]: temps<-résultat[, 1]+résultat[, 2]+résultat[, 3]
          gagnant<-min(temps)
          gagnant
          which(temps==min(temps))
```

292.15
H249: 249

8 Question

Créez un vecteur appelé *ann* de qui représente les années de développement dans calcul d'annuité de 5 ans, qui donne le résultat suivant $\{1,\ldots,5\}$

```
In [42]: ann<-1:5
         ann
```

1. 1 2. 2 3. 3 4. 4 5. 5

Créer un vecteur contenant les fameux facteurs d'actualisation v^n qui servent à calculer la valeur présente d'une série de paiements $n = 5$ avec un taux d'intérêt de 2.5%

$$v^n = \frac{1}{1+i} \tag{1}$$

```
In [44]: i<-.025
         v_n<-(1+i)**(-ann)
         v_n
```

1. 0.975609756097561 2. 0.951814396192743 3. 0.928599410919749 4. 0.905950644799755 5. 0.883854287609517

Calculer la valeur présente d'une annuité 5 ans avec qui 153.25$ par année

$$\begin{aligned} PV &= a_n \\ &= v + v^2 +, \ldots, v^n \\ &= \sum_{j=1}^{n} v^j \end{aligned} \tag{2}$$

```
In [46]: pmt<-153.25
         sum(pmt*v_n)
```

711.973216953662

Reproduiser votre calcul avec la fonction suivante:

$$\begin{aligned} PV &= a_n \\ &= v + v^2 +, \ldots, v^n \\ &= \sum_{j=1}^{n} v^j \\ &= \frac{1 - v^n}{i} \end{aligned} \tag{3}$$

Lorsque le taux d'intérêt est constant d'une année à l'autre

```
In [ ]: n<-5
        i<-.025
```

```
In [60]: vn<-(1+i)**-n
         vn
```

0.883854287609517

```
In [63]: PV<-(1-vn)/i
         PV
```

4.64582849561931

```
In [64]: pmt*PV
```

711.973216953659

9 Question

on se rappelle du taux *Effective rate of discount*

$$d_t = \frac{a(t) - a(t-1)}{a(t-1)} \tag{4}$$

Le taux *discount* se calcule avec la fonction suivante:

$$d = \frac{i}{1+i} = iv \tag{5}$$

Soit un taux d'intérêt de 5%, quel sera alors de taux de *discount* avec seulement 6 décimales

```
In [70]: i<-.05
         d<-i/(1+i)
         round(d,6)
```

0.047619

10 Question

Écrivez un code R pour créer la liste suivante :

```
In [349]: (x <- list(ssd = c(256, 128, 512), machine = "Macbook Pro", best = TRUE))
```

$ssd 1. 256 2. 128 3. 512

$machine 'Macbook Pro'

$best TRUE

Ecrivez un code qui extrait les différente tailles du ssd **seulement**

```
In [350]: x[[1]]
```

1. 256 2. 128 3. 512

```
In [351]: x$ssd
```

1. 256 2. 128 3. 512
Extraire les étiquettes de la liste;

```
In [352]: names(x)
```

1. 'ssd' 2. 'machine' 3. 'best'
Extraire le 3e élément du premier élément de liste:

```
In [353]: x[[1]][3]
```

512
Remplacer le dernier élément par le vecteur `T,F,T`

```
In [354]: x[[3]]<-c(T,F,T)
```

```
In [355]: x
```

$ssd 1. 256 2. 128 3. 512

$machine 'Macbook Pro'

$best 1. TRUE 2. FALSE 3. TRUE

11 Question

Soit le vecteur suivant:

```
In [362]: x<-c(71,18,86,5,58,19,14,9,74,75,59,24,7,51,50,63,35,53,72,61)
          x
```

1. 71 2. 18 3. 86 4. 5 5. 58 6. 19 7. 14 8. 9 9. 74 10. 75 11. 59 12. 24 13. 7 14. 51 15. 50 16. 63 17. 35 18. 53 19. 72 20. 61

Extraire le 10e élément du vecteur

```
In [363]: x[10]
```

75

Extraire une partie du vecteur allant composé du 1er, 3e, ..., 19e élément

```
In [367]: x[seq(from = 1, to = 19, by = 2)]
```

1. 71 2. 86 3. 58 4. 14 5. 74 6. 59 7. 7 8. 50 9. 35 10. 72

Extraire les éléments divisibles par deux (*even numbers*)

```
In [371]: x[x%%2==0]
```

1. 18 2. 86 3. 58 4. 14 5. 74 6. 24 7. 50 8. 72

Extraire les éléments non divisibles par deux (*odd numbers*)

```
In [372]: x[x%%2!=0]
```

1. 71 2. 5 3. 19 4. 9 5. 75 6. 59 7. 7 8. 51 9. 63 10. 35 11. 53 12. 61

Tous les éléments sauf 3e, 5e et 17e éléments

```
In [375]: x[-c(3, 5, 17)]
```

1. 71 2. 18 3. 5 4. 19 5. 14 6. 9 7. 74 8. 75 9. 59 10. 24 11. 7 12. 51 13. 50 14. 63 15. 53 16. 72 17. 61

Dans le vecteur `x`, combien d'éléments sont pairs et combien sont impairs

```
In [373]: length(x[x%%2==0])
```

8

```
In [374]: length(x[x%%2!=0])
```

12

12 Question

Soit une matrice 12X7,

```
In [56]: x <- matrix(sample(1:100, 12*7), 12, 7)
         x
```

97	55	61	20	86	87	35
69	52	75	9	72	57	43
36	23	30	14	10	78	59
83	100	85	24	53	40	1
84	4	6	33	66	91	71
79	63	92	21	62	70	60
38	25	3	8	74	81	5
26	46	64	49	39	17	65
58	68	11	47	54	45	50
95	82	96	29	16	88	37
89	93	34	15	90	32	41
2	94	27	7	76	51	12

extraire l'élément de la 5e ligne et 6e colonne

```
In [379]: x[5,6]
```

45

Extraire tout le contenu de la 3e ligne et la 9 ligne

```
In [380]: x[c(3,9),]
```

96	87	17	77	86	1	69
72	3	97	19	27	10	56

Extraire **tout** le contenu des colonnes impaires

```
In [385]: x[,c(seq(from = 1, to = 7, by = 2))]
```

40	95	100	16
47	89	62	43
96	17	86	69
81	23	55	25
63	68	88	2
59	51	46	58
31	80	83	39
78	26	32	37
72	97	27	56
98	60	90	64
30	41	73	49
76	48	14	13

13 Questions mathématiques financière

$$1+i=\left(1+\frac{i^{(m)}}{m}\right)^{m}=(1-d)^{-1}=\left(1-\frac{d^{(m)}}{m}\right)^{-m}=e^{\delta} \tag{6}$$

13.1 Q1

En tenant compte de l'équation (6) Quelle est la valeur présente (arrondi à deux décimales) de 1000$ que vous aller recevoir dans 6 et 1/4 avec un taux *effective rate of discount* de 9.27% par année

```
In [393]: PV<-round(1000*(1-.0927)^6.25,2)
          PV
```

544.43

13.2 Q2

En tenant compte de l'équation (6) Quelle est la valeur accumulée (arrondi à deux décimales) de 1300$ que vous aller recevoir dans 10 et 1/2 avec un taux *effective rate of discount* de 5.3286% par année

```
In [22]: PV<-round(1300*(1-.053286)^-10.5,2)
         PV
```

2310.18

13.3 Q3

En tenant compte de l'équation (6) Quelle est la valeur accumulée (arrondi à deux décimales) de 50232$ que vous aller recevoir dans 17 ans avec un taux *nominal rate of interest* de 13% par année convertible trimestriellement

```
In [396]: FV=round(50232*(1+.13/4)^(17*4),2)
          FV
```

442083.77

13.4 Q4

En tenant compte de l'équation (6) Calculer la valeur présente de 82309$ à payer dans 8 ans avec un taux *nominal rate of discount* de 6% par année composée mensuellement

```
In [397]: FV<-round(82309*(1-(.06/12))^(12*8),2)
          FV
```

50870.16

13.5 Q4

Calculer la valeur présente d'une *annuity-immediate* avec des paiements de 50$ chaque 6 mois pour 10 ans au taux d'intérêt nominal de 4% composé semi-annuellement:

```
In [399]: pmts<-rep(50, 20)
          pmts
```

1. 50 2. 50 3. 50 4. 50 5. 50 6. 50 7. 50 8. 50 9. 50 10. 50 11. 50 12. 50 13. 50 14. 50 15. 50 16. 50 17. 50 18. 50 19. 50 20. 50

```
In [401]: actu<-1:20
          actu
```

1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10 11. 11 12. 12 13. 13 14. 14 15. 15 16. 16 17. 17 18. 18 19. 19 20. 20

```
In [406]: sum(pmts*(1.02)^-actu)
```

817.571667229856

14 Question probabilité

14.1 Q1

Simuler une des valeurs tirées d'une distribution normale. Imaginez une population dont la taille moyenne est de 1.70m et un écart-type de 0.1m. En utilisant `rnorm` simulez 100 valeurs et sauvegardez cer dernières dans un objet de type vecteur appelé `taille`.

Note Fixez votre seed à une valeur **123**

```
In [12]: set.seed(123)
         taille <- rnorm(n = 100, mean = 1.70, sd = .1)
```

Donnez un sommaire des statistiques descriptives du vecteur `taille`

```
In [13]: summary(taille)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.469   1.651   1.706   1.709   1.769   1.919
```

14.2 Q2

Quelle est la probabilité qu'une personne soir plus petit que 1.90m ? Votre réponse arrondie à deux décimales

```
In [14]: round(pnorm(1.90, mean = 1.70, sd = .1),2)
```

0.98

Note

Si on veut les formats en pourcentage, on peut utiliser la fonction `percent` du package `formattable`

```
In [7]: install.packages("formattable")
```

```
Updating HTML index of packages in '.Library'
Making 'packages.html' ... done
```

```
In [8]: library(formattable)
```

```
In [15]: percent(pnorm(1.90, mean = 1.70, sd = .1))
```

97.72%

Quelle est la probabilité que la taille d'une personne soit plus grande que 1.60 m

```
In [10]: percent(1-pnorm(1.60, mean = 1.70, sd = .1))
```

84.13%

14.3 Q3

Le temps d'attente (en minute) dans une clinique suit une loi exponentielle avec un taux de 1/50. Utiliser la fonction `rexp` afin de simuler les tempes d'attente pour 30 personnes dans cette clinique.

```
In [16]: set.seed(123)
         (patients <- rexp(rate = 1/50, n =30))
```

1. 42.1728630529201 2. 28.8305135443807 3. 66.4527433903372 4. 1.5788679554156 5. 2.81054880470037 6. 15.8250608188855 7. 15.7113646107649 8. 7.26334019564092 9. 136.311823216485 10. 1.45767235412863 11. 50.2415028845376 12. 24.0107363829887 13. 14.0506813768297 14. 18.8558915533567 15. 9.41420204471797 16. 42.4893064869052 17. 78.1601769807649 18. 23.9380208168278 19. 29.5467417687178 20. 202.050585568625 21. 42.1574865566887 22. 48.2935605549669 23. 74.2637897009036 24. 67.4022242871529 25. 58.4264492129392 26. 80.2926171529007 27. 74.8371434355986 28. 78.5326273447613 29. 1.5883871980738 30. 29.8924845643342

Quelle est la probabilité qu'une personne attende moins que 10 minutes?

```
In [18]: percent(pexp(q = 10,rate = 1/50))
```

18.13%

Supposons que la patience des gens atteint sa limite au bout de 60 minutes. Ça veut dire que s'ils attendent plus que 60 minutes, ils quittent la salle.

S'il y'a 100 personnes dans la salle, combien vont-ils quitter la salle?

```
In [21]: percent(1 - pexp(q=60, rate =1/50))
```

30.12%

3_1_cours

October 15, 2018

Table of Contents

1 Importation des données

Nous avons vu dans le dernier cours les *data frames*, nous l'avions créée manuellement. Toutefois, nous allons souvent importer des données en pratique sous plusieurs formats. Ces fichiers que nous allons importer seront souvent dans des fichiers `.csv` (*Comma-separated values*). Ces fichiers sont très populaires et ils sont générés par Excel .

1.1 read.csv

```
In [1]: options(repr.matrix.max.cols=8, repr.matrix.max.rows=8) #seulement pour afficher 8 ligne
```

On peut lire les fichiers `.csv` localement en précisant le chemin exacte menant vers du fichier en question.

```
In [2]: # test_csv <-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/exemple_1.
        test_csv <-read.csv("exemple_1.csv")
        test_csv
```

Segment	VitesseM	PuissanceEstim
Km1	31.9	130
Km2	33.3	165
Km3	28.1	130
Km4	30.8	133
Km5	27.7	103
Km6	31.2	154

Ou directement à partir du web:

```
In [1]: test_csv <-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/exemple_1.cs
        test_csv
```

Segment	VitesseM	PuissanceEstim
Km1	31.9	130
Km2	33.3	165
Km3	28.1	130
Km4	30.8	133
Km5	27.7	103
Km6	31.2	154

Lorsque nous écrivons `read.csv`, R traite importe ce fichier sous format `data frame`, il nous retourne les noms de colonnes, les lignes ainsi que la classe du df

```
In [2]: attributes(test_csv)
```

$names 1. 'Segment' 2. 'VitesseM' 3. 'PuissanceEstim'

$class 'data.frame'

$row.names 1. 1 2. 2 3. 3 4. 4 5. 5 6. 6

Dans la méthode `read.csv`, il existe un argument optionnel `_header_` qui est par défaut `header=T`. Cet argument spécifie si les données que nous voulons importer possèdent des noms de colonne (`header=TRUE` ~ `header=T`.) ou pas (`header=FALSE` ~ `header=F`.). Regardons ce que ça donnerait si nous changeons la valeur `header=F`;

```
In [9]: test_csv <-read.csv("exemple_1.csv", header = F)
        test_csv
```

V1	V2	V3
Segment	VitesseM	PuissanceEstim
Km1	31.9	130
Km2	33.3	165
Km3	28.1	130
Km4	30.8	133
Km5	27.7	103
Km6	31.2	154

On remarque que R crée des noms de colonnes appelés `V1, V2...etc.`

```
In [11]: exemple <-read.csv("exemple_1.csv", header = T)
         exemple
```

Segment	VitesseM	PuissanceEstim
Km1	31.9	130
Km2	33.3	165
Km3	28.1	130
Km4	30.8	133
Km5	27.7	103
Km6	31.2	154

Regardons la classe de la variable "Segment";

```
In [12]: class(exemple$Segment)
```

'factor'

Surprise!. En effet, il existe une autre option dans la méthode `read.csv` qui permet de traiter les catégories en type caractère.

```
In [14]: exemple <-read.csv("exemple_1.csv", header = T, stringsAsFactors=F)
         exemple
```

Segment	VitesseM	PuissanceEstim
Km1	31.9	130
Km2	33.3	165
Km3	28.1	130
Km4	30.8	133
Km5	27.7	103
Km6	31.2	154

Maintenant, regardons la classe de la variable "Segment"

```
In [15]: class(exemple$Segment)
```

'character'

1.1.1 Une partie seulement du df

Il est possible de lire seulement certaines colonnes du fichier csv qu'on veut importer;

```
In [30]: exemple <-read.csv("exemple_1.csv", header = T, stringsAsFactors=F)[,2:3]
         exemple
```

VitesseM	PuissanceEstim
31.9	130
33.3	165
28.1	130
30.8	133
27.7	103
31.2	154

```
In [31]: exemple <-read.csv("exemple_1.csv", header = T, stringsAsFactors=F)[,c(1,3)]
         exemple
```

Segment	PuissanceEstim
Km1	130
Km2	165
Km3	130
Km4	133
Km5	103
Km6	154

1.2 read.table

Une autre façon d'importer des données à l'intérieur des df est d'utiliser la méthode `read.table` qui traite les fichiers `text`;

```
In [17]: read.table("exemple_1.txt", header=T)
```

Segment.VitesseM.PuissanceEstim
Km1,31.9,130
Km2,33.3,165
Km3,28.1,130
Km4,30.8,133
Km5,27.7,103
Km6,31.2,154

On voit bien que les colonnes n'ont pas été séparées comme il faut. Nous devons spécifier les caractères qui séparent ces variables.

```
In [18]: read.table("exemple_1.txt", header=T, sep = ",")
```

Segment	VitesseM	PuissanceEstim
Km1	31.9	130
Km2	33.3	165
Km3	28.1	130
Km4	30.8	133
Km5	27.7	103
Km6	31.2	154

1.3 En utilisant le package `RCurl`

Il est possible d'utiliser la bibliothèque `RCurl` qui offre plus d'options. Dans ce cours, nous nous limitons à l'utilisation de `read.csv`. Pour plus d'informations sur cette bibliothèque, vous pouvez lire plus de détails la documentation de ce *package*.

```
In [23]: install.packages("RCurl")

Updating HTML index of packages in '.Library'
Making 'packages.html' ... done


In [24]: library(RCurl)

Loading required package: bitops


In [26]: x <- getURL("https://raw.githubusercontent.com/aronlindberg/latent_growth_classes/maste
         y <- read.csv(text = x)

In [28]: head(y)
```

top100_repository_name	month	monthly_increase	monthly_begin_at	monthly_end_with
Bukkit	2012-03	9	431	440
Bukkit	2012-04	19	438	457
Bukkit	2012-05	19	455	474
Bukkit	2012-06	18	475	493
Bukkit	2012-07	15	492	507
Bukkit	2012-08	50	506	556

2 Traiter les valeurs manquantes

Afin d'illustrer le traitement des valeurs manquantes dans R, importons les données de l'exemple 2_2

```
In [7]: read.table("https://raw.githubusercontent.com/nmeraihi/data/master/exemple_2_2.txt", hea
```

km	temps	vitesseMoyenne	puissanceMoyenne	bpm
1.24	4:01	19.1	160	134
NA	9:42	30.2	133	146
1.02	1:57	30.8	141	139
17.61	36:11	29.2	NA	144
9.27	19:10	29.0	121	143

On peut appliquer un test booléen afin de vérifier l'existance des valeurs manquantes comme suit;

```
In [8]: df<-read.table("https://raw.githubusercontent.com/nmeraihi/data/master/exemple_2_2.txt",
        is.na(df)
```

km	temps	vitesseMoyenne	puissanceMoyenne	bpm
FALSE	FALSE	FALSE	FALSE	FALSE
TRUE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE

Cette fonction nous retourne un *data frame* du même format que le *df* test. Le résultat obtrenu sont valeurs `TRUE` sur les éléments manquants, et des `FALSE` sur les valeurs existantes.

On peut faire le test sur une partie précise du *df*;

```
In [9]: is.na(df[1,1])
```

FALSE

```
In [10]: is.na(df[2,1])
```

TRUE

Mais pourquoi préocupe t-on tant des valeurs manquantes? Eh bien, les valeurs manquantes sont le cauchemar #1 de toute personne qui manipule les données, que ce soit en entreprise ou pour un utilisation personnelle.

Essayons de faire un calcul de la moyenne du nombre de km;

```
In [11]: mean(df$km)
```

```
[1] NA
```

R nous retourne NA même si nous avons une seule observation qui est manquante
On peut régler ce problème avec la fonction na.omit()

```
In [12]: mean(na.omit(df$km))
```

7.285
Le calcul de la moyenne a été fait sur les variables;

```
In [13]: na.omit(df$km)
```

1. 1.24 2. 1.02 3. 17.61 4. 9.27
La fonction mean possède un argument optionnel appelé na.rm = qui ignore les valeurs manquantes;

```
In [14]: mean(df$km,na.rm = T)
```

7.285
Lorsque nous utilisons la fonction ns.omit, le df se réduit à un df qui ne possède aucune ligne contenant les valeurs manquantes;

```
In [15]: na.omit(df)
```

	km	temps	vitesseMoyenne	puissanceMoyenne	bpm
1	1.24	4:01	19.1	160	134
3	1.02	1:57	30.8	141	139
5	9.27	19:10	29.0	121	143

On peut aller modifier directement la valeur de cet élément

```
In [16]: df[2,1]<-4.84
          df[4,4]<-125
```

```
In [11]: df
```

km	temps	vitesseMoyenne	puissanceMoyenne	bpm
1.24	4:01	19.1	160	134
4.84	9:42	30.2	133	146
1.02	1:57	30.8	141	139
17.61	36:11	29.2	125	144
9.27	19:10	29.0	121	143

2.1 Remplacer toutes les valeurs manquantes:

Des fois, il peut être utile de remplacer toutes les valeurs manquantes par des 0. Je dis bien des fois, car les valeurs manquantes sont absentes et non des 0.

```
In [18]: df<-read.table("https://raw.githubusercontent.com/nmeraihi/data/master/exemple_2_2.txt"
         df
```

km	temps	vitesseMoyenne	puissanceMoyenne	bpm
1.24	4:01	19.1	160	134
NA	9:42	30.2	133	146
1.02	1:57	30.8	141	139
17.61	36:11	29.2	NA	144
9.27	19:10	29.0	121	143

Nous remplaçons alors les `NA` par `0` ou par toute autre valeur comme suit;

```
In [19]: df[is.na(df)] <- 0
```

```
In [20]: df
```

km	temps	vitesseMoyenne	puissanceMoyenne	bpm
1.24	4:01	19.1	160	134
0.00	9:42	30.2	133	146
1.02	1:57	30.8	141	139
17.61	36:11	29.2	0	144
9.27	19:10	29.0	121	143

3_2_cours

October 15, 2018

Table of Contents


```
In [2]: options(repr.matrix.max.cols=8, repr.matrix.max.rows=5)
```

1 Opérations sur les matrices

1.1 apply

Nous avons vu qu'il était possible de faire des opérations sur les matrices en ligne ou en colonne. Toutefois, ce n'est pas toutes les fonctions statistiques qui sont applicables sur des colonnes et/ou lignes comme `colMeans`. Pour appliquer d'utres sortes de fonctions, nous devons utiliser la fonction `apply`.

On peut alors utiliser `apply` lorsqu'on veut appliquer un calcule ou une fonction quelconque (FUN) sur des colonnes ou des lignes d'une matrice (incluant les matrices plus que 2D)

Soit une matrice de 12 premiers entiers;

```
In [2]: m<-matrix(1:12, nrow=3)
        m
```

1 4 7 10
2 5 8 11
3 6 9 12

Calculons le logarithme naturel de chaque élément de cette matrice:

```
In [3]: h<-apply(m, c(1,2), log) #c(1,2) ça veut dire sur ligne et colonne
        h
```

0.0000000 1.386294 1.945910 2.302585
0.6931472 1.609438 2.079442 2.397895
1.0986123 1.791759 2.197225 2.484907

Créons une matrice `3 x 1` qui nous retourne le résultat de la somme de chaque ligne;

```
In [4]: h<-matrix(apply(m, 1, sum))
        h
```

22
26
30

Si nous comparerons à la fonction `rowSums` que nous avons vue;

```
In [7]: rowSums(m)
```

1. 22 2. 26 3. 30

1.2 lapply

La fonction lapply applique une fonction quelconque (FUN) à tous les éléments d'un vecteur ou d'une liste X et retourne le résultat sous forme de liste.

Dans l'exemple suivant, nous avons une liste de trois vecteurs {vecteur_1, vecteur_2, vecteur_3} de taille différente, on voudrait savoir quelle est la taille de chaque élément, on voudrait la réponse dans une **liste**;

```
In [4]: x <- list(vecteur_1 = 1, vecteur_2 = 1:17, vecteur_3 = 55:97)
        lapply(x, FUN = length)
```

$vecteur_1 1

$vecteur_2 17

$vecteur_3 43

Dans le résultat ci-haut, la fonction `lapply` nous a retourné une liste de trois éléments avec la taille de chaque vecteur.

Regardons un autre exemple où nous cherchons à créer quatre échantillons aléatoires de taille {5, 6, 7, 8} tirés du vecteur `x= 1 2 3 4 5 6 7 8 9 10`

```
In [5]: set.seed(123)
        lapply(5:8, sample, x = 1:10)
```

1. (a) 3 (b) 8 (c) 4 (d) 7 (e) 6
2. (a) 1 (b) 5 (c) 8 (d) 4 (e) 3 (f) 9
3. (a) 5 (b) 7 (c) 10 (d) 1 (e) 6 (f) 2 (g) 9
4. (a) 4 (b) 9 (c) 8 (d) 5 (e) 10 (f) 7 (g) 3 (h) 6

1.3 sapply

Dans certains cas, on voudrait appliquer une fonction quelconque sur une liste, mais on ne veut pas que `R` nous retourne une une liste, on désire plutôt que `R` nous retourne un vecteur. La fonction `sapply` fait exactement cela. Le résultat est donc **s**implifiée par rapport à celui de `lapply`, d'où le nom de la fonction.

La taille de chaque élément de notre liste;

```
In [7]: sapply(x, FUN = length)
```

vecteur_1 1 **vecteur_2** 17 **vecteur_3** 43

ou la somme de chaque élément de notre liste `x`

```
In [6]: sapply(x, FUN = sum)
```

vecteur_1 1 **vecteur_2** 153 **vecteur_3** 3268

Si le résultat de chaque application de la fonction est un vecteur et que les vecteurs sont tous de la même longueur, alors sapply retourne une matrice, remplie comme toujours par colonne :

```
In [7]: (x <- lapply(rep(5, 3), sample, x = 1:10))
```

1. (a) 6 (b) 10 (c) 3 (d) 2 (e) 9
2. (a) 10 (b) 7 (c) 9 (d) 1 (e) 3
3. (a) 8 (b) 2 (c) 3 (d) 9 (e) 1

```
In [8]: sapply(x, sort)
```

2	1	1
3	3	2
6	7	3
9	9	8
10	10	9

1.4 autre

Il existe d'autres façons de manipuler les matrices, listes, vecteurs ...etc. Dans ce cours nous avons couvert les trois principaux, toutefois, on peut avoir besoin dans certains cas d'utiliser `vapply`, `mapply`, `Map`, `rapply` ou même `tapply` qui s'apparentent tous aux trois fonctions que nous avons couverts avec plus d'options ou format différent du résultat obtenu.

2 Manipulation avec dplyr

2.1 le package dplyr

Dans ce cours, afin de manipuler les données, nous allons utiliser la librairie `dplyr` qui assure une manipulation plus intuitive des données. Toutefois, vous pouvez utiliser tout autre libraire ou même les fonctions de base de `R`.

```
In [11]: # install.packages("dplyr")
```

D'abord, téléchargeons un petit *df* afin d'illustrer la théorie. Dans ce *df*, nous avons les données d'un cycliste qui est sorti un jour d'été faire un petit tour dans l'île de Montréal. Chaque observation, représente le nombre de km parcourus d'un parcours (*lap*), le temps que ça a pris, la vitesse moyenne en `km/h` de chaque *lap*, la puissance moyenne en `watts` et finalement, les battements de cours par minutes.

```
In [9]: df<-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/exemple_2.txt")
        df
```

km	temps	vitesseMoyenne	puissanceMoyenne	bpm
1.24	4:01	19.1	160	134
4.84	9:42	30.2	133	146
1.02	1:57	30.8	141	139
17.61	36:11	29.2	125	144
9.27	19:10	29.0	121	143

Il possible d'ordonner les données avec la fonction de base de `R` appelé `order`. Par exemple, on voudrait ordonner notre *df* en ordre croissant sur la variable `puissanceMoyenne`

```
In [10]: df[order(df$puissanceMoyenne),]
```

	km	temps	vitesseMoyenne	puissanceMoyenne	bpm
5	9.27	19:10	29.0	121	143
4	17.61	36:11	29.2	125	144
2	4.84	9:42	30.2	133	146
3	1.02	1:57	30.8	141	139
1	1.24	4:01	19.1	160	134

Par défaut, l'ordre est est croissant, on peut le rendre décroissant en ajoutant l'argument `decreasing = T`

```
In [11]: df[order(df$vitesseMoyenne, decreasing = T),]
```

	km	temps	vitesseMoyenne	puissanceMoyenne	bpm
3	1.02	1:57	30.8	141	139
2	4.84	9:42	30.2	133	146
4	17.61	36:11	29.2	125	144
5	9.27	19:10	29.0	121	143
1	1.24	4:01	19.1	160	134

Toutefois, lorsque nous avons une base de données comportant un nombre plus important de variables, la syntaxe peut devenir plus compliquée et lourde d'écriture. Regardons un autre exemple:

```
In [3]: df_ass <-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/assurance.csv"
        head(df_ass)
```

numeropol	debut_pol	fin_pol	freq_paiement	⋯	cout6	cout7	nbsin	equipe
4	11-4-1996	10-4-1997	12	⋯	NA	NA	0	3
4	11-4-1997	10-4-1998	12	⋯	NA	NA	0	3
4	11-4-2002	17-7-2002	12	⋯	NA	NA	0	3
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮
4	11-4-2003	10-4-2004	12	⋯	NA	NA	0	3
12	3-5-1995	2-5-1996	1	⋯	NA	NA	0	3

Affichons notre base de données en ordre croissant sur le nombre de sinistres et le numéro de police;

```
In [15]: df_ass[order(df_ass$nbsin, df_ass$numeropol, decreasing = T),]
```

	numeropol	debut_pol	fin_pol	freq_paiement	···	cout6	cout7	nbsin	equipe
988	2006	13-6-1996	12-6-1997	12	···	NA	NA	2	3
902	1820	1-10-1996	30-9-1997	1	···	NA	NA	2	3
861	1733	10-5-1998	9-5-1999	1	···	NA	NA	2	3
⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮
4	4	18-7-2002	10-4-2003	12	···	NA	NA	0	3
5	4	11-4-2003	10-4-2004	12	···	NA	NA	0	3

Dans le tableau affiché ci-haut, on voit bien que cette fonction ne nous permet pas d'appliquer un ordre croissant ou décroissant sur une variable précise

2.2 arrange

Maintenant, utilisons le paquet (*package*) `dplyr` qui nous permet de plus facilement d'appliquer un ordre quelconque sur une variable précise indépendamment des autres variables;

```
In [4]: library(dplyr, warn.conflicts = FALSE)
```

```
In [17]: arrange(df_ass, desc(nbsin), numeropol)
```

numeropol	debut_pol	fin_pol	freq_paiement	···	cout6	cout7	nbsin	equipe
71	15-2-1996	14-2-1997	1	···	NA	NA	2	3
79	20-11-1997	21-6-1998	12	···	NA	NA	2	3
116	4-9-1998	11-6-1999	1	···	NA	NA	2	3
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮
2036	14-3-2000	26-2-2001	12	···	NA	NA	0	3
2036	27-2-2001	13-3-2001	12	···	NA	NA	0	3

2.3 select

Ce paquet nous permet aussi de sélectionner des variables d'intérêt. Par exemple, dans notre `df_ass`, on désire seulement sélectionner les variables `numeropol`, `type_territoire` et `nbsin`

```
In [5]: select(df_ass, numeropol,type_territoire, nbsin)
```

numeropol	type_territoire	nbsin
4	Semi-urbain	0
4	Semi-urbain	0
4	Semi-urbain	0
⋮	⋮	⋮
2036	Semi-urbain	0
2036	Semi-urbain	1

2.4 filter

Afin de filtrer des données sur des observations d'intérêt. On peut utiliser la fonction de base de **R** `which`. Par exemple dans les données `Cars93` du package `MASS`, on voudrait extraire les véhicules ayant 8 cylindres. On voudrait également afficher que les deux variables `'Horsepower'` et `'Passengers'`

```
In [19]: library(MASS, warn.conflicts = F)
```

```
In [20]: Cars93[which(Cars93$Cylinders==8), c('Horsepower' , 'Passengers')]
```

	Horsepower	Passengers
10	200	6
11	295	5
18	170	6
⋮	⋮	⋮
48	278	5
52	210	6

Toutefois, la fonction `filter` de la librairie `dyplr` est plus flexible lorsqu'il s'agit d'appliquer des filtres plus complexes. Essayons le même exemple avec cette fonction;

```
In [21]: filter(Cars93, Cylinders==8)[c('Horsepower' , 'Passengers')]
```

Horsepower	Passengers
200	6
295	5
170	6
⋮	⋮
278	5
210	6

Si l'on cherche les médecins qui ont eu deux sinistres dans notre base de données `df_ass`;

```
In [22]: filter(df_ass, nbsin==2, type_prof=="Médecin")
```

numeropol	debut_pol	fin_pol	freq_paiement	⋯	cout6	cout7	nbsin	equipe
71	15-2-1996	14-2-1997	1	⋯	NA	NA	2	3
140	15-4-1995	14-4-1996	12	⋯	NA	NA	2	3
1820	1-10-1996	30-9-1997	1	⋯	NA	NA	2	3

2.5 mutate

Dans ce package, on trouve aussi la fonction `mutate` qui permet d'ajouter de nouvelles variables à notre df

```
In [23]: mutate(df, arrondi=round(df$vitesseMoyenne,0))
```

km	temps	vitesseMoyenne	puissanceMoyenne	bpm	arrondi
1.24	4:01	19.1	160	134	19
4.84	9:42	30.2	133	146	30
1.02	1:57	30.8	141	139	31
17.61	36:11	29.2	125	144	29
9.27	19:10	29.0	121	143	29

Ajoutons maintenant trois nouvlles variables;

```
In [24]: mutate(df, arrondi=round(df$vitesseMoyenne,0), segementStrava=paste("segment",1:5,sep =
```

km	temps	vitesseMoyenne	puissanceMoyenne	bpm	arrondi	segementStrava	arrondi_2
1.24	4:01	19.1	160	134	19	segment_1	9.5
4.84	9:42	30.2	133	146	30	segment_2	15.0
1.02	1:57	30.8	141	139	31	segment_3	15.5
17.61	36:11	29.2	125	144	29	segment_4	14.5
9.27	19:10	29.0	121	143	29	segment_5	14.5

2.6 summarize

La fonction `summarize` est très similaire à la fonction `mutate`. Toutefois, contrairement à `mutate`, la fonction `summarize` ne travaille pas sur une copie du df, mais elle crée un tout nouveau df avec les nouvelles variables.

```
In [25]: summarise(df,TotalKmParcour=sum(km))
```

TotalKmParcour
33.98

```
In [26]: summarise(df,TotalKmParcour=sum(km), vitesseMoyenne= mean(df$vitesseMoyenne), puissance
```

TotalKmParcour	vitesseMoyenne	puissanceMoyenne
33.98	27.66	136

On voit bien que l'écriture du code commence à être un peu plus compliquée lorsque nous avons plusieurs parenthèses dans notre fonction. Pour remédier à ce problème, nous verrons la notion de *piiping*;

2.7 L'opérateur Pipe: %>%

Avant d'aller plus loin, introduisons l'opérateur de *pipe*:%>%. **dplyr** importe cet opérateur d'une autre librairie (`magrittr`). Cet opérateur vous permet de diriger la sortie d'une fonction vers l'entrée d'une autre fonction. Au lieu d'imbriquer des fonctions (lecture de l'intérieur vers l'extérieur), l'idée de *piping* est de lire les fonctions de gauche à droite.

Crédit de l'image Pipes in R Tutorial For Beginners

Lorsque nous avons écrit:

```
In [6]: select(df_ass, numeropol, type_territoire, nbsin)
```

numeropol	type_territoire	nbsin
4	Semi-urbain	0
4	Semi-urbain	0
4	Semi-urbain	0
⋮	⋮	⋮
2036	Semi-urbain	0
2036	Semi-urbain	1

Si on lit ce que nous avons écrit précédemment de l'intérieur vert l'extérieur, en utilisant le *piping*, nous aurons ceci:

```
In [7]: df_ass %>%
            select (numeropol,type_territoire, nbsin)
```

numeropol	type_territoire	nbsin
4	Semi-urbain	0
4	Semi-urbain	0
4	Semi-urbain	0
⋮	⋮	⋮
2036	Semi-urbain	0
2036	Semi-urbain	1

ou;

```
In [8]: df_ass %>%
            select (numeropol,type_territoire, nbsin) %>%
            head
```

numeropol	type_territoire	nbsin
4	Semi-urbain	0
4	Semi-urbain	0
4	Semi-urbain	0
⋮	⋮	⋮
4	Semi-urbain	0
12	Semi-urbain	0

2.8 group_by

Nous pouvons aussi grouper les données comme nous le faisions dans `SAS` avec les `PROC SQL`

```
In [11]: df_ass$coutTot<-rowSums(df_ass[,c(19:25)], na.rm = T, dims = 1)
```

```
In [12]: df_ass
```

numeropol	debut_pol	fin_pol	freq_paiement	⋯	cout7	nbsin	equipe	coutTot
4	11-4-1996	10-4-1997	12	⋯	NA	0	3	0
4	11-4-1997	10-4-1998	12	⋯	NA	0	3	0
4	11-4-2002	17-7-2002	12	⋯	NA	0	3	0
⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮	⋮
2036	27-2-2001	13-3-2001	12	⋯	NA	0	3	0.0
2036	14-3-2001	13-3-2002	12	⋯	NA	1	3	231051.8

```
In [13]: summarise(df_ass,TotalNbSin=sum(nbsin), TotCout= sum((coutTot), na.rm = T))
```

TotalNbSin	TotCout
156	1078791

Cherchons par exemple nombre de sinistres totaux ainsi que leurs coûts par territoire. En utilisant la syntaxe du *piping*, ça devient plus facile d'inclure plus de sous-groupes;

```
In [14]: df_ass %>%
             group_by(type_territoire) %>%
             summarise(TotalNbSin=sum(nbsin),
                       TotCout= sum((coutTot), na.rm = T)
                      )
```

type_territoire	TotalNbSin	TotCout
Rural	51	547105.01
Semi-urbain	80	471157.64
Urbain	25	60528.81

3 Jointure des bases des données

Dans cette section, nous allons joindre deux ou plusieurs df. Mais d'abord importons deux df afin illustrer quelques exemples;

```
In [15]: df_demo <-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/donnes_demo.
         df_demo
```

name	province	company	langue	date_naissance	agee
Shane Robinson	Nova Scotia	May Ltd	fr	1944-10-20	72
Courtney Nguyen	Saskatchewan	Foley, Moore and Mitchell	en	1985-12-09	31
Lori Washington	Yukon Territory	Robinson-Reyes	fr	1970-01-27	47
⋮	⋮	⋮	⋮	⋮	⋮
Heidi Freeman	Northwest Territories	Singh, Esparza and Santos	en	1951-06-07	65
Morgan Buchanan	Northwest Territories	Rollins Inc	fr	1971-07-31	45

```
In [16]: df_auto <-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/cars_info.cs
         df_auto
```

numeropol	marque_voiture	couleur_voiture	presence_alarme	license_plate
1	Autres	Autre	0	DW 3168
5	RENAULT	Autre	0	926 1RL
13	RENAULT	Autre	1	SOV 828
⋮	⋮	⋮	⋮	⋮
84	HONDA	Autre	0	CBV 102
91	BMW	Autre	1	UOR-0725

Dans ces deux df, nous avons une colonne en commun `numeropol`

```
In [17]: df_demo$numeropol
```

1. 1 2. 5 3. 13 4. 16 5. 22 6. 28 7. 29 8. 49 9. 53 10. 57 11. 59 12. 65 13. 67 14. 68 15. 69 16. 72 17. 78 18. 83 19. 84 20. 91

```
In [18]: df_auto$numeropol
```

1. 1 2. 5 3. 13 4. 16 5. 22 6. 22 7. 28 8. 29 9. 49 10. 53 11. 53 12. 57 13. 59 14. 65 15. 65 16. 67 17. 68 18. 69 19. 69 20. 72 21. 78 22. 83 23. 84 24. 84 25. 91

On peut voir l'index des lignes qui se trouvent dans les deux df

```
In [19]: match(df_demo$numeropol, df_auto$numeropol)
```

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

```
In [20]: df_demo$numeropol[match(df_demo$numeropol, df_auto$numeropol)]
```

1. 1 2. 5 3. 13 4. 16 5. 22 6. 29 7. 49 8. 53 9. 57 10. 65 11. 67 12. 68 13. 72 14. 78 15. 83 16. 91 17. <NA> 18. <NA> 19. <NA> 20. <NA>

On peut aussi faire un test logique sur la présence des observations du df_demo dans df_auto;

```
In [21]: df_demo$numeropol %in% df_auto$numeropol
```

1. TRUE 2. TRUE 3. TRUE 4. TRUE 5. TRUE 6. TRUE 7. TRUE 8. TRUE 9. TRUE 10. TRUE 11. TRUE 12. TRUE 13. TRUE 14. TRUE 15. TRUE 16. TRUE 17. TRUE 18. TRUE 19. TRUE 20. TRUE

ou le contraire maintenant

```
In [22]: df_auto$numeropol %in% df_demo$numeropol
```

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

Dans ce cas toutes les variables se trouvent dans les deux df

```
In [23]: merge(df_demo,df_auto, by.x = "numeropol",  by.y = "numeropol") # x est le df_demo et y
```

numeropol	name	province	company	···	marque_voitu
1	Shane Robinson	Nova Scotia	May Ltd	···	Autres
5	Courtney Nguyen	Saskatchewan	Foley, Moore and Mitchell	···	RENAULT
13	Lori Washington	Yukon Territory	Robinson-Reyes	···	RENAULT
⋮	⋮	⋮	⋮	⋱	⋮
84	Heidi Freeman	Northwest Territories	Singh, Esparza and Santos	···	HONDA
91	Morgan Buchanan	Northwest Territories	Rollins Inc	···	BMW

Que serait-il arrivé si l'on n'avait pas spécifié les arguments `by.x = "numeropol",  by.y = "numeropol"`?

```
In [24]: merge(df_demo,df_auto)
```

numeropol	name	province	company	···	marque_voitu
1	Shane Robinson	Nova Scotia	May Ltd	···	Autres
5	Courtney Nguyen	Saskatchewan	Foley, Moore and Mitchell	···	RENAULT
13	Lori Washington	Yukon Territory	Robinson-Reyes	···	RENAULT
⋮	⋮	⋮	⋮	⋱	⋮
84	Heidi Freeman	Northwest Territories	Singh, Esparza and Santos	···	HONDA
91	Morgan Buchanan	Northwest Territories	Rollins Inc	···	BMW

Cela a bien fonctionné, car R a automatiquement trouvé les noms de colonnes communs au deux df;

Maintenant, changeons les noms de colonnes et voyons ce qui arrive

```
In [25]: names(df_auto)[names(df_auto)=="numeropol"] <- "auto_numpol"
```

Bien évidemment, cela crée une jointure croisée comme on l'avait vu dans les cours de SAS

```
In [26]: head(merge(df_demo,df_auto))
```

name	province	company	langue	⋯	marque_voiture	couleu
Shane Robinson	Nova Scotia	May Ltd	fr	⋯	Autres	Autre
Courtney Nguyen	Saskatchewan	Foley, Moore and Mitchell	en	⋯	Autres	Autre
Lori Washington	Yukon Territory	Robinson-Reyes	fr	⋯	Autres	Autre
⋮	⋮	⋮	⋮	⋱	⋮	⋮
Jeffrey Garcia	Nunavut	Berger-Thompson	en	⋯	Autres	Autre
Colleen Coleman	Saskatchewan	Simmons-Smith	en	⋯	Autres	Autre

On vient bien que dans la dernière colonne `license_plate`, nous obtenons la même observation ce qui est clairement une erreur;

Corrigeons le problème;

```
In [27]: head(merge(df_demo,df_auto, by.x = "numeropol",  by.y = "auto_numpol"))
```

numeropol	name	province	company	⋯	marque_voiture	c
1	Shane Robinson	Nova Scotia	May Ltd	⋯	Autres	A
5	Courtney Nguyen	Saskatchewan	Foley, Moore and Mitchell	⋯	RENAULT	A
13	Lori Washington	Yukon Territory	Robinson-Reyes	⋯	RENAULT	A
⋮	⋮	⋮	⋮	⋱	⋮	⋮
22	Jeffrey Garcia	Nunavut	Berger-Thompson	⋯	VOLKSWAGEN	A
22	Jeffrey Garcia	Nunavut	Berger-Thompson	⋯	VOLKSWAGEN	A

left_join

la fonction `left_join` prend toute l'information de gauche et l'information existante de la partie droite qui est basée sur le critère en commun

```
In [28]: x<-data.frame(nom=c("Gabriel", "Adel", "NM", "Mathieu", "Amine", "Mohamed"),
                       bureaux=c("5518", "4538", "5518", "5517", "4538", "4540"))
```

x

nom	bureaux
Gabriel	5518
Adel	4538
NM	5518
⋮	⋮
Amine	4538
Mohamed	4540

```
In [29]: y<-data.frame(nom=c("Gabriel", "Adel", "JP", "Mathieu", "Amine"),
                       diplome=c("M.Sc", "Ph.D", "Ph.D", "Ph.D", "Ph.D"))
```

y

nom	diplome
Gabriel	M.Sc
Adel	Ph.D
JP	Ph.D
Mathieu	Ph.D
Amine	Ph.D

```
In [30]: left_join(x,y,by = "nom")
```

```
Warning message:
‘‘Column `nom` joining factors with different levels, coercing to character vector’’
```

nom	bureaux	diplome
Gabriel	5518	M.Sc
Adel	4538	Ph.D
NM	5518	NA
⋮	⋮	⋮
Amine	4538	Ph.D
Mohamed	4540	NA

3.1 inner_join

Cette fonction permet de retourner **seulement** les éléments en commun des deux df

```
In [31]: inner_join(x,y,by = "nom")
```

```
Warning message:
‘‘Column `nom` joining factors with different levels, coercing to character vector’’
```

nom	bureaux	diplome
Gabriel	5518	M.Sc
Adel	4538	Ph.D
Mathieu	5517	Ph.D
Amine	4538	Ph.D

3.2 semi_join

Cette fonction retourne seulement les éléments du premier df qui se retrouve dans le deuxième df, sans nous retourner les éléments de ce dernier

```
In [ ]: semi_join(x,y,by = "nom")
```

3.3 anti_join

Cette fonction le contraire de la précédente

```
In [ ]: anti_join(x,y,by = "nom")
```

3_4_Solutions

October 15, 2018

```
In [38]: options(repr.matrix.max.cols=8, repr.matrix.max.rows=5)
```

```
In [28]: path<-"https://raw.githubusercontent.com/nmeraihi/data/master/"
```

1 Question 1

1.1 a)

Importer les données `qc_hommes_2.csv` à partir du répertoire data github dans un *data frame* df

```
In [29]: df<-read.csv(paste(path,"qc_hommes_2.csv",sep = ""), sep=",")
```

```
In [30]: head(df)
```

age	lx
0 an	100000
1 an	99501
2 ans	99483
3 ans	99467
4 ans	99454
5 ans	99442

```
In [31]: tail(df)
```

	age	lx
106	105 ans	96
107	106 ans	51
108	107 ans	26
109	108 ans	13
110	109 ans	6
111	110 ans et plus	3

1.2 b)

Dans la colonne age, garder seulement la partie numérique. Vous devriez alors obtenir age={0,1,2 ...}

```
In [153]: df$age<-gsub("ans", "", df$age)
          df$age<-gsub("an", "", df$age)
```

1.3 c)

À ce df, ajouter une nouvelle colonne `dx` (nombre de décès entre l'âge x et x+n). Donc dx est le nombre de décès qui surviennent dans chaque intervalle d'âge au sein d'une cohorte initiale de 100 000 naissances vivantes à l'âge 0.

$$d_x = l_x - l_{x+1}$$

```
In [154]: a<-df[-nrow(df), 2]-df[-1, 2]
          a<-c(a, a[length(a)])
          df$dx<-a
```

1.4 d)

Calculer qx (quotient de mortalité entre l'âge x et x+n). Donc qx est probabilité qu'un individu d'âge x décède avant d'atteindre l'âge x+n.

$$q_x = \frac{d_x}{l_x}$$

```
In [155]: df$qx<-round(df$dx/df$lx,5)
```

```
In [156]: head(df)
```

age	lx	dx	qx
0	100000	499	0.00499
1	99501	18	0.00018
2	99483	16	0.00016
3	99467	13	0.00013
4	99454	12	0.00012
5	99442	11	0.00011

1.5 e)

Maintenant que vous avez toutes les données, on peut calculer la probabilité qu'un individu d'âge x survive jusqu'à l'âge x+n.

$$tP_x = \frac{l_{x+t}}{l_x}$$

Calculer la probabilité qu'un individu de 22 ans survive les trois prochaines années

```
In [159]: age<-22
          t<-3
          p<-df[age+1+t,2]/df[age+1,2]
          p
```

0.998192831903079

```
In [160]: library(formattable)
```

```
In [161]: percent(p)
```

99.82%

2 Question 2

```
In [162]: Id=c(1,2,3,4)
          Age=c(14,12,15,10)
          Sex=c('F','M','M','F')
          Code=c('a','b','c','d')
          df1=data.frame(Id,Age)
          df2=data.frame(Id,Sex,Code)
```

Avec les données suivantes;

```
In [163]: df1
```

Id	Age
1	14
2	12
3	15
4	10

```
In [164]: df2
```

Id	Sex	Code
1	F	a
2	M	b
3	M	c
4	F	d

Créer un *data frame* `M` qui fait une jointure de `df1` et `df2`

```
In [165]: M=merge(df1,df2,by='Id')
          M
```

Id	Age	Sex	Code
1	14	F	a
2	12	M	b
3	15	M	c
4	10	F	d

3 Question 3

Selon un journaliste de la CNBC, le prix de l'action de Apple (AAPL) est très corrélé avec le prix de l'action de Boeing Co (BA).

Calculer la corrélation des prix Adj Close **mensuels** de ces deux compagnies sur la période allant du 2016-11-01 au 2017-10-01.

Indice: créer deux vecteur avec les valeurs des prix. Vous pouvez importer les données à partir de finance yahoo dans la section *Historical Data* avec les dates et périodes indiquées ci-haut.

```
In [2]: path<-"https://raw.githubusercontent.com/nmeraihi/data/master/"
```

```
In [170]: df_app <-read.csv(paste(path,"AAPL_month.csv",sep = ""), header = T)
```

```
In [176]: df_ba <-read.csv(paste(path,"BA_month.csv",sep = ""), header = T)

In [179]: a<-cbind(df_app$Adj.Close,df_ba$Adj.Close)

In [180]: colnames(a)<-c("Apple", "Boeing")

In [181]: rownames(a)<-seq(as.Date("2016/11/1"), by = "month", length.out = 12)

In [182]: cor(a)
```

	Apple	Boeing
Apple	1.000000	0.872264
Boeing	0.872264	1.000000

4 Question 4

Créer un *data frame* avec les données HackerRank-Developer-Survey. Dans ces données, sont une série de réponse que les développeurs de HackerRank ont répondu suite à un sondage ayant pour but de comprendre les l'intérêt des femmes envers l'informatique.

```
In [1]: library(dplyr, warn.conflicts = F)

In [3]: values <- read.csv(paste(path,"HackerRank-Developer-Survey-2018-Values.csv",sep = ""), h

In [4]: head(values)
```

RespondentID	StartDate	EndDate	CountryNumeric2	q1AgeBeginCoding	q2Age
6464453728	10/19/17 11:51	10/20/17 12:05	South Korea	16 - 20 years old	18 - 24 years
6478031510	10/26/17 6:18	10/26/17 7:49	Ukraine	16 - 20 years old	25 - 34 years
6464392829	10/19/17 10:44	10/19/17 10:56	Malaysia	11 - 15 years old	12 - 18 years
6481629912	10/27/17 1:51	10/27/17 2:05	Curaçao	11 - 15 years old	12 - 18 years
6488385057	10/31/17 11:46	10/31/17 11:59		16 - 20 years old	25 - 34 years
6463843138	10/19/17 3:02	10/19/17 3:18	United States	41 - 50 years old	35 - 44 years

4.1 a)

En utilisanrtt le package `dplyr`, faites un petit tableau qui donne la proportion des hommes et des femmes dans ce *dataset*.

Utilisez la variable `q3Gender`

```
In [34]: values_2<-values %>%
             group_by(q3Gender) %>%
                 filter(q3Gender %in% c('Male','Female'))%>%
                     count()

In [35]: values_2$n<-(values_2$n/ sum(values_2$n)) * 100
         values_2
```

q3Gender	n
Female	16.55688
Male	83.44312

4.2 b)

En utilisanrtt le package `dplyr`, faites un tableau qui donne la proportion des hommes et des femmes en les séparant par le fait qu'ils soient étudiants ou non.

Utilisez les variables `q3Gender`, `is_student` et `q8Student`

```
In [36]: values$is_student <- ifelse(values$q8Student == '','Yes','No')
```

```
In [37]: values %>% group_by(q3Gender, is_student) %>%
             filter(q3Gender %in% c('Male','Female')) %>%
             count() %>%
             ungroup() %>%
             group_by(is_student) %>%
             mutate(n = (n/ sum(n)) * 100)
```

q3Gender	is_student	n
Female	No	20.82685
Female	Yes	13.55364
Male	No	79.17315
Male	Yes	86.44636

4.3 c)

Dressez un tableau qui donne le nombre de répondants par pays (utilisez la variable `CountryNumeric2`)

```
In [13]: values %>% group_by(CountryNumeric2) %>% count() %>% head()
```

CountryNumeric2	n
	3991
Afghanistan	3
Albania	8
Algeria	22
American Samoa	1
Andorra	1

4.4 d)

Faites un tableau qui donne le nombre de répondants en les classant par le diplôme obtenu.

Utilisez la variable `q4Education`

```
In [26]: values %>%
                     filter(!is.na(q4Education))%>%
                     group_by(q4Education)%>%
                     summarise(Total = n())%>%
                     arrange(desc(Total))  %>%
                     mutate(q4Education = reorder(q4Education,Total)) %>%
                     head(10)
```

q4Education	Total
College graduate	12010
Post graduate degree (Masters, PhD)	6030
Some college	2499
Some post graduate work (Masters, PhD)	2493
High school graduate	1289
Some high school	316
#NULL!	305
Vocational training (like bootcamp)	148

4.5 e)

Faites un tableau qui donne le nombre de développeurs par catégorie d'âge.
Utilisez la variable `q1AgeBeginCoding`

```
In [27]: values %>%
                    filter(!is.na(q1AgeBeginCoding)) %>%
                    group_by(q1AgeBeginCoding) %>%
                    summarise(Total = n()) %>%
                    arrange(desc(Total)) %>%
                    mutate(q1AgeBeginCoding = reorder(q1AgeBeginCoding,Total)) %>%
                    head(10)
```

q1AgeBeginCoding	Total
16 - 20 years old	14293
11 - 15 years old	5264
21 - 25 years old	3626
5 - 10 years old	933
26 - 30 years old	642
31 - 35 years old	193
36 - 40 years old	67
41 - 50 years old	34
#NULL!	30
50+ years or older	8

4_1_cours

October 15, 2018

Table of Contents

1 Les fonctions sur les caractères

Les actuaires doivent souvent travailler avec des bases de données qui contiennent des variables de type caractère. Il arrive qu'on veuille modifier le texte ou en extraire une partie. Dans cette section, nous verrons les fonctions les plus utilisées qui nous facilitent la tâche.

mais d'abord, créons quelques objets de type `text`:

```
In [1]: un_bonjour <- "Bonjour"
        salutations <- c("Bonjour", "Hi", "hey", "Salam")
        mot <- "world"
        question <- "Quel est le language de programmation préféré des actuaires?"
        reponse <- "Rrrrrrrr"
```

```
In [1]: un_bonjour<-"Bonjour"
        salutations<-c("Bonjour","Hi","Hey","Salam")
        mot<-"World"
```

```
question<-"Quel est le langage de programmation préféré des actuaires?"
réponse<-"Rrrrrrrrrr"
```

1.1 substr()

La fonction `substr(x, start, stop)` nous sert souvent lorsqu'on veut extraire une partie d'une chaîne de caractères.

Par exemple, appliquons une extraction du premier mot `Quel`, ce dernier commence à l'index #1 et termine à l'index #4

```
In [5]: question
```

'Quel est le language de programmation préféré des actuaires?'

```
In [8]: substr(question, 13, 20)
```

'language'

```
In [10]: substr(question, 1, 4)
```

'Quel'
pour le mot `langage`;

```
In [2]: substr(question, 13, 19)
```

'langage'

1.2 nchar()

Dans un plus long texte, nous ignorons la longueur du texte. Si l'on veut compter le nombre de caractères à l'intérieur d'une chaine de caractères, on utilise alors `nchar`

```
In [9]: nchar(question
```

60
On s'en sert souvent lorsqu'on ne veut pas fixer l'argument maximum de la fonction;

```
In [7]: substr(question, nchar(question)-9, nchar(question))
```

'actuaires?'
Si l'on applique cette fonction sur un vecteur, nous obtenons bien sûr un vecteur comme résultat

```
In [8]: substr(salutations, nchar(salutations)-2, nchar(salutations))
```

1. 'our' 2. 'Hi' 3. 'Hey' 4. 'lam'

1.3 Paste

Lorsqu'on veut concaténer des chaines de caractères, on utilise alors la fonction `paste`. **Passons nos salutations au monde!**

```
In [9]: paste(salutations, mot)
```

1. 'Bonjour World' 2. 'Hi World' 3. 'Hey World' 4. 'Salam World'

L'argument `sep` nous spécifie quel caractère nous utilisons afin de séperer ce qu'on veut concatener;

```
In [10]: paste(salutations, mot, sep=" -_- ")
```

1. 'Bonjour -_- World' 2. 'Hi -_- World' 3. 'Hey -_- World' 4. 'Salam -_- World'

Par défaut le séparateur est un espace. Si l'on voulait enlever l'espace, il faudrait alors le spécifier dans l'argument `sep=''`

```
In [11]: paste(salutations, mot, sep="")
```

1. 'BonjourWorld' 2. 'HiWorld' 3. 'HeyWorld' 4. 'SalamWorld'

Bien sûr, nous ne sommes pas limités à concaténer seulement deux mots, on peut ajouter directement d'autres caractères

```
In [12]: paste(salutations, mot, "!", sep=", ")
```

1. 'Bonjour, World, !' 2. 'Hi, World, !' 3. 'Hey, World, !' 4. 'Salam, World, !'

```
In [13]: paste(salutations, ", ",mot, "!", sep="")
```

1. 'Bonjour, World!' 2. 'Hi, World!' 3. 'Hey, World!' 4. 'Salam, World!'

Autre exemple qu'on pourrait souvent utiliser lorsqu'on veut nommer des colonnes dans une matrice ou un *data frame*. Par exemple, nous avons 10 colonnes qu'on voudrait appeler `cout_1` à `cout_10`

```
In [15]: paste("cout_", 1:10, sep="")
```

1. 'cout_1' 2. 'cout_2' 3. 'cout_3' 4. 'cout_4' 5. 'cout_5' 6. 'cout_6' 7. 'cout_7' 8. 'cout_8' 9. 'cout_9' 10. 'cout_10'

1.4 grep

Une fonction très utile lorsqu'on cherche l'index d'un élément à l'intérieur d'un vecteur d'éléments constitués de chaînes de caractères. Dans l'exemple suivant, nous cherchons l'index de l'occurrence de la lettre . Autrement dit, on demande de nous donner à quels numéros de l'élément, un caractère donné apparaît.

```
In [17]: grep("H", salutations)
```

1. 2 2. 3

Ça nous dit que le 1er ainsi que le 3e élément contienent les lettre `H`

Si l'on veut extraire ces éléments, nous ajoutons alors l'argument `value=TRUE`

```
In [18]: grep("H", salutations, value=TRUE)
```

1. 'Hi' 2. 'Hey'

On se rappelle que R contient une base de données qui contient l'information géographique de tous les états du pays de l'oncle Sam. Dans un autre exemple, cherchons les noms de ces états avec le mot NEW dedans.

```
In [19]: grep("New", state.name, value=TRUE)
```

1. 'New Hampshire' 2. 'New Jersey' 3. 'New Mexico' 4. 'New York'

1.5 strsplit

Cette fonction permet de séparer les chaines de caractères par un argument donné.

```
In [21]: strsplit(question, " ")
```

1. (a) 'Quel' (b) 'est' (c) 'le' (d) 'langage' (e) 'de' (f) 'programmation' (g) 'préféré' (h) 'des' (i) 'actuaires?'

1.6 sub

Lorsqu'on veut un caractère ou une chaine de caractères à l'intérieur d'une autre chaine de caractère, nous utilisons alors la fonction `sub`

```
In [22]: sub("actuaires", "geeks", question)
```

'Quel est le langage de programmation préféré des geeks?'

La fonction `sub` remplace seulement la première occurrence.

1.7 gsub

La fonction `gsub` remplace un caractère donné à toutes les occurrences;

```
In [23]: sub(" ", "-", question)
```

'Quel-est le langage de programmation préféré des actuaires?'

```
In [25]: gsub(" ", "-", question)
```

'Quel-est-le-langage-de-programmation-préféré-des-actuaires?'

les dates et heures

En pratique, il arrive souvent qu'on travaille avec les dates. Pensez seulement à la variable **date de naissance** qu'on retrouve dans toutes les bases de données.

La fonction qui nous donne la date courante dans R est la suivante:

```
In [26]: Sys.Date()
```

2018-03-23

Celle qui nous donne l'heure;

```
In [27]: Sys.time()
```

```
[1] "2018-03-23 15:43:01 EDT"
```

Assignons le temps actuel à la variable `time_1`

```
In [28]: time_1 <- Sys.time()
```

Bien évidemment, on peut appliquer des aditions et soustractions aux dates

```
In [29]: Sys.Date()+1 # pour la date de demain
```

2018-03-24

```
In [30]: Sys.Date()- 1 # pour la date d'hier
```

2018-03-22
Toutefois, les mêmes opérations se font par secondes lorsqu'on utilise `Sys.time()`

```
In [31]: time_1
```

```
[1] "2018-03-23 15:43:10 EDT"
```

```
In [32]: time_1+1
```

```
[1] "2018-03-23 15:43:11 EDT"
```

```
In [33]: time_1-60
```

```
[1] "2018-03-23 15:42:10 EDT"
```

Lorsqu'on veut soustraire une heure à notre temps, on soustrait alors 3600 secondes!

```
In [ ]: time_1+3600
```

Lorsque le résultat apparait à l'écran, à première vue, nous avons l'impression que nous obtenons un type caractère. Vérifions alors le type avec la fonction `class`

```
In [ ]: class(Sys.Date())
```

On voit bien que le type du résultat obtenu est bien `date`. Mai pour le temps, nous obtenons toute autre chose.

```
In [34]: class(Sys.time())
```

1. 'POSIXct' 2. 'POSIXt'
Ce qu'on obtient s'appelle un objet "POSIXct". On peut considérer cela comme numérique, et ce temps change numériquement en secondes depuis 1970.
Vérifions cela en forçant le format avec la fonction `as.numeric`

```
In [35]: as.numeric(Sys.time())
```

1521834992.99031
Alors que si nous forçons un format de type caractère;

```
In [36]: as.character(Sys.time())
```

'2018-03-23 15:56:46'
Nous obtenons presque le même résultat qu'au début, mais cette fois avec le type caractère.

1.8 weekdays

il est possible d'avoir le jour courant avec la fonction `weekdays()`
Créons d'abord un vecteur contenant les dates des deux dernières semaines.

```
In [37]: dates_2sem<-Sys.Date()- 1:10
         dates_2sem
```

1. 2018-03-22 2. 2018-03-21 3. 2018-03-20 4. 2018-03-19 5. 2018-03-18 6. 2018-03-17 7. 2018-03-16 8. 2018-03-15 9. 2018-03-14 10. 2018-03-13

```
In [38]: Sys.setlocale()
```

'en_CA.UTF-8/en_CA.UTF-8/en_CA.UTF-8/C/en_CA.UTF-8/en_CA.UTF-8'

```
In [39]: weekdays(dates_2sem)
```

1. 'Thursday' 2. 'Wednesday' 3. 'Tuesday' 4. 'Monday' 5. 'Sunday' 6. 'Saturday' 7. 'Friday' 8. 'Thursday' 9. 'Wednesday' 10. 'Tuesday'
Si l'on veut afficher en français, nous devons alors changer l'affichage local. Un package appelé `lubridate` nous permet de le faire facilement

```
In [41]: require("lubridate")
         Sys.setlocale(locale="fr_FR.UTF-8")
```

'fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/en_CA.UTF-8'

```
In [42]: weekdays(dates_2sem)
```

1. 'Jeudi' 2. 'Mercredi' 3. 'Mardi' 4. 'Lundi' 5. 'Dimanche' 6. 'Samedi' 7. 'Vendredi' 8. 'Jeudi' 9. 'Mercredi' 10. 'Mardi'

1.9 months

Et les mois avec la fonction `months`

```
In [43]: months(dates_2sem)
```

1. 'mars' 2. 'mars' 3. 'mars' 4. 'mars' 5. 'mars' 6. 'mars' 7. 'mars' 8. 'mars' 9. 'mars' 10. 'mars'

1.10 seq

On se rappelle de la fonction seq qui sert à générer une séquence d'objets incrémentés d'une unité quelconque. Utilisons cette fonction afin de générer toutes les dates du jour entre deux dates données.

Par exemple entre la date du premier cours et la date du dernier cours;

```
In [44]: seq(from = as.Date("06/09/17", "%d/%m/%y"), to = as.Date("13/12/17","%d/%m/%y"), by = "
```

1. 2017-09-06 2. 2017-09-07 3. 2017-09-08 4. 2017-09-09 5. 2017-09-10 6. 2017-09-11 7. 2017-09-12 8. 2017-09-13 9. 2017-09-14 10. 2017-09-15 11. 2017-09-16 12. 2017-09-17 13. 2017-09-18 14. 2017-09-19 15. 2017-09-20 16. 2017-09-21 17. 2017-09-22 18. 2017-09-23 19. 2017-09-24 20. 2017-09-25 21. 2017-09-26 22. 2017-09-27 23. 2017-09-28 24. 2017-09-29 25. 2017-09-30 26. 2017-10-01 27. 2017-10-02 28. 2017-10-03 29. 2017-10-04 30. 2017-10-05 31. 2017-10-06 32. 2017-10-07 33. 2017-10-08 34. 2017-10-09 35. 2017-10-10 36. 2017-10-11 37. 2017-10-12 38. 2017-10-13 39. 2017-10-14 40. 2017-10-15 41. 2017-10-16 42. 2017-10-17 43. 2017-10-18 44. 2017-10-19 45. 2017-10-20 46. 2017-10-21 47. 2017-10-22 48. 2017-10-23 49. 2017-10-24 50. 2017-10-25 51. 2017-10-26 52. 2017-10-27 53. 2017-10-28 54. 2017-10-29 55. 2017-10-30 56. 2017-10-31 57. 2017-11-01 58. 2017-11-02 59. 2017-11-03 60. 2017-11-04 61. 2017-11-05 62. 2017-11-06 63. 2017-11-07 64. 2017-11-08 65. 2017-11-09 66. 2017-11-10 67. 2017-11-11 68. 2017-11-12 69. 2017-11-13 70. 2017-11-14 71. 2017-11-15 72. 2017-11-16 73. 2017-11-17 74. 2017-11-18 75. 2017-11-19 76. 2017-11-20 77. 2017-11-21 78. 2017-11-22 79. 2017-11-23 80. 2017-11-24 81. 2017-11-25 82. 2017-11-26 83. 2017-11-27 84. 2017-11-28 85. 2017-11-29 86. 2017-11-30 87. 2017-12-01 88. 2017-12-02 89. 2017-12-03 90. 2017-12-04 91. 2017-12-05 92. 2017-12-06 93. 2017-12-07 94. 2017-12-08 95. 2017-12-09 96. 2017-12-10 97. 2017-12-11 98. 2017-12-12 99. 2017-12-13

Et si nous voulions toutes les dates de chaque cours (à chaque semaine)

```
In [45]: seq(from = as.Date("06/09/17", "%d/%m/%y"), to = as.Date("13/12/17","%d/%m/%y"), by = "
```

1. 2017-09-06 2. 2017-09-13 3. 2017-09-20 4. 2017-09-27 5. 2017-10-04 6. 2017-10-11 7. 2017-10-18 8. 2017-10-25 9. 2017-11-01 10. 2017-11-08 11. 2017-11-15 12. 2017-11-22 13. 2017-11-29 14. 2017-12-06 15. 2017-12-13

Faites la même chose pour les dates de démo par exemple, sachant la première démo avait commencé le 19 septembre

```
In [46]: seq(from = as.Date("19/09/17", "%d/%m/%y"), to = as.Date("12/12/17","%d/%m/%y"), by = "
```

1. 2017-09-19 2. 2017-09-26 3. 2017-10-03 4. 2017-10-10 5. 2017-10-17 6. 2017-10-24 7. 2017-10-31 8. 2017-11-07 9. 2017-11-14 10. 2017-11-21 11. 2017-11-28 12. 2017-12-05 13. 2017-12-12

1.11 difftime

Si on veut savoir combien de temps s'est passé entre deux variables (d'heure) données, on deux options;

La première consiste à simplement soustraire la première variable de la deuxième;

```
In [47]: time_2<-Sys.time()

In [48]: time_2 - time_1
```

```
Time difference of 18.66099 mins
```

L'aute option est d'utiliser la fonctin `difftime`

```
In [51]: difftime(time_1, time_2)
```

```
Time difference of -18.66099 mins
```

```
In [52]: difftime(time_2, time_1)
```

```
Time difference of 18.66099 mins
```

On peut ajouter l'argument `units` afin d'afficher les unités voulues

```
In [53]: difftime(time_2, time_1, units='sec')
```

```
Time difference of 1119.659 secs
```

Si l'on veut seulement le résultat en chiffre, on peut le transformer en valeur numérique tel que nous avons appris

```
In [54]: as.numeric(difftime(time_2, time_1, units='sec'))
```

1119.65947914124

Toutes ces fonctions de dates sont importantes lorsqu'on veut travailler sur des problèmes traitant les séries temporelles par exemple

2 Manipulation de fichiers

Il arrive souvent qu'on manipule des fichiers directement avec R, soit pour tirer une information quelconque sur les fichiers ou en créer d'autres fichiers

2.1 File.exists

On utilise la fonction file.exists() lorsqu'on veut faire un test booléen sur l'existence d'un fichier (peu importe le type de fichier).

Nous nous servons de cette fonction lorsque nous voulons modifier (ou créer) un fichier. Donc avons la modification (ou la création), nous validons d'abord l'existence de ce dernier.

```
In [16]: file.exists("nbmerge.py")
```

TRUE

```
In [14]: file.exists("cars_info_test-11111.csv")
```

FALSE

2.2 file.rename

On peut renommer le ficher avec;

```
In [ ]: file.rename("cars_info_test.csv", "cars_info_test2.csv")
```

2.3 file.create

On peur créer un tout nouveau fichier

```
In [ ]: file.create("vide.txt")
```

2.4 list.files

On peut lister tous les fichiers contenus dans le répertoire courant

```
In [15]: list.files()
```

1. 'cours_10_1.ipynb' 2. 'cours_10_2.ipynb' 3. 'cours_10.ipynb' 4. 'cours_12.ipynb' 5. 'cours_13.ipynb' 6. 'cours_8_1.ipynb' 7. 'cours_8_2.ipynb' 8. 'cours_9_1.ipynb' 9. 'cours_9_2.ipynb' 10. 'cours_9.ipynb' 11. 'Exercices_8_solutions.ipynb' 12. 'Exercices_8.ipynb' 13. 'Exercices_9.ipynb' 14. 'Exercices_R_C9.ipynb' 15. 'images' 16. 'nbmerge.py'

```
In [ ]: file.info("cars_info_test2.csv")
```

```
In [ ]: file.info("cars_info_test2.csv")$size
```

4_1_cours

October 15, 2018

Table of Contents

1 Les fonctions sur les caractères

Les actuaires doivent souvent travailler avec des bases de données qui contiennent des variables de type caractère. Il arrive qu'on veuille modifier le texte ou en extraire une partie. Dans cette section, nous verrons les fonctions les plus utilisées qui nous facilitent la tâche.

mais d'abord, créons quelques objets de type `text`:

```
In [1]: un_bonjour <- "Bonjour"
        salutations <- c("Bonjour", "Hi", "hey", "Salam")
        mot <- "world"
        question <- "Quel est le language de programmation préféré des actuaires?"
        reponse <- "Rrrrrrrr"
```

```
In [1]: un_bonjour<-"Bonjour"
        salutations<-c("Bonjour","Hi","Hey","Salam")
        mot<-"World"
```

```
question<-"Quel est le langage de programmation préféré des actuaires?"
réponse<-"Rrrrrrrrrr"
```

1.1 substr()

La fonction `substr(x, start, stop)` nous sert souvent lorsqu'on veut extraire une partie d'une chaîne de caractères.

Par exemple, appliquons une extraction du premier mot `Quel`, ce dernier commence à l'index #1 et termine à l'index #4

```
In [5]: question
```

'Quel est le language de programmation préféré des actuaires?'

```
In [8]: substr(question, 13, 20)
```

'language'

```
In [10]: substr(question, 1, 4)
```

'Quel'
pour le mot `langage`;

```
In [2]: substr(question, 13, 19)
```

'langage'

1.2 nchar()

Dans un plus long texte, nous ignorons la longueur du texte. Si l'on veut compter le nombre de caractères à l'intérieur d'une chaine de caractères, on utilise alors `nchar`

```
In [9]: nchar(question
```

60
On s'en sert souvent lorsqu'on ne veut pas fixer l'argument maximum de la fonction;

```
In [7]: substr(question, nchar(question)-9, nchar(question))
```

'actuaires?'
Si l'on applique cette fonction sur un vecteur, nous obtenons bien sûr un vecteur comme résultat

```
In [8]: substr(salutations, nchar(salutations)-2, nchar(salutations))
```

1. 'our' 2. 'Hi' 3. 'Hey' 4. 'lam'

1.3 Paste

Lorsqu'on veut concaténer des chaines de caractères, on utilise alors la fonction `paste`. **Passons nos salutations au monde!**

```
In [9]: paste(salutations, mot)
```

1. 'Bonjour World' 2. 'Hi World' 3. 'Hey World' 4. 'Salam World'

L'argument `sep` nous spécifie quel caractère nous utilisons afin de séperer ce qu'on veut concatener;

```
In [10]: paste(salutations, mot, sep=" -_- ")
```

1. 'Bonjour -_- World' 2. 'Hi -_- World' 3. 'Hey -_- World' 4. 'Salam -_- World'

Par défaut le séparateur est un espace. Si l'on voulait enlever l'espace, il faudrait alors le spécifier dans l'argument `sep=''`

```
In [11]: paste(salutations, mot, sep="")
```

1. 'BonjourWorld' 2. 'HiWorld' 3. 'HeyWorld' 4. 'SalamWorld'

Bien sûr, nous ne sommes pas limités à concaténer seulement deux mots, on peut ajouter directement d'autres caractères

```
In [12]: paste(salutations, mot, "!", sep=", ")
```

1. 'Bonjour, World, !' 2. 'Hi, World, !' 3. 'Hey, World, !' 4. 'Salam, World, !'

```
In [13]: paste(salutations, ", ",mot, "!", sep="")
```

1. 'Bonjour, World!' 2. 'Hi, World!' 3. 'Hey, World!' 4. 'Salam, World!'

Autre exemple qu'on pourrait souvent utiliser lorsqu'on veut nommer des colonnes dans une matrice ou un *data frame*. Par exemple, nous avons 10 colonnes qu'on voudrait appeler `cout_1` à `cout_10`

```
In [15]: paste("cout_", 1:10, sep="")
```

1. 'cout_1' 2. 'cout_2' 3. 'cout_3' 4. 'cout_4' 5. 'cout_5' 6. 'cout_6' 7. 'cout_7' 8. 'cout_8' 9. 'cout_9' 10. 'cout_10'

1.4 grep

Une fonction très utile lorsqu'on cherche l'index d'un élément à l'intérieur d'un vecteur d'éléments constitués de chaînes de caractères. Dans l'exemple suivant, nous cherchons l'index de l'occurrence de la lettre . Autrement dit, on demande de nous donner à quels numéros de l'élément, un caractère donné apparaît.

```
In [17]: grep("H", salutations)
```

1. 2 2. 3

Ça nous dit que le 1er ainsi que le 3e élément contienent les lettre `H`

Si l'on veut extraire ces éléments, nous ajoutons alors l'argument `value=TRUE`

```
In [18]: grep("H", salutations, value=TRUE)
```

1. 'Hi' 2. 'Hey'

On se rappelle que R contient une base de données qui contient l'information géographique de tous les états du pays de l'oncle Sam. Dans un autre exemple, cherchons les noms de ces états avec le mot NEW dedans.

```
In [19]: grep("New", state.name, value=TRUE)
```

1. 'New Hampshire' 2. 'New Jersey' 3. 'New Mexico' 4. 'New York'

1.5 strsplit

Cette fonction permet de séparer les chaines de caractères par un argument donné.

```
In [21]: strsplit(question, " ")
```

1. (a) 'Quel' (b) 'est' (c) 'le' (d) 'langage' (e) 'de' (f) 'programmation' (g) 'préféré' (h) 'des' (i) 'actuaires?'

1.6 sub

Lorsqu'on veut un caractère ou une chaine de caractères à l'intérieur d'une autre chaine de caractère, nous utilisons alors la fonction `sub`

```
In [22]: sub("actuaires", "geeks", question)
```

'Quel est le langage de programmation préféré des geeks?'

La fonction `sub` remplace seulement la première occurrence.

1.7 gsub

La fonction `gsub` remplace un caractère donné à toutes les occurrences;

```
In [23]: sub(" ", "-", question)
```

'Quel-est le langage de programmation préféré des actuaires?'

```
In [25]: gsub(" ", "-", question)
```

'Quel-est-le-langage-de-programmation-préféré-des-actuaires?'

les dates et heures

En pratique, il arrive souvent qu'on travaille avec les dates. Pensez seulement à la variable **date de naissance** qu'on retrouve dans toutes les bases de données.

La fonction qui nous donne la date courante dans R est la suivante:

```
In [26]: Sys.Date()
```

2018-03-23

Celle qui nous donne l'heure;

```
In [27]: Sys.time()
```

```
[1] "2018-03-23 15:43:01 EDT"
```

Assignons le temps actuel à la variable `time_1`

```
In [28]: time_1 <- Sys.time()
```

Bien évidemment, on peut appliquer des aditions et soustractions aux dates

```
In [29]: Sys.Date()+1 # pour la date de demain
```

2018-03-24

```
In [30]: Sys.Date()- 1 # pour la date d'hier
```

2018-03-22
Toutefois, les mêmes opérations se font par secondes lorsqu'on utilise `Sys.time()`

```
In [31]: time_1
```

```
[1] "2018-03-23 15:43:10 EDT"
```

```
In [32]: time_1+1
```

```
[1] "2018-03-23 15:43:11 EDT"
```

```
In [33]: time_1-60
```

```
[1] "2018-03-23 15:42:10 EDT"
```

Lorsqu'on veut soustraire une heure à notre temps, on soustrait alors 3600 secondes!

```
In [ ]: time_1+3600
```

Lorsque le résultat apparait à l'écran, à première vue, nous avons l'impression que nous obtenons un type caractère. Vérifions alors le type avec la fonction `class`

```
In [ ]: class(Sys.Date())
```

On voit bien que le type du résultat obtenu est bien `date`. Mai pour le temps, nous obtenons toute autre chose.

```
In [34]: class(Sys.time())
```

1. 'POSIXct' 2. 'POSIXt'
Ce qu'on obtient s'appelle un objet "POSIXct". On peut considérer cela comme numérique, et ce temps change numériquement en secondes depuis 1970.
Vérifions cela en forçant le format avec la fonction `as.numeric`

```
In [35]: as.numeric(Sys.time())
```

1521834992.99031
Alors que si nous forçons un format de type caractère;

```
In [36]: as.character(Sys.time())
```

'2018-03-23 15:56:46'
Nous obtenons presque le même résultat qu'au début, mais cette fois avec le type caractère.

1.8 weekdays

il est possible d'avoir le jour courant avec la fonction `weekdays()`
Créons d'abord un vecteur contenant les dates des deux dernières semaines.

```
In [37]: dates_2sem<-Sys.Date()- 1:10
         dates_2sem
```

1. 2018-03-22 2. 2018-03-21 3. 2018-03-20 4. 2018-03-19 5. 2018-03-18 6. 2018-03-17 7. 2018-03-16 8. 2018-03-15 9. 2018-03-14 10. 2018-03-13

```
In [38]: Sys.setlocale()
```

'en_CA.UTF-8/en_CA.UTF-8/en_CA.UTF-8/C/en_CA.UTF-8/en_CA.UTF-8'

```
In [39]: weekdays(dates_2sem)
```

1. 'Thursday' 2. 'Wednesday' 3. 'Tuesday' 4. 'Monday' 5. 'Sunday' 6. 'Saturday' 7. 'Friday' 8. 'Thursday' 9. 'Wednesday' 10. 'Tuesday'
Si l'on veut afficher en français, nous devons alors changer l'affichage local. Un package appelé `lubridate` nous permet de le faire facilement

```
In [41]: require("lubridate")
         Sys.setlocale(locale="fr_FR.UTF-8")
```

'fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/en_CA.UTF-8'

```
In [42]: weekdays(dates_2sem)
```

1. 'Jeudi' 2. 'Mercredi' 3. 'Mardi' 4. 'Lundi' 5. 'Dimanche' 6. 'Samedi' 7. 'Vendredi' 8. 'Jeudi' 9. 'Mercredi' 10. 'Mardi'

1.9 months

Et les mois avec la fonction `months`

```
In [43]: months(dates_2sem)
```

1. 'mars' 2. 'mars' 3. 'mars' 4. 'mars' 5. 'mars' 6. 'mars' 7. 'mars' 8. 'mars' 9. 'mars' 10. 'mars'

1.10 seq

On se rappelle de la fonction seq qui sert à générer une séquence d'objets incrémentés d'une unité quelconque. Utilisons cette fonction afin de générer toutes les dates du jour entre deux dates données.

Par exemple entre la date du premier cours et la date du dernier cours;

```
In [44]: seq(from = as.Date("06/09/17", "%d/%m/%y"), to = as.Date("13/12/17","%d/%m/%y"), by = "
```

1. 2017-09-06 2. 2017-09-07 3. 2017-09-08 4. 2017-09-09 5. 2017-09-10 6. 2017-09-11 7. 2017-09-12 8. 2017-09-13 9. 2017-09-14 10. 2017-09-15 11. 2017-09-16 12. 2017-09-17 13. 2017-09-18 14. 2017-09-19 15. 2017-09-20 16. 2017-09-21 17. 2017-09-22 18. 2017-09-23 19. 2017-09-24 20. 2017-09-25 21. 2017-09-26 22. 2017-09-27 23. 2017-09-28 24. 2017-09-29 25. 2017-09-30 26. 2017-10-01 27. 2017-10-02 28. 2017-10-03 29. 2017-10-04 30. 2017-10-05 31. 2017-10-06 32. 2017-10-07 33. 2017-10-08 34. 2017-10-09 35. 2017-10-10 36. 2017-10-11 37. 2017-10-12 38. 2017-10-13 39. 2017-10-14 40. 2017-10-15 41. 2017-10-16 42. 2017-10-17 43. 2017-10-18 44. 2017-10-19 45. 2017-10-20 46. 2017-10-21 47. 2017-10-22 48. 2017-10-23 49. 2017-10-24 50. 2017-10-25 51. 2017-10-26 52. 2017-10-27 53. 2017-10-28 54. 2017-10-29 55. 2017-10-30 56. 2017-10-31 57. 2017-11-01 58. 2017-11-02 59. 2017-11-03 60. 2017-11-04 61. 2017-11-05 62. 2017-11-06 63. 2017-11-07 64. 2017-11-08 65. 2017-11-09 66. 2017-11-10 67. 2017-11-11 68. 2017-11-12 69. 2017-11-13 70. 2017-11-14 71. 2017-11-15 72. 2017-11-16 73. 2017-11-17 74. 2017-11-18 75. 2017-11-19 76. 2017-11-20 77. 2017-11-21 78. 2017-11-22 79. 2017-11-23 80. 2017-11-24 81. 2017-11-25 82. 2017-11-26 83. 2017-11-27 84. 2017-11-28 85. 2017-11-29 86. 2017-11-30 87. 2017-12-01 88. 2017-12-02 89. 2017-12-03 90. 2017-12-04 91. 2017-12-05 92. 2017-12-06 93. 2017-12-07 94. 2017-12-08 95. 2017-12-09 96. 2017-12-10 97. 2017-12-11 98. 2017-12-12 99. 2017-12-13

Et si nous voulions toutes les dates de chaque cours (à chaque semaine)

```
In [45]: seq(from = as.Date("06/09/17", "%d/%m/%y"), to = as.Date("13/12/17","%d/%m/%y"), by = "
```

1. 2017-09-06 2. 2017-09-13 3. 2017-09-20 4. 2017-09-27 5. 2017-10-04 6. 2017-10-11 7. 2017-10-18 8. 2017-10-25 9. 2017-11-01 10. 2017-11-08 11. 2017-11-15 12. 2017-11-22 13. 2017-11-29 14. 2017-12-06 15. 2017-12-13

Faites la même chose pour les dates de démo par exemple, sachant la première démo avait commencé le 19 septembre

```
In [46]: seq(from = as.Date("19/09/17", "%d/%m/%y"), to = as.Date("12/12/17","%d/%m/%y"), by = "
```

1. 2017-09-19 2. 2017-09-26 3. 2017-10-03 4. 2017-10-10 5. 2017-10-17 6. 2017-10-24 7. 2017-10-31 8. 2017-11-07 9. 2017-11-14 10. 2017-11-21 11. 2017-11-28 12. 2017-12-05 13. 2017-12-12

1.11 difftime

Si on veut savoir combien de temps s'est passé entre deux variables (d'heure) données, on deux options;

La première consiste à simplement soustraire la première variable de la deuxième;

```
In [47]: time_2<-Sys.time()

In [48]: time_2 - time_1
```

```
Time difference of 18.66099 mins
```

L'aute option est d'utiliser la fonctin `difftime`

```
In [51]: difftime(time_1, time_2)
```

```
Time difference of -18.66099 mins
```

```
In [52]: difftime(time_2, time_1)
```

```
Time difference of 18.66099 mins
```

On peut ajouter l'argument `units` afin d'afficher les unités voulues

```
In [53]: difftime(time_2, time_1, units='sec')
```

```
Time difference of 1119.659 secs
```

Si l'on veut seulement le résultat en chiffre, on peut le transformer en valeur numérique tel que nous avons appris

```
In [54]: as.numeric(difftime(time_2, time_1, units='sec'))
```

1119.65947914124

Toutes ces fonctions de dates sont importantes lorsqu'on veut travailler sur des problèmes traitant les séries temporelles par exemple

2 Manipulation de fichiers

Il arrive souvent qu'on manipule des fichiers directement avec R, soit pour tirer une information quelconque sur les fichiers ou en créer d'autres fichiers

2.1 File.exists

On utilise la fonction file.exists() lorsqu'on veut faire un test booléen sur l'existence d'un fichier (peu importe le type de fichier).

Nous nous servons de cette fonction lorsque nous voulons modifier (ou créer) un fichier. Donc avons la modification (ou la création), nous validons d'abord l'existence de ce dernier.

```
In [16]: file.exists("nbmerge.py")
```

TRUE

```
In [14]: file.exists("cars_info_test-11111.csv")
```

FALSE

2.2 file.rename

On peut renommer le ficher avec;

```
In [ ]: file.rename("cars_info_test.csv", "cars_info_test2.csv")
```

2.3 file.create

On peur créer un tout nouveau fichier

```
In [ ]: file.create("vide.txt")
```

2.4 list.files

On peut lister tous les fichiers contenus dans le répertoire courant

```
In [15]: list.files()
```

1. 'cours_10_1.ipynb' 2. 'cours_10_2.ipynb' 3. 'cours_10.ipynb' 4. 'cours_12.ipynb' 5. 'cours_13.ipynb' 6. 'cours_8_1.ipynb' 7. 'cours_8_2.ipynb' 8. 'cours_9_1.ipynb' 9. 'cours_9_2.ipynb' 10. 'cours_9.ipynb' 11. 'Exercices_8_solutions.ipynb' 12. 'Exercices_8.ipynb' 13. 'Exercices_9.ipynb' 14. 'Exercices_R_C9.ipynb' 15. 'images' 16. 'nbmerge.py'

```
In [ ]: file.info("cars_info_test2.csv")
```

```
In [ ]: file.info("cars_info_test2.csv")$size
```

4_3_Exercices_Solutions

October 15, 2018

```
In [21]: library(stringr)
```

1 no 1

Créer n'importe quuel fichier sur votre répertoire de travail et vérifier s'il existe.
Note ce numéros sert juste à pratiquer `file.create` et `file.exists`.

```
In [1]: file.create("vide.txt")
```

TRUE

```
In [3]: file.exists("vide.txt")
```

TRUE

```
In [4]: file.exists("vide.csv")
```

FALSE

2 no 2

Créez une variable qui représente votre date de naissance appelée date_naiss.

```
In [74]: date_naiss<-ymd('1990-10-16')
         date_naiss
```

1990-10-16
Quelle est la date d'aujourd'hui?

```
In [75]: auj=Sys.Date()
         auj
```

2018-09-30
Quelle jour de semaine vous êtes né?

```
In [ ]: weekdays(date_naiss)
```

Affichez le résultat en espagnole

```
In [ ]: Sys.setlocale(locale="es_ES.UTF-8")
```

```
In [ ]: weekdays(date_naiss)
```

Remettre votre ordi en Français

```
In [ ]: Sys.setlocale(locale="fr_FR.UTF-8")
```

3 no 3

Afin de remercier vos collègues de travail de leur présence à votre première formation que vous avez donné, vous décidez de leur envoyer un message de remerciemenets. Toutefois, vous voulez ajouter votre touche personnelle en écrivant un émail à chaucun de vos collègues. en mentionnant leur prénoms à l'entête de du message.

Votre message ressmble à ceci: ___

```
Chèr "prénom",

Un petit mot pour vous dire merci. Merci d'avoir contribué à la réussite de ma première présenta

À bientôt!
```

Toutefois, vous vous rendez compte qu'il faut accorder le mot "chèr" au masculin ou au féminin selon le sexe de votre collègue. Vous ne voulez certainement pas perdre votre temps à le faire manuellement, alors vous confiez la tâche à votre bel ordinateur qui comprends bien le langage R!

```
In [52]: a<-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/coll%C3%A8gues.csv"
```

```
In [53]: tail(a)
```

	birthdate	PrenomNom	sex
66	1983-08-04	Sabine Lejeune	F
67	1995-05-24	Timothée Marchand	M
68	1990-06-05	Sylvie Roux-Delattre	F
69	1985-02-14	Patrick-Daniel Barre	M
70	1981-02-20	Laurent Lacombe-Lecomte	M
71	1990-03-08	Marcel David	M

```
In [54]: library(stringr)
```

```
In [55]: a$prenom<- str_split_fixed(a$PrenomNom, " ", 2)[,1]
         a$nom<- str_split_fixed(a$PrenomNom, " ", 2)[,2]
```

```
In [56]: head(a)
```

birthdate	PrenomNom	sex	prenom	nom
1985-07-02	Anaïs-Adrienne Launay	F	Anaïs-Adrienne	Launay
1994-11-11	Benjamin Delmas de la Gros	M	Benjamin	Delmas de la Gros
1991-03-20	Laurence Lucas	F	Laurence	Lucas
1993-12-01	Philippine Laurent	F	Philippine	Laurent
1984-03-22	Louise Paris	F	Louise	Paris
1976-07-29	André Joly	M	André	Joly

```
In [57]: a$forme_mas_fem<-ifelse(a$sex=="F", paste("Chère", a$prenom, ","), paste("Chèr", a$pren
```

```
In [58]: a$message<-paste(a$forme,
                          "Un petit mot pour vous dire merci. Merci d'avoir contribué à la réuss
                         "À bientôt!",sep="\n")
```

```
In [59]: print(head(a$message))
```

```
[1] "Chère Anaïs-Adrienne ,\nUn petit mot pour vous dire merci. Merci d'avoir contribué à la réu
[2] "Chèr Benjamin ,\nUn petit mot pour vous dire merci. Merci d'avoir contribué à la réussite d
[3] "Chère Laurence ,\nUn petit mot pour vous dire merci. Merci d'avoir contribué à la réussite
[4] "Chère Philippine ,\nUn petit mot pour vous dire merci. Merci d'avoir contribué à la réussit
[5] "Chère Louise ,\nUn petit mot pour vous dire merci. Merci d'avoir contribué à la réussite de
[6] "Chèr André ,\nUn petit mot pour vous dire merci. Merci d'avoir contribué à la réussite de m
```

Lorsqu'on affiche le message qui doit être envoyé à chaque collègue, on remarque: * Les charactères `\n` permettent de créer un saut de ligne dans le paragraphe lorsque ce dernier est écrirt dans un fichier `.txt` * On remarque aussi que nous nous adresson bien à nos collègues féminis par `Chère` (Chère Anaïs-Adrienne) alors que les collègues masculins par `Chèr` (Chèr André).

4 no 4

À partir du fichier csv précédent, calculez l'âge pour chaucne des lignes avec les deux méthodes décrite ci-dessous.

4.1 calculé

Dans la plupart du temps, l'âge peut se calculer comme suit:

$$\frac{(\text{date d'aujourd'hui}) - (\text{date de naissance})}{365.25}$$

Nous avons appris à créer des fonctions, à partir de la formule précédente, créer une fonction appelé `age` qui calclule l'âge d'une personne. Cette fonction prends comme argument la date de naissance et la date courante. Par défaut, la date de naissance est 1970-01-01 et la date courante est la date d'aujourd'hui

```
In [30]: library(lubridate)
```

```
In [69]: date_naiss=ymd('1992-01-01')
         date_cour=Sys.Date()
```

```
In [45]: age<-function(date_naiss=ymd('1970-01-01'), date_cour=Sys.Date()){
             return(as.numeric((date_cour-date_naiss)/365.25))
         }
```

```
In [46]: age(date_naiss, date_cour)
```

26.7460643394935

Améliorez notre fonction âge avec une validation que les arguments entrés sont bien des dates D'abbord vérifions la condition;

```
In [47]: is.Date(date_naiss) && is.Date(date_cour)
```

TRUE
ensuite nous pouvons insérer

```
In [48]: age<-function(date_naiss=ymd('1970-01-01'), date_cour=Sys.Date()){
             if (is.Date(date_naiss) && is.Date(date_cour)){
             return(as.numeric((date_cour-date_naiss)/365.25))
                 }
         }
```

```
In [49]: age(date_naiss, date_cour)
```

26.7460643394935
À partir des données de l'exercice précédent (no3), créez une variole appelée age_collegues qui calcule l'âge de vos collègues avec votre nouvelle fonction que vous venez de créer:

```
In [63]: a$age_collegues<-round(age(date_cour =Sys.Date() , date_naiss = as.Date(a$birthdate)))
```

S'il est plus grand que vous de 3 ans et plus, vous le vouvoyez. Sinon vous ne vous gênez pas de le tutoyer. Puisque nous avons déjà le message, nous pouvons juste modifier ce message en change le `vous` par `tu` lorsque `age_collegues-mon_age>=3` avec la fonction `gsub`.

```
In [100]: mon_age=age(date_naiss,auj)
          round(mon_age,0)
```

28

```
In [98]: a$message_poli <- ifelse(a$age_collegues-round(mon_age,0)<3,
                a$message_poli <- gsub(x=a$message,pattern = "vous", replacement = "te"),
                a$message_poli <- a$message)
```

```
In [99]: head(a,2)
```

birthdate	PrenomNom	sex	prenom	nom	forme_mas_fem
1985-07-02	Anaïs-Adrienne Launay	F	Anaïs-Adrienne	Launay	Chère Anaïs-Adrie
1994-11-11	Benjamin Delmas de la Gros	M	Benjamin	Delmas de la Gros	Chèr Benjamin ,

Version dplyr:

```
In [107]: library(dplyr)
```

```
In [114]: a %>% mutate(message_poli = ifelse(age_collegues-round(mon_age,0)<3,
                                        gsub(x=message,pattern = "vous", replacement = "te"),
                                        message)) %>% head(2)
```

birthdate	PrenomNom	sex	prenom	nom	forme_mas_fem
1985-07-02	Anaïs-Adrienne Launay	F	Anaïs-Adrienne	Launay	Chère Anaïs-Adrie
1994-11-11	Benjamin Delmas de la Gros	M	Benjamin	Delmas de la Gros	Chèr Benjamin ,

Créerons tous ces messages dans des fichiers `.txt`. Le nom de chaque fichier sera le nom de votre collègue et son prénom comme suit `prenon_non.txt`. Tous ces fichiers se trouveront un répertoire appelé `lettres`.

```
In [175]: for (i in 1:length(a)){
              nomfichier <- paste("/Users/nour/Downloads/lettres/",
                                  paste(gsub(a$prenom[i],pattern = " ", replacement = ""),
                                        gsub(a$nom[i],pattern = " ", replacement = "")
                                        , sep="_")
                                  , ".txt",sep="")
              print(nomfichier)
              writeLines(text=a$message_poli[i], con=nomfichier)
          }
```

```
[1] "/Users/nour/Downloads/lettres/Anaïs-Adrienne_Launay.txt"
[1] "/Users/nour/Downloads/lettres/Benjamin_DelmasdelaGros.txt"
[1] "/Users/nour/Downloads/lettres/Laurence_Lucas.txt"
[1] "/Users/nour/Downloads/lettres/Philippine_Laurent.txt"
[1] "/Users/nour/Downloads/lettres/Louise_Paris.txt"
[1] "/Users/nour/Downloads/lettres/André_Joly.txt"
[1] "/Users/nour/Downloads/lettres/Léon_Gosselin.txt"
[1] "/Users/nour/Downloads/lettres/Nicolas_Normand-Fontaine.txt"
[1] "/Users/nour/Downloads/lettres/Camille_Charrier.txt"
[1] "/Users/nour/Downloads/lettres/Julien_Legrand.txt"
```

4.2 lubridate

Dans le package `lubridate`, il existe une fonction qui calcule l'intervale entre deux date, utilisons cette fonction pour valider le calcul de notre fonction age que nous avons crée précédemment

```
In [ ]: a$age_lubri<-round(interval(start = as.Date(a$birthdate), end =Sys.Date() ) /
                duration(num = 1, units = "years"),0)
```

```
In [ ]: head(a)
```

5 for loops

5.1 numéro1

En utilisant les variables suivantes: x=1 y=40 i=c(1:10) Pour cet exercice, écrivez une boucle for () qui incrémente x par trois et diminue y par deux, pour chaque i.

```
In [ ]: x=1
        y=40
        i=c(1:10)
        for (j in i)
        {x=x+3
         y=y-2
        print(paste(x, y, sep="_"))
        }
```

5.2 Numéro 2

Soit les variables:

```
In [ ]: a=15:10
        b=20:15
```

Pour cet exercice, écrivez une boucle `while()` qui calcule un vecteur x = 225 224 221 216 209 200, tel que

```
x[1]=a[1]*b[6]
x[2]=a[2]*b[5]
x[3]=a[3]*b[5]
.
.
x[6]=a[6]*b[1]
```

```
In [ ]: x=c()
        i=1
        j=6
        while (i<7)
        {
          x[i]=a[i]*b[j]
          i=i+1
          j=j-1
        }
        x
```

5.3 numero 3

en utilisant la variable suivante:

a=1:10

Pour cet exercice, écrivez une boucle `while ()` qui calcule un vecteur x = 1 3 6 10 15 21 28 36 45 55, tel que

```
x[1]=a[1]
x[2]=a[1]+a[2]
x[3]=a[1]+a[2]+a[3]
.
.
```

```
In [ ]: a=1:10

        i=1
        x=c()
        while (i<=10)
          {
          x[i]=sum(a[1:i])
          i=i+1
          }
        x
```

5.4 Numéro 4

en utilisant la variable suivante:

```
i=10
x=10
```

Pour cet exercice, écrivez une boucle `repeat()` qui diminue en calculant `x = x / i` jusqu'à ce que `i = 0`.

```
In [5]: i=10
        x=10

        repeat
          {x=x/i
          print(x)
          i=i-1
          if (i==0)
            break
          }
```

```
[1] 1
[1] 0.1111111
[1] 0.01388889
[1] 0.001984127
[1] 0.0003306878
[1] 6.613757e-05
[1] 1.653439e-05
[1] 5.511464e-06
[1] 2.755732e-06
[1] 2.755732e-06
```

5.5 Numéro 5

Puisque vous êtes en fin de session, vous voulez faire un plan d'étude pour bien réussir vos examens. Ecrivez une boucle repeat qui permet d'afficher toutes les dates entre aujourd'hui et le jour de votre dernier examen:

```
In [8]: auj<-as.Date(Sys.Date(),"%d/%m/%Y")
        dernier_exam<-as.Date("18/04/2018","%d/%m/%Y")
        repeat
        {
        auj<-auj+1
            print(auj)
        if (auj==dernier_exam)
          break
        }
```

```
[1] "2018-03-29"
[1] "2018-03-30"
[1] "2018-03-31"
[1] "2018-04-01"
[1] "2018-04-02"
[1] "2018-04-03"
[1] "2018-04-04"
[1] "2018-04-05"
[1] "2018-04-06"
[1] "2018-04-07"
[1] "2018-04-08"
[1] "2018-04-09"
[1] "2018-04-10"
[1] "2018-04-11"
[1] "2018-04-12"
[1] "2018-04-13"
[1] "2018-04-14"
[1] "2018-04-15"
[1] "2018-04-16"
[1] "2018-04-17"
[1] "2018-04-18"
```

5_1_cours

October 15, 2018

1 Introduction

Visualiser les données est une étape cruciale lorsqu'on analyse les données actuarielles. Les actuaires sont d'abord passionnés par les chiffres afin d'analyser les différents risques. Souvent, les données sont en forme de tableau, il est important de prendre l'habitude de visualiser les données sous forme de graphique afin d'avoir un aperçu. Pensez seulement aux données aberrantes.

1.1 Créer un graphique

Il y'a plusieurs manières de visualiser les données avec `R`. Dans ce cours, nous utilisons le *package* `ggplot2`, ce dernier est très riche cohérent dans la création de graphiques, il est aussi le plus utilisé dans la communauté scientifique, l'aide sur internet est aussi très abondante.

Tout d'abord, il faut charger le *package* en question;

```
In [2]: library(ggplot2)
```

Utilisons les données `mpg` qui sont aussi dans le package `ggplot2` afin d'illustrer quelques exemples;

```
In [3]: head(mpg,3)
```

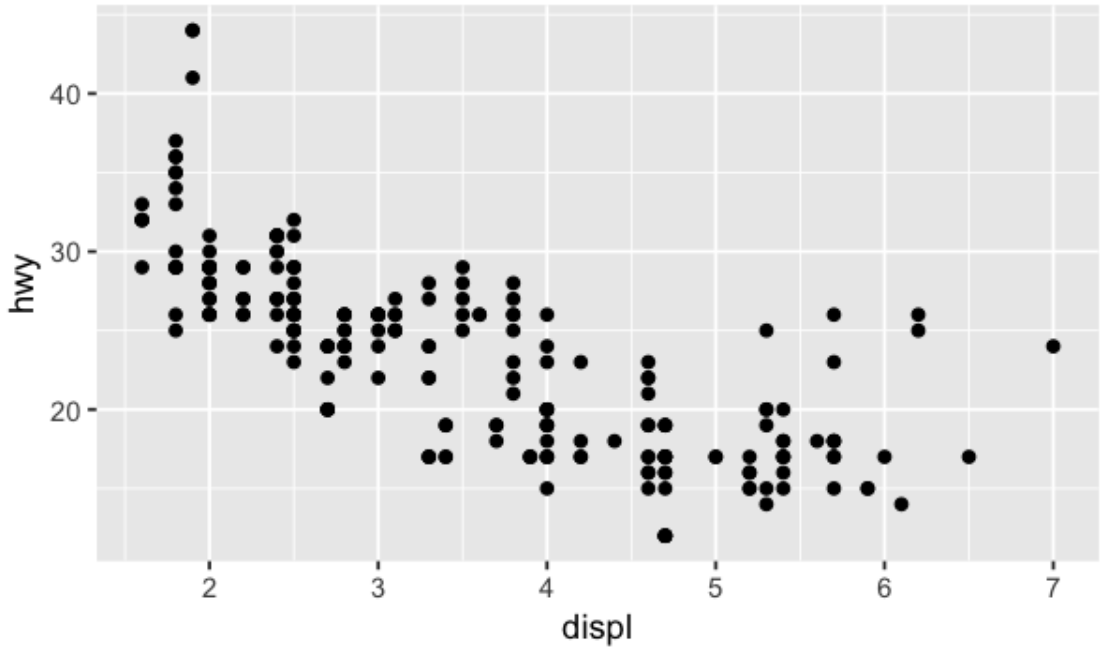
manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact

Nous nous intéressons alors à la relation entre la taille du moteur displ et hwy, l'efficacité énergétique d'une voiture sur l'autoroute, en miles par gallon(mpg). Une voiture à faible rendement énergétique consomme plus de carburant qu'une voiture à haut rendement énergétique lorsqu'ils parcourent la même distance.

1.2 geom

Chaque fonction `geom` dans ggplot2 prend un argument de mappage. Ceci définit comment les variables de votre ensemble de données sont mappées aux propriétés visuelles. L'argument de mappage est toujours associé à `aes ()`, et les arguments x et y de `aes ()` spécifient les variables à mapper avec les axes x et y. ggplot2 recherche la variable mappée dans l'argument de données, dans ce cas, mpg

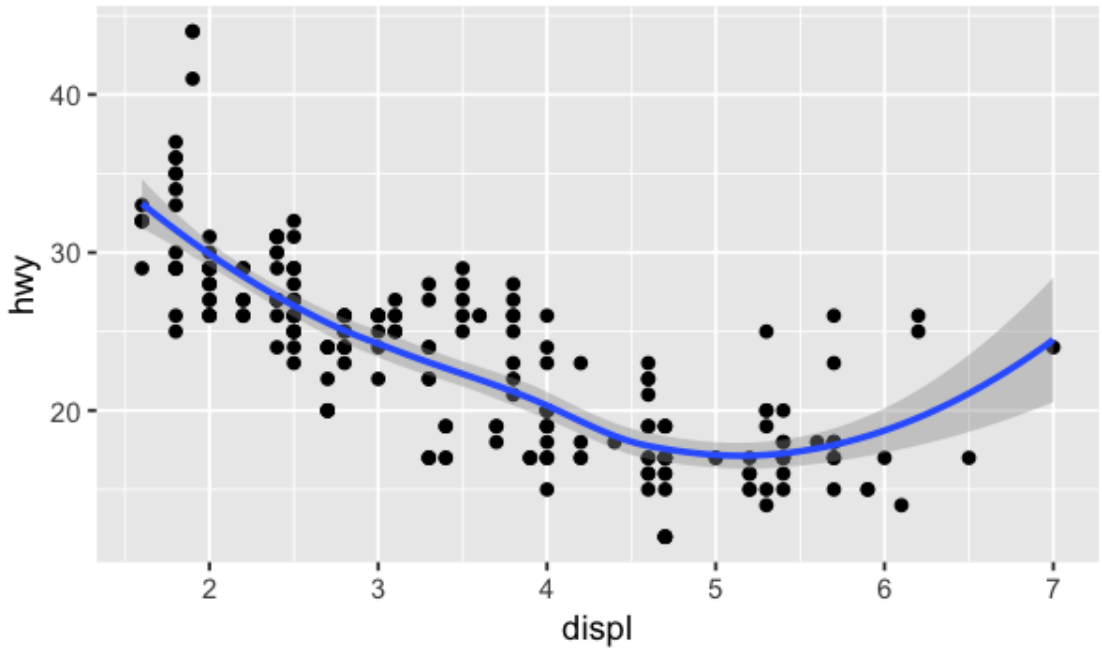
```
In [5]: options(repr.plot.width=5, repr.plot.height=3)
        ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy))
```



On peut mettre plusieurs `geom` dans un seul graphique. Ici, on ajoute une courbe de tendance aux points de notre graphique de base.

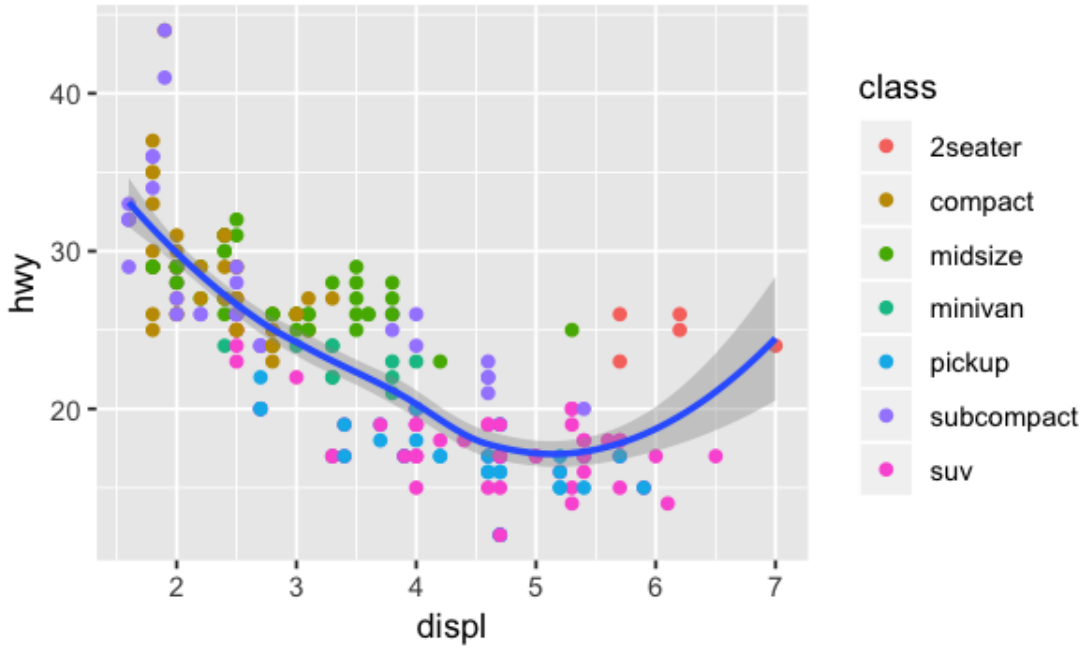
```
In [6]: ggplot(data = mpg) +
        geom_point(mapping = aes(x = displ, y = hwy)) +
        geom_smooth(mapping = aes(x = displ, y = hwy))
```

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



In [7]:
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth()
```

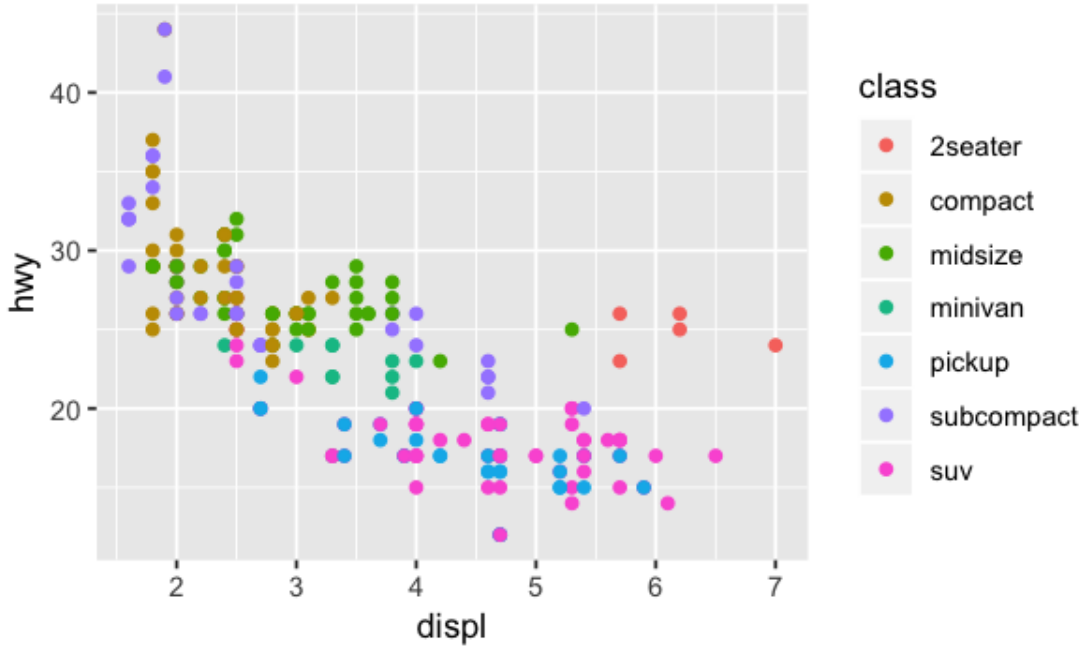
```
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



1.3 aes

Dans le graphique précédent, vous avez surement remarqué que certains points sont de mêmes couleurs. En effet, la fonction `aes` permet de modifier l'esthétique (*aesthetic*) du graphique. On peut alors ajouter une troisième variable comme la classe, à un nuage de points bidimensionnel. L'esthétique comprend d'autres éléments comme la taille, la forme ou la couleur de vos points.

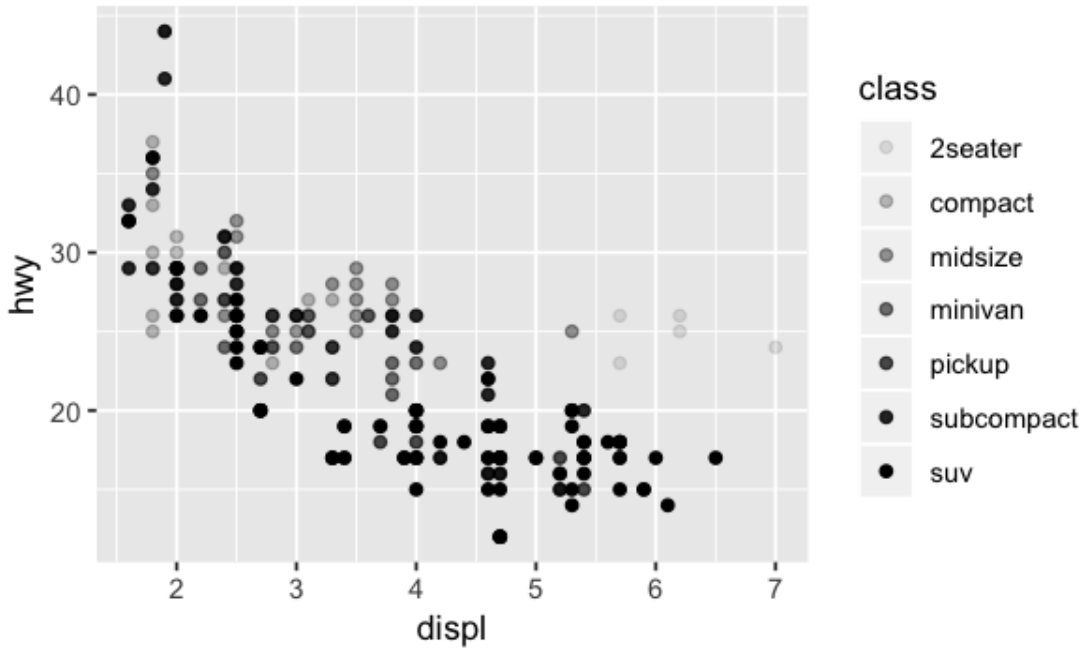
```
In [8]: ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



Si l'on utilise l'argument `alpha`, on obtient alors;

```
In [10]: ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy, alpha = class))
```

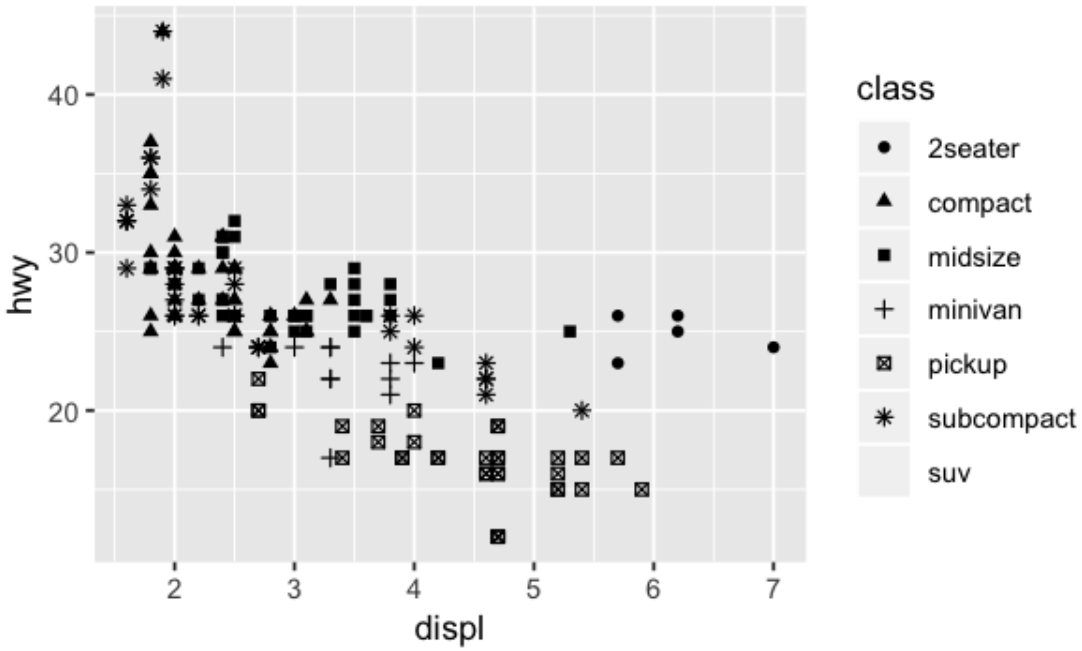
```
Warning message:
"Using alpha for a discrete variable is not advised."
```



ou la variable shape

```
In [11]: ggplot(data = mpg) +
         geom_point(mapping = aes(x = displ, y = hwy, shape = class))
```

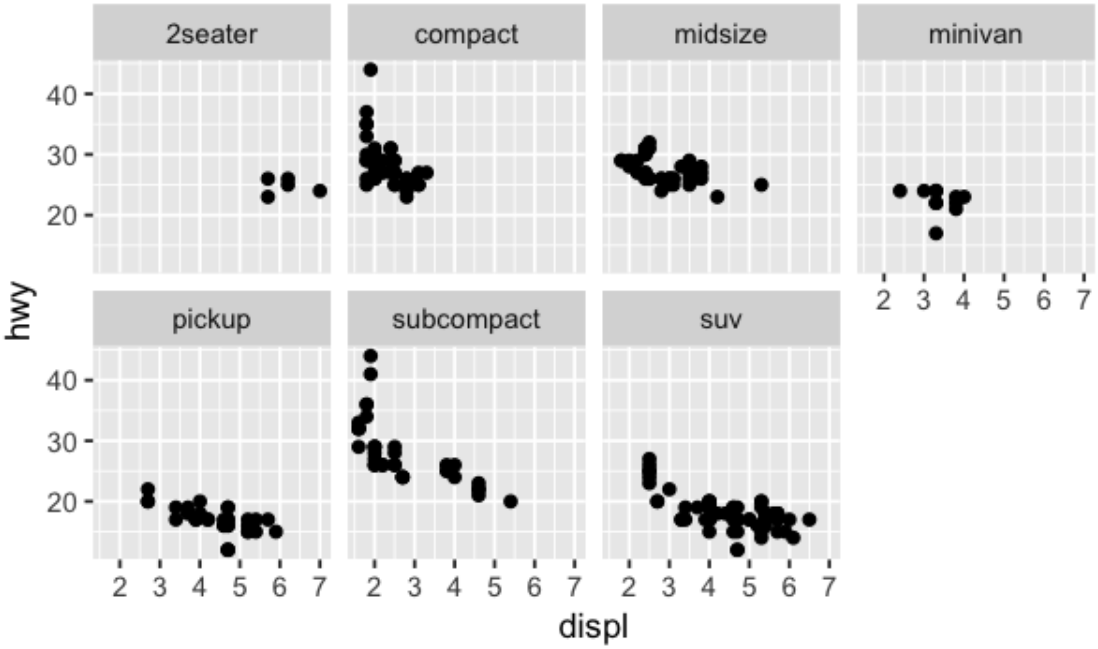
```
Warning message:
‘‘The shape palette can deal with a maximum of 6 discrete values because
more than 6 becomes difficult to discriminate; you have 7. Consider
specifying shapes manually if you must have them.’’Warning message:
‘‘Removed 62 rows containing missing values (geom_point).’’
```



on obtient un message d'avertissement nous indiquant que l'argument `shape` contient seulement 6 catégories et que dans nos donnés nous en avons 7:

Une autre méthode, particulièrement utile pour les variables catégorielles, consiste à diviser le graphique en facettes, sous-placettes qui affichent chacune une catégorie (ou groupe) des données.

```
In [12]: ggplot(data = mpg) +
         geom_point(mapping = aes(x = displ, y = hwy)) +
         facet_wrap(~ class, nrow = 2)
```



2 Quelques exemples

Bien évidemment, on ne peut pas couvrir tous cas possibles de graphiques. On ne peut les maîtriser que par la pratique sans avoir à connaître par coeur la syntaxe. Vous pouvez toujours consulter cet aide-mémoire qui résume bien comment tracer des graphiques avec la librairie `ggplot2`

2.1 geom_bar

Il est toujours plus rapide d'avoir une idée de la forme des données catégorielle lorsqu'on les représente dans un graphique de type `bar plot`

Soit les données suivantes qui montrent les moyennes des résultats d'examen du cours ACT3035 du groupe d'hiver 2018 vs automne 2017.

```
In [26]: c(rep(1,3), rep(2,3), rep(3,3))
```

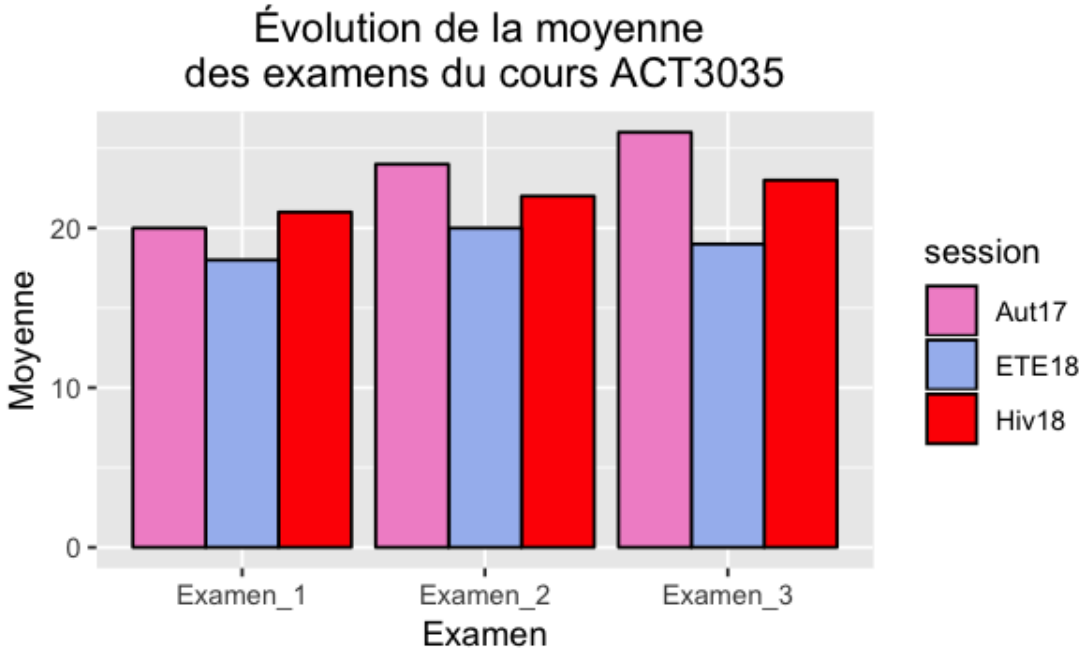
1. 1 2. 1 3. 1 4. 2 5. 2 6. 2 7. 3 8. 3 9. 3

```
In [27]: dat4 <- data.frame(
             session = factor(c("Aut17","Aut17","Aut17","Hiv18","Hiv18","Hiv18", "ETE18", "ETE18
             examen = c("Examen_1","Examen_2", "Examen_3","Examen_1","Examen_2", "Examen_3", "Ex
             moyenne = c(20, 24, 26, 21, 22, 23,18, 20, 19),
             ordre=c(c(rep(1,3), rep(2,3), rep(3,3)))
         )
```

```
In [31]: dat4
```

session	examen	moyenne	ordre
Aut17	Examen_1	20	1
Aut17	Examen_2	24	1
Aut17	Examen_3	26	1
Hiv18	Examen_1	21	2
Hiv18	Examen_2	22	2
Hiv18	Examen_3	23	2
ETE18	Examen_1	18	3
ETE18	Examen_2	20	3
ETE18	Examen_3	19	3

```
In [30]: ggplot(data=dat4, aes(x=reorder(examen, -ordre), y=moyenne, fill=session)) +
             geom_bar(stat="identity", position=position_dodge(), colour="black")+
             scale_fill_manual(values=c("#f293ce", "#a5bcef", "red")) +
             xlab("Examen") + ylab("Moyenne") +
             ggtitle("Évolution de la moyenne \ndes examens du cours ACT3035") +
             theme(plot.title = element_text(hjust = 0.5))
```



Rmarque: . Par défaut, le `geom_bar` utilise le nombre d'occurrences d'une variable (*count*) comme valeur de la haute de la barre. Si l'on veut que la hauteur corresponde à une valeur d'une variable dans le *data frame*, on doit alors préciser cela avec l'option `stat="identity"` car cette dernière est `stat="bin"` par défaut.

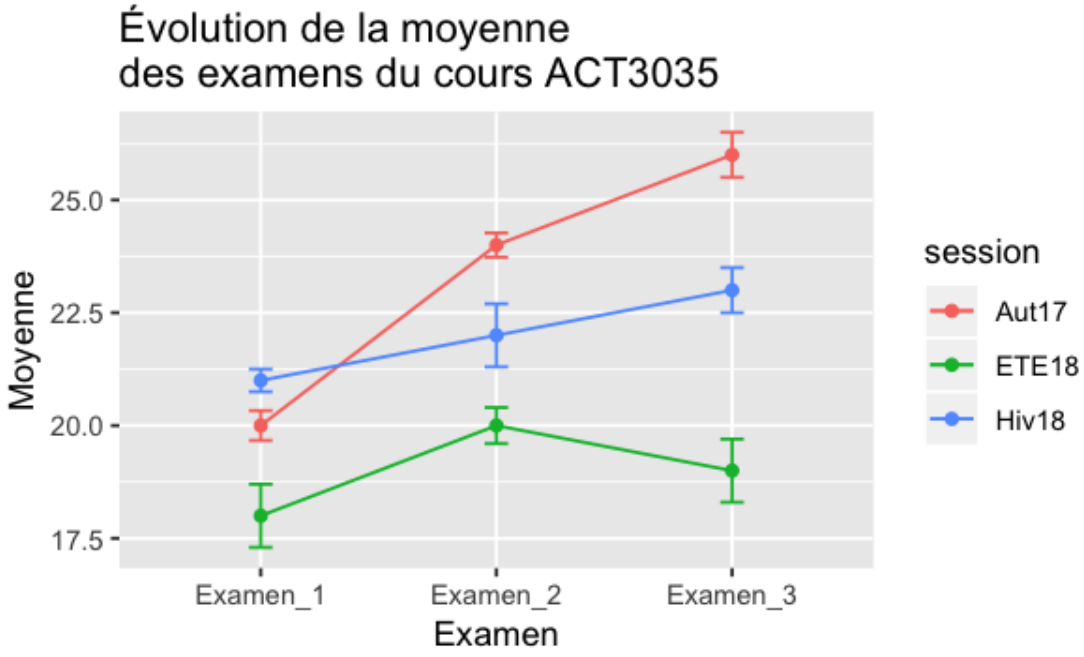
2.2 geom_line

On peut aussi représenter ces données sur des lignes continues en ajoutant deux petites lignes représentant la moyenne $\pm$ l'écart-type

Pour ce faire, ajoutons à notre df les données de l'écart-type

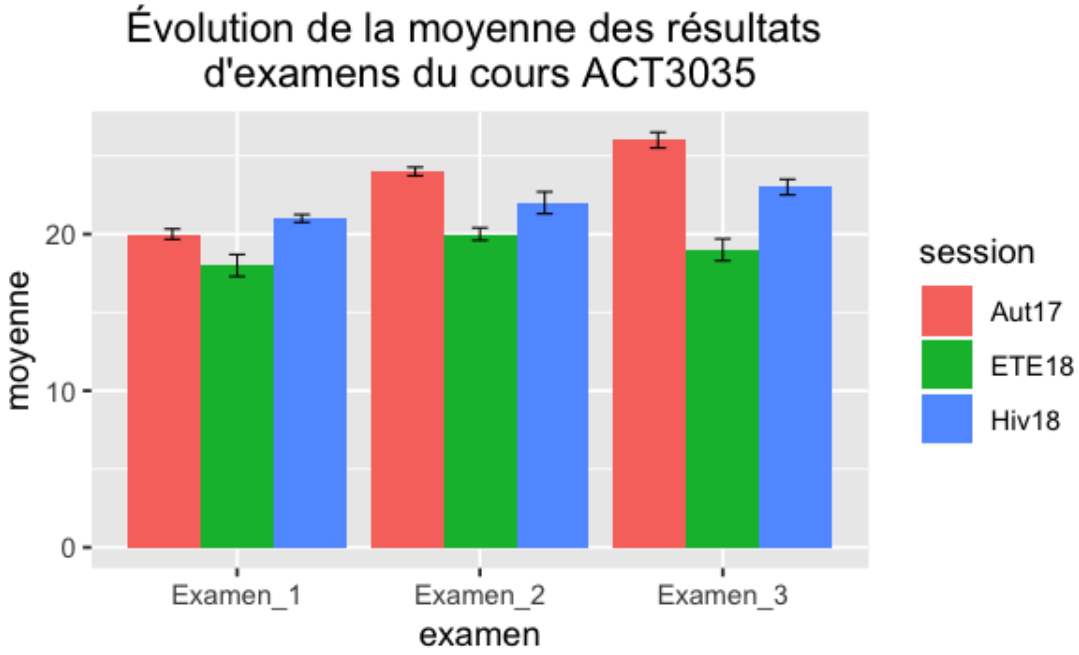
```
In [32]: dat4$se<-c(.33, .27, .5, .25, .7, .5, .7, .4, .7)
```

```
In [33]: ggplot(dat4, aes(x=examen, y=moyenne, colour=session, group=session,)) +
             geom_errorbar(aes(ymin=moyenne-se, ymax=moyenne+se), width=.1) +
             geom_line() +
             geom_point()+xlab("Examen") + ylab("Moyenne") +
             ggtitle("Évolution de la moyenne \ndes examens du cours ACT3035")
```



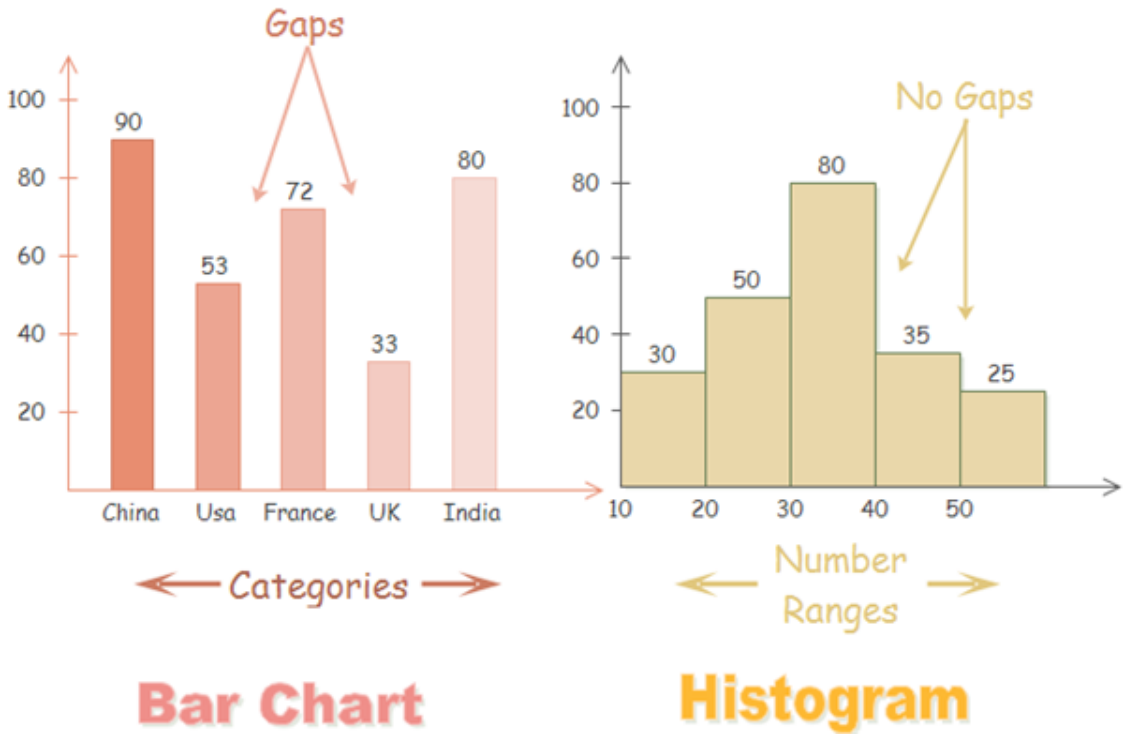
Nous pouvons faire la même chose avec les `geom_bar` que nous avons vu;

```
In [34]: ggplot(dat4, aes(x=examen, y=moyenne, fill=session)) +
             geom_bar(position=position_dodge(), stat="identity") +
             geom_errorbar(aes(ymin=moyenne-se, ymax=moyenne+se),
                           width=.2,                    # Width of the error bars
                           position=position_dodge(.8),
                         size=.3    # Thinner lines
                          )+
             ggtitle("Évolution de la moyenne des résultats \nd'examens du cours ACT3035") +
             theme(plot.title = element_text(hjust = 0.5))
```



2.3 geom_histogram

La grande différence entre un histogramme et un graphique de type `bar plot` se situe principalement dans le type de données. Dans le premier cas, les données sont groupées par des données catégorielles. Alors que dans le deuxième cas, les données sont présentées en groupe de données continues (quantitative).



Nous avons besoin alors des données continues afin d'illustrer quelques exemples d'histogrammes. Générons-en quelques données continues;

```
In [35]: set.seed(1234)
         dat <- data.frame(cond = factor(rep(c("A","B"), each=200)),
                           rating = c(rnorm(200),rnorm(200, mean=.8)))
```

Nous avons généré 200 observations **(A)** tirées d'une distribution normale centrée réduite, et 200 observations **(B)** tirées d'une distribution normale de moyenne mu=.8

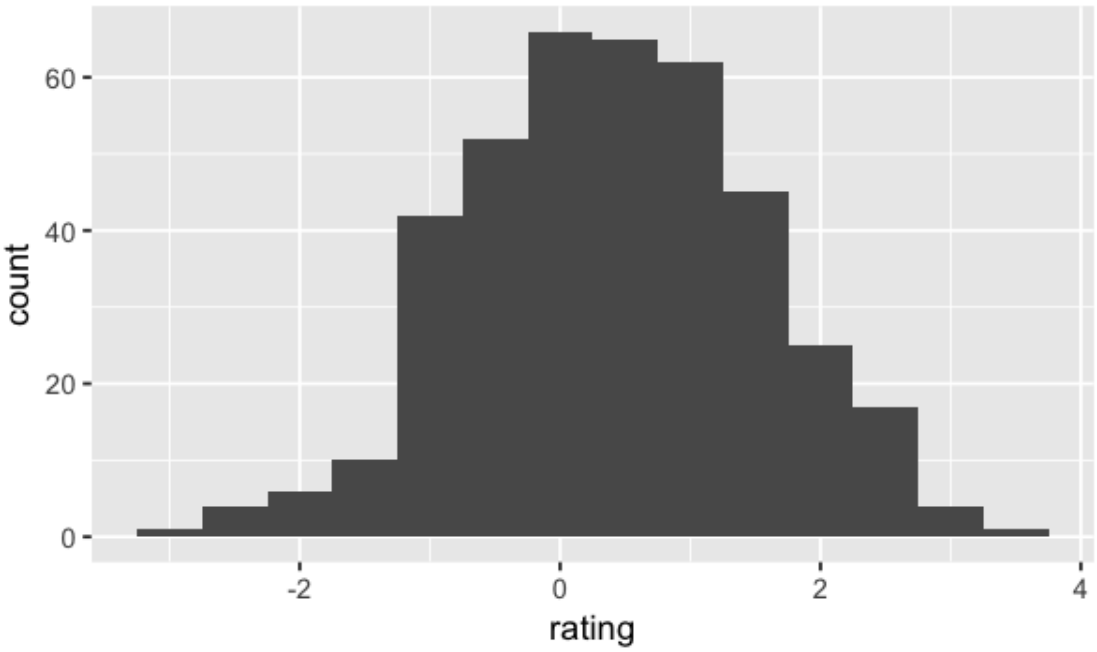
```
In [17]: table(dat$cond)
```

```
  A   B
200 200
```

```
In [38]: # head(dat[sample(nrow(dat)),])
```

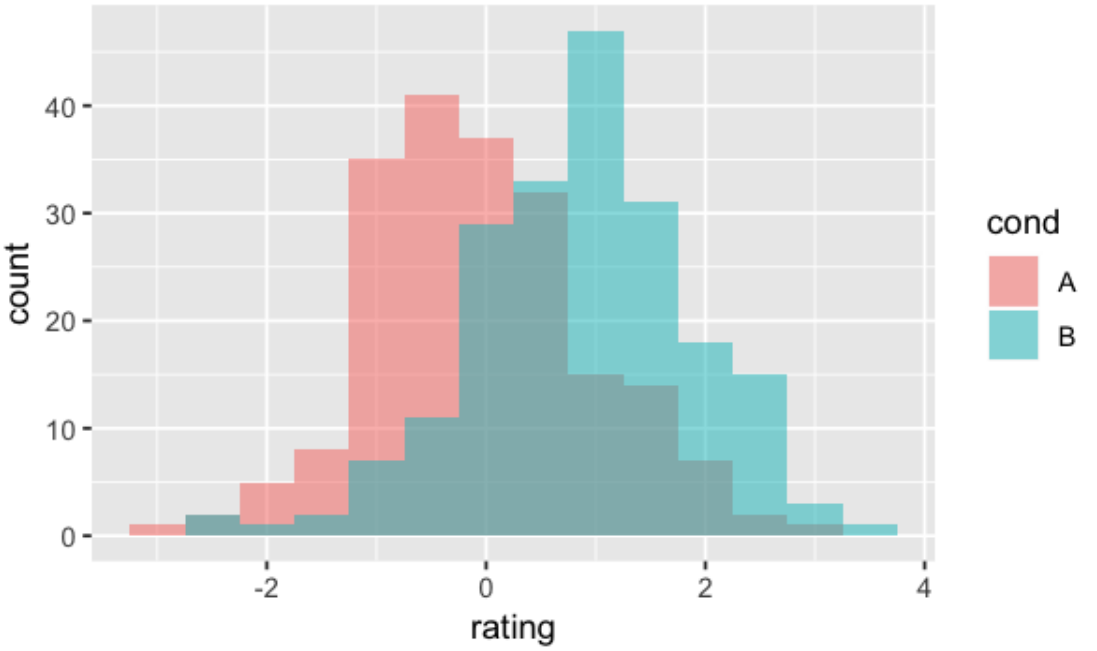
Puisque nous avons là des données continues, représentons-les alors dans un graphique de type `geom_histogram`;

```
In [39]: ggplot(dat, aes(x=rating)) + geom_histogram(binwidth=.5)
```



On voit bien qu'on ne peut pas distinguer les observations `A` de `B`. Pour ce faire, nous devons alors ajouter la condition à `fill=cond` à notre `aes`

```
In [40]: ggplot(dat, aes(x=rating, fill=cond)) +
             geom_histogram(binwidth=.5, alpha=.5, position="identity")
```



On peut même ajouter la moyenne de chaque variable;

```
In [41]: library(plyr)
         cdat <- ddply(dat, "cond", summarise, rating.mean=mean(rating))
         cdat
```

cond	rating.mean
A	-0.05775928
B	0.87324927

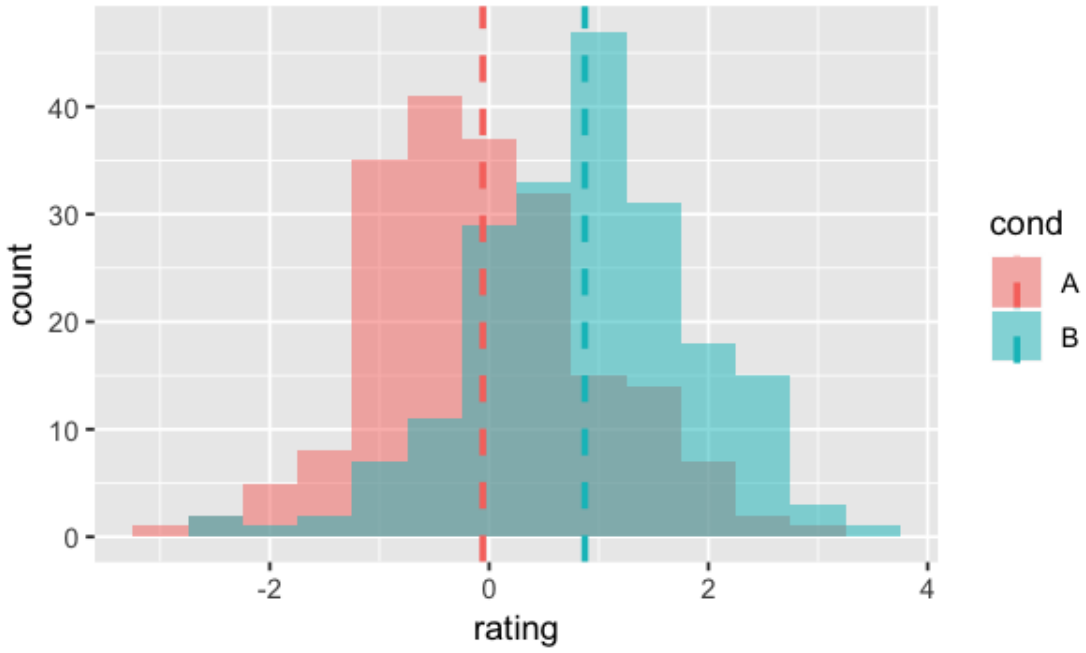
Ou calculer directement la moyenne de chaque groupe comme suit;

```
In [22]: moyenne_B<-mean(dat$rating[dat$cond == "B"])
         moyenne_B
```

0.873249267913921

Utilisons les données 'cdat

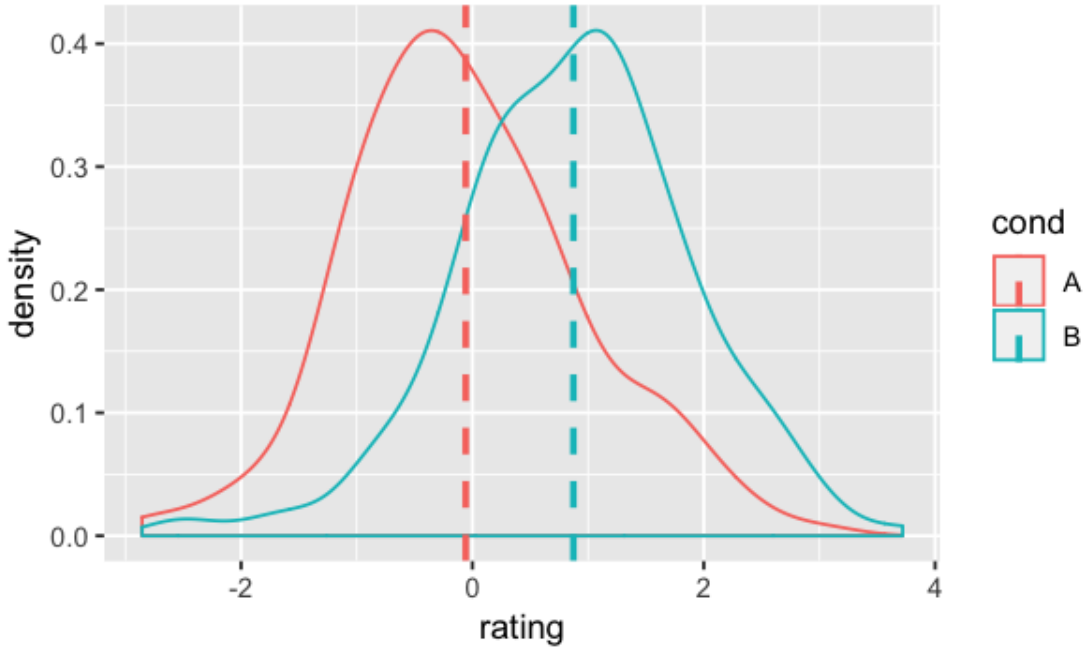
```
In [42]: ggplot(dat, aes(x=rating, fill=cond)) +
             geom_histogram(binwidth=.5, alpha=.5, position="identity") +
             geom_vline(data=cdat, aes(xintercept=rating.mean,  colour=cond),
                        linetype="dashed", size=1)
```



2.4 geom_density

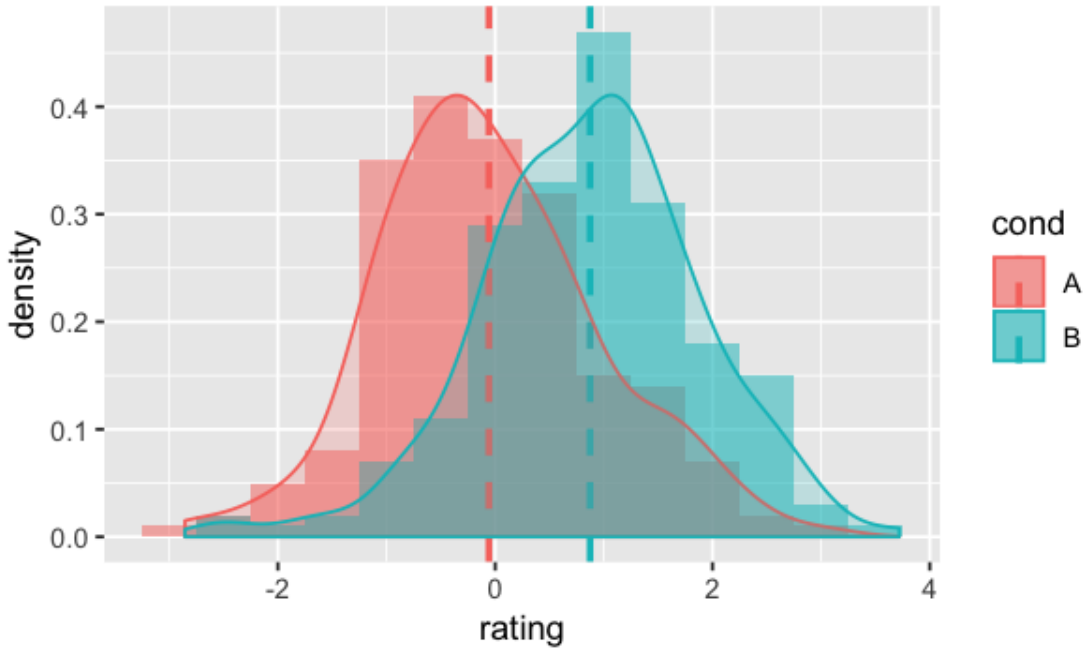
On peut présenter ces mêmes données sous deux courbes puisqu'elles sont continues;

```
In [43]: ggplot(dat, aes(x=rating, colour=cond)) +
             geom_density() +
             geom_vline(data=cdat, aes(xintercept=rating.mean,  colour=cond),
                        linetype="dashed", size=1)
```



Sans oublier qu'il est aussi possible d'ajouter des courbes aux histogrammes;

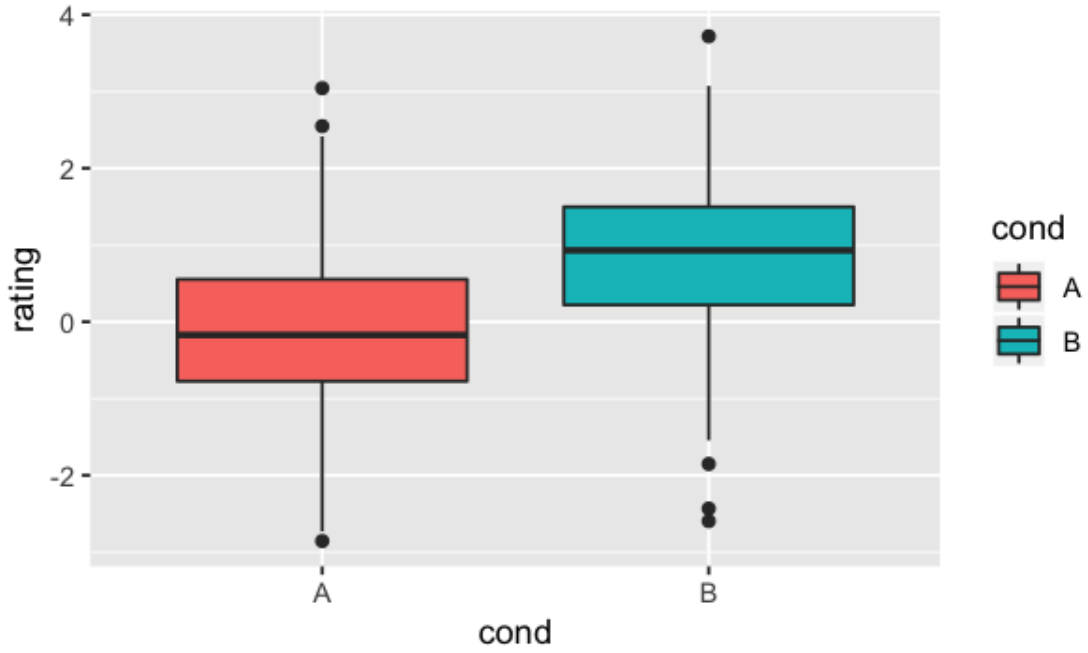
```
In [44]: ggplot(dat, aes(x=rating, fill=cond)) +
             geom_histogram(aes(y=..density..),binwidth=.5, alpha=.5, position="identity") +
             geom_vline(data=cdat, aes(xintercept=rating.mean,  colour=cond),
                        linetype="dashed", size=1)+geom_density(alpha=.2, aes( colour=cond))
```



2.5 geom_boxplot

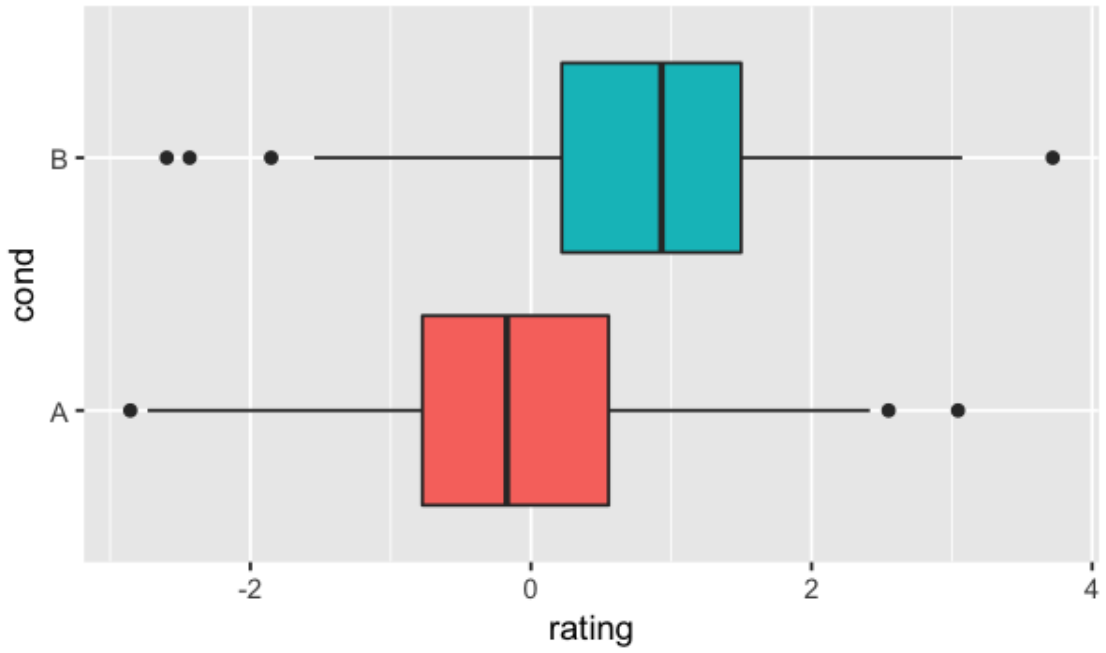
Lorsqu'il s'agit de présenter les données statistiques de base sur des catégories de variables, un graphique de type `geom_boxplot` devient alors incontournable;

```
In [45]: ggplot(dat, aes(x=cond, y=rating, fill=cond)) + geom_boxplot()
```



Voici comment renverser les axes;

```
In [46]: ggplot(dat, aes(x=cond, y=rating, fill=cond)) + geom_boxplot() +
             guides(fill=FALSE) + coord_flip()
```



2.6 ggcorrplot

Lorsque nous avons des données où nous trouvons la corrélation entre chaque variable (deux à deux), il peut être difficile de voir où sont les variables les plus (ou les moins) corrélées;

```
In [47]: df_app<-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/stocks_correla
                          header = T)[ ,2:9]
         head(df_app)
```

DO	AMZN	AAPL	BA	FB	HON	ITW	NFLX
18.06	750.57	108.6043	146.1413	118.42	111.5562	122.7038	117.00
17.70	768.66	114.3968	152.2714	115.05	114.0969	124.5074	123.80
16.38	823.48	119.8588	159.8419	130.32	116.5296	125.3345	140.71
16.84	845.04	135.3066	176.2839	135.54	122.6161	130.0739	142.13
16.71	886.54	142.5098	174.4757	142.05	123.6397	130.5272	147.81
14.42	924.99	142.4999	182.3383	150.25	129.8480	136.7335	152.20

Dans ces données, nous trouvons le prix de l'action des entreprises citées dans le tableau au 2017-01-01. On peut alors calculer la corrélation entre les variables avec la fonction `corr`

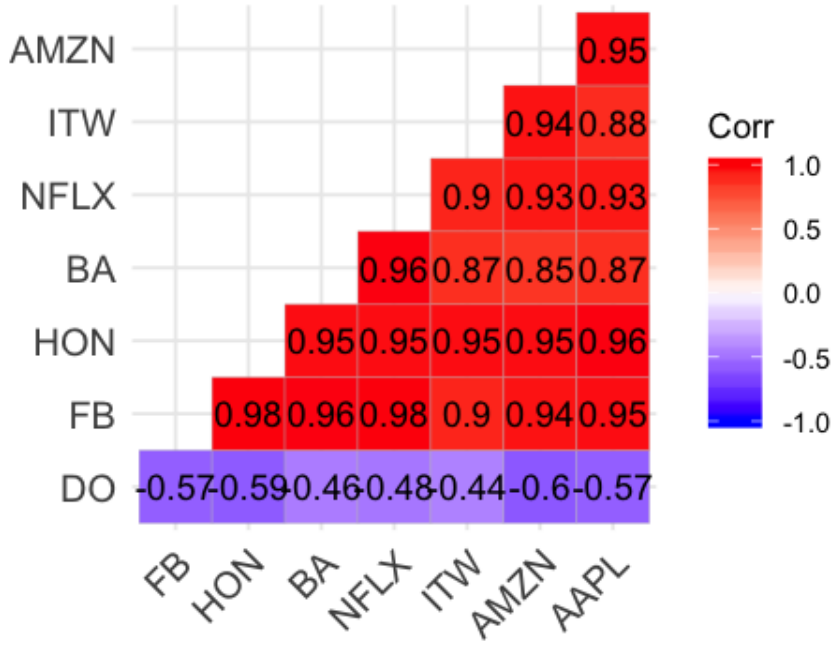
```
In [48]: mat_corr<-cor(df_app)
         mat_corr
```

	DO	AMZN	AAPL	BA	FB	HON	ITW	NFLX
DO	1.0000000	-0.5991331	-0.5734723	-0.4594365	-0.5740901	-0.5931147	-0.4391862	-0.48087
AMZN	-0.5991331	1.0000000	0.9491803	0.8523493	0.9373536	0.9481765	0.9413468	0.93135
AAPL	-0.5734723	0.9491803	1.0000000	0.8722640	0.9504718	0.9553516	0.8783502	0.92754
BA	-0.4594365	0.8523493	0.8722640	1.0000000	0.9606722	0.9450005	0.8729395	0.95504
FB	-0.5740901	0.9373536	0.9504718	0.9606722	1.0000000	0.9786910	0.8972479	0.97628
HON	-0.5931147	0.9481765	0.9553516	0.9450005	0.9786910	1.0000000	0.9464567	0.95398
ITW	-0.4391862	0.9413468	0.8783502	0.8729395	0.8972479	0.9464567	1.0000000	0.89938
NFLX	-0.4808701	0.9313529	0.9275452	0.9550461	0.9762811	0.9539829	0.8993815	1.00000

Utilisons la librairie `ggcorrplot` qui nous offre beaucoup d'options pour visualiser un tel type de graphique

```
In [49]: library(ggcorrplot)
```

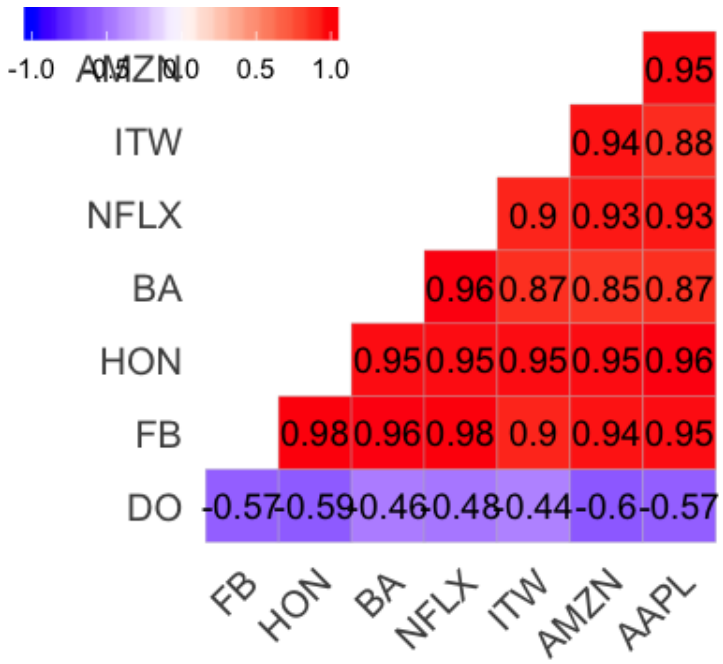
```
In [50]: ggcorrplot(mat_corr, hc.order = TRUE, type = "lower",
             lab = TRUE)
```



Modifions un peu ce graphique;

```
In [51]: ggcorrplot(mat_corr, hc.order = TRUE, type = "lower",
              lab = TRUE)+theme(
            axis.title.x = element_blank(),
            axis.title.y = element_blank(),
            panel.grid.major = element_blank(),
```

```
panel.border = element_blank(),
panel.background = element_blank(),
axis.ticks = element_blank(),
legend.justification = c(1, 0),
legend.position = c(0.3, 0.85),
legend.direction = "horizontal")+
guides(fill = guide_colorbar(barwidth = 7, barheight = 1,
              title.position = "top", title.hjust = 0.5, title="Corrélation Pearson")
```



3 Conclusion

Bien sûr, il ne faut pas s'attendre à ce que l'on connaisse par coeur toutes les options de graphique. Car de toute façon, il existe beaucoup d'exemples dans la documentation de ggplot2.

5_2_cours

October 15, 2018

1 La méthode de dichotomie

Selon la définition de Wikipedia;

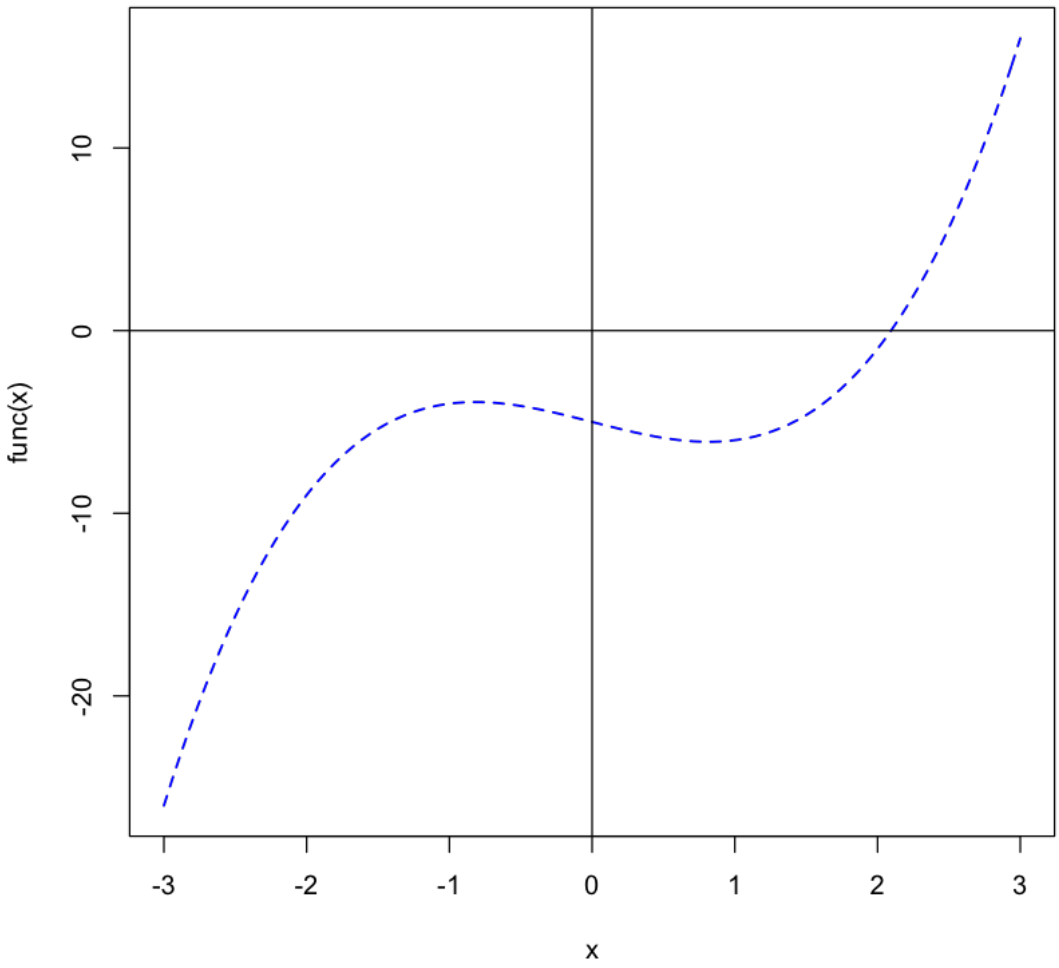
> "La méthode de dichotomie ou méthode de la bissection est, en mathématiques, un algorithme de recherche d'un zéro d'une fonction qui consiste à répéter des partages d'un intervalle en deux parties puis à sélectionner le sous-intervalle dans lequel existe un zéro de la fonction."

Prenons l'exemple da la fonction suivante:

$$x^3 - 2x - 5$$

Si l'on trace un graphique de cette fonction, on obtient ceci:

```
In [22]: func <- function(x) {
           x^3 - 2 * x - 5
         }
         curve(func, xlim=c(-3,3), col='blue', lwd=1.5, lty=2)
         abline(h=0)
         abline(v=0)
```



On peut alors programmer l'algorithme suivant afin de trouve le zéro de cette fonction;

```
In [23]: bisec_w <-function(f, a, b, e=1e-7){
         while (b-a > e) {
             m <- (a+b)/2
             ifelse(f(a)*f(m)<=0, b <- m, a <- m)
         }
             return(m)
             }
```

On peut vérifier alors le résultat de cette fonction:

```
In [24]: bisec_w(func, 2, 3)
```

2.09455150365829

2 Calcul numérique d'une intégrale

Voici un exemple de calcul numérique d'une intégrale évalue entre deux bornes;

```
In [25]: fonction <- function(x) {1/((x+1)*sqrt(x))}
```

$$\int_0^{\text{inf}} \frac{1}{(x+1)\sqrt{x}} dx \tag{1}$$

```
In [26]: integrate(fonction, lower = 0, upper = Inf)
```

```
3.141593 with absolute error < 2.7e-05
```

$$\int_{-1.96}^{1.96} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx \tag{2}$$

```
In [27]: f <- function(x) {1/sqrt(2*pi)*exp(-x^2/2)}
         integrate(f, lower = -1.96, upper = 1.96)
```

```
0.9500042 with absolute error < 1e-11
```

5_3_Exercices_Solutions

October 15, 2018

1 Q1

Faites les étapes suivantes:

- lire les données à partir de l'url donnes_demo (3 pts)
- réer une fonction qui calcule l'âge de chaque client en date du premier jours du mois courant dans une nouvelle variable d'aun
- Créer un graphique du type *Bar Chart* sur le âge des clients. Par ce graphique, on verra facilement le nombre d'assurés par catégorie d'âge (3 pts)

Votre graphique contient les éléments suivants: * Un titre (1 pts) * Une étiquette de l'axe des x (âge des assurés) (1 pts) * Une étiquette de l'axe des y (nombre d'assurés) (1 pts)

```
In [60]: a<-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/1000_HF.csv")
```

```
In [61]: head(a)
```

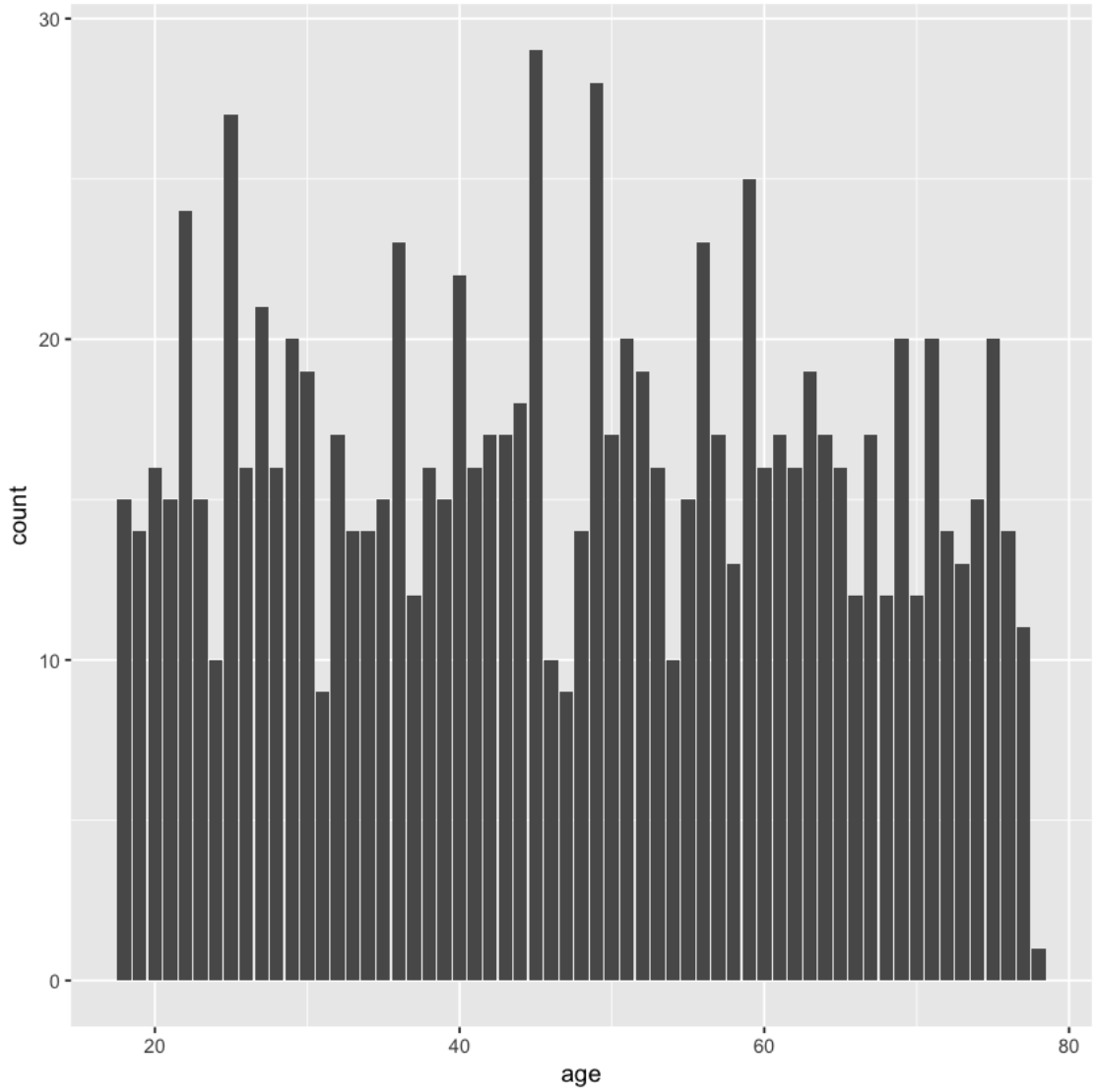
first_name	last_name	birth_date	address	job
Anaïs	Chevallier	1944-08-06	87930 Justin Inlet	Chargé de recherche en acoustique m
Christine	Leveque	1951-01-02	1792 Lauren Glens	Secrétaire juridique
Maryse	Chartier	1988-01-24	71833 Emily Gateway	Développeur humanitaire
Avide	Damico	1967-03-31	5510 Christine Land	Conseiller en séjours
Anaïs	Laroche	1975-02-20	1123 Tracy Landing Suite 232	Technicien
Pénélope	Bernard	1952-06-01	0232 Mccullough Divide	Chercheur en biologie

```
In [62]: library(lubridate)
         dt <- Sys.Date()
         day(dt) <- 01
         a$age<-round(as.numeric(dt-as.Date(a$birth_date))/365.25,0)
```

```
In [8]: library(ggplot2)
```

```
In [57]: lectu_graph<-function(data, variable, ...){
             ret <- ggplot(data, aes_string(x=variable), ...) + geom_bar()
             return(ret)
         }
```

```
In [59]: lectu_graph(a, "age")
```



2 Q2

https://s3.amazonaws.com/www.no

À partir de la base de données suivante, reproduisez le graphique suivant*:

Ignorez la variable `freq_pmt`. Considérez que les paiements sont reçus une fois par année et c'est au même mois que le mois de la date d'expiration. *chaque détail compte (titre, xlab, ordre des mois ...etc)

```
In [ ]: # install.packages("httr")
```

```
In [ ]: library(httr)
```

```
In [63]: url<-"https://raw.githubusercontent.com/nmeraihi/data/master/pmt_details.csv"
```

```
In [ ]: http_error(url) #veerifaction que ma variable url est bien un url sans erreur
```

Donc on peut faire un `if` pour valider le url

```
In [ ]: if (http_error(url)==F){
            pmt_det<-read.csv(url)
        }
```

```
In [ ]: head(pmt_det)
```

ou le faire comme d'habitude;

```
In [64]: pmt_det<-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/pmt_details.c
         head(pmt_det)
```

numeropol	cout_prime	credit_card_number	credit_card_provider	credit_card_expire	freq_pmt
1	1060.28	4.427476e+15	Voyager	04/23	12
5	1200.89	5.303389e+15	JCB 16 digit	08/26	1
13	940.54	3.528569e+15	Maestro	08/22	12
16	860.75	6.011570e+15	VISA 13 digit	03/23	1
22	790.17	5.262495e+15	Maestro	08/20	1
28	940.16	4.583364e+15	Discover	03/20	1

l'avantage de vérifier l'url et de valider le tout dans une fonction avant l'exécution:

```
In [65]: require("lubridate")
```

```
In [67]: require("zoo")
```

```
In [ ]: # detach("package:zoo", unload=TRUE) # juste pour unload le package afin de vous
        # montrer l'utilité de ce package dans la ligne qui suit
```

Puisque le format de la date d'éxpiration de la carte de crédit a un format date inhabituel, on utilisera le pckage `zoo` afin de pouvoir extraire le mois de cette date.
Remarquez qu'on aurait pu utiliser les fonction `substr` pour extraire le mois seulement, toutefois il faut retransformer le charctère extrait en `int64`

```
In [68]: mois<-as.yearmon(pmt_det$credit_card_expire, "%m/%y")
```

Maintenant on peut finalement utiliser la fonction `month` pour avoir le mois;

```
In [69]: month.abb[head(as.numeric(format(mois, "%m")))]
```

1. 'Apr' 2. 'Aug' 3. 'Aug' 4. 'Mar' 5. 'Aug' 6. 'Mar'
insérons le tout dans une nouvelle variable (colonne)

```
In [70]: pmt_det$mois_pmt<-month.abb[as.numeric(format(mois, "%m"))]
```

Suite à la question de un de vos collègues, voici la fonction qui permet d'ordonner `sort`

```
In [71]: sort(factor(head(pmt_det$mois_pmt), levels = month.abb))
```

1. Mar 2. Mar 3. Apr 4. Aug 5. Aug 6. Aug

```
In [72]: head(pmt_det$mois_pmt)
```

1. 'Apr' 2. 'Aug' 3. 'Aug' 4. 'Mar' 5. 'Aug' 6. 'Mar'
Jetons un coup d'oeil à notre nouveau df

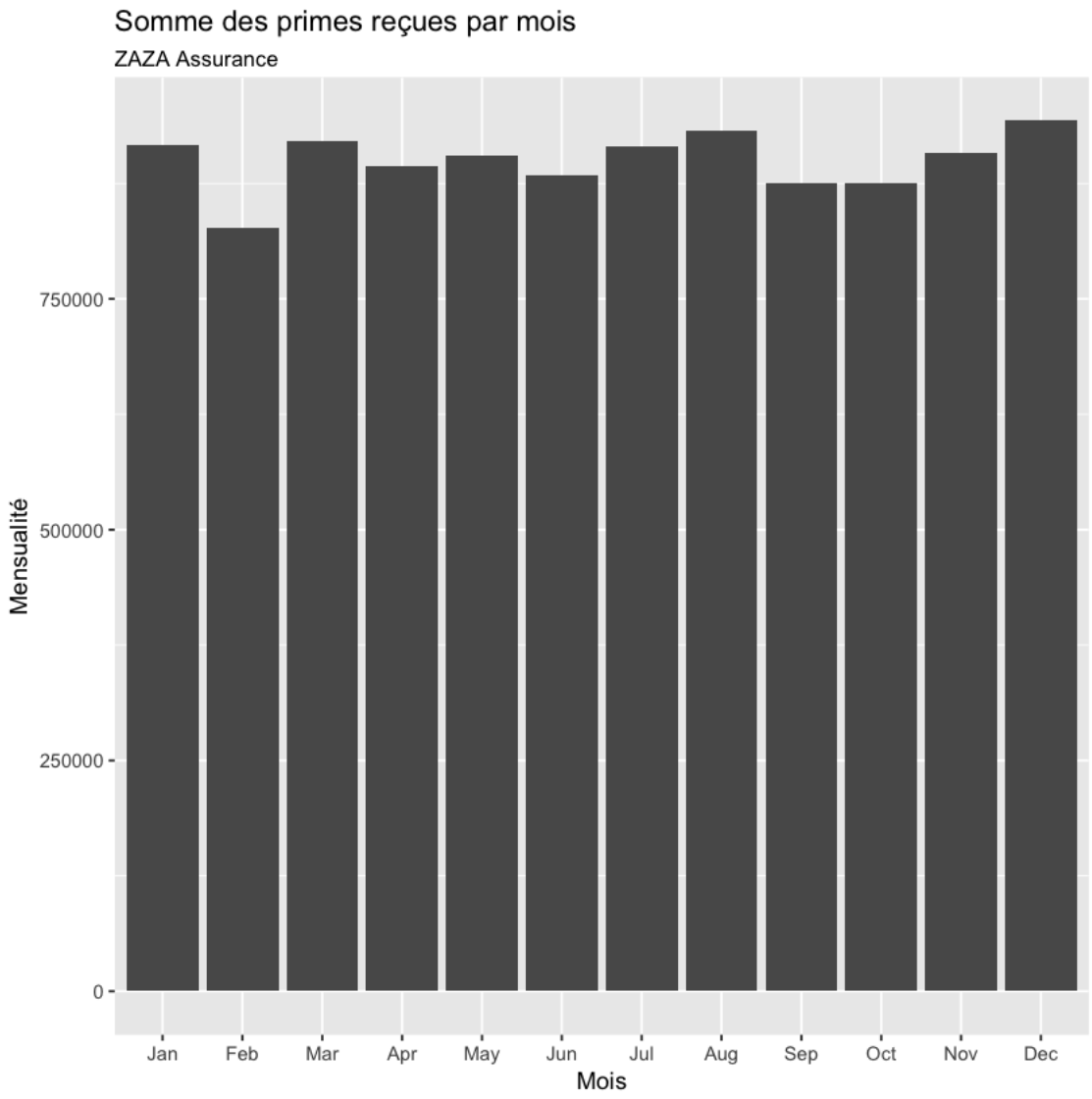
```
In [73]: head(pmt_det)
```

numeropol	cout_prime	credit_card_number	credit_card_provider	credit_card_expire	freq_pmt	r
1	1060.28	4.427476e+15	Voyager	04/23	12	A
5	1200.89	5.303389e+15	JCB 16 digit	08/26	1	A
13	940.54	3.528569e+15	Maestro	08/22	12	A
16	860.75	6.011570e+15	VISA 13 digit	03/23	1	M
22	790.17	5.262495e+15	Maestro	08/20	1	A
28	940.16	4.583364e+15	Discover	03/20	1	M

Et maintenant utilisons `ggplot` pour tracer le graph demandé;

```
In [74]: library(ggplot2)
```

```
In [75]: ggplot(data = pmt_det,
           aes(sort(factor(pmt_det$mois_pmt, levels = month.abb)), cout_prime)) +
           stat_summary(fun.y = sum,
             geom = "bar", )+
             xlab("Mois")+
             ylab("Mensualité")+
             labs(title = "Somme des primes reçues par mois", subtitle="ZAZA Assurance")
```



3 Q3

3.1 a)

Faites un graphique qui permet de voir l'évolution des coûts de sinistre dans le temps. Sur l'axe des x, on devrait voir les mois et l'année (1999-01, 1999-02 ...). À des fins de l'exercice, imaginez que s'il y'a un sinistre, il se passe toujours la même date que le `debut_pol` dans les données suivantes. Vous pouvez utiliser la fonction `aggregate` pour regrouper les sinistres par mois.

```
In [78]: donnes_demo<-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/donnes_de
         head(donnes_demo)
```

name	province	company	langue	date_naissance	agee	age_p
Shane Robinson	Nova Scotia	May Ltd	fr	1944-10-20	72	24
Courtney Nguyen	Saskatchewan	Foley, Moore and Mitchell	en	1985-12-09	31	24
Lori Washington	Yukon Territory	Robinson-Reyes	fr	1970-01-27	47	28
Sarah Castillo	Alberta	Wood, Brady and English	fr	2000-08-23	16	16
Jeffrey Garcia	Nunavut	Berger-Thompson	en	1969-10-25	47	20
Colleen Coleman	Saskatchewan	Simmons-Smith	en	1984-10-16	32	23

```
In [81]: tail(donnes_demo)
```

	name	province	company	langue	date_naissance
15	Heather Maldonado	Nunavut	Walker Group	en	1999-02-23
16	Christina Howard	Nova Scotia	Pena and Sons	en	1969-05-22
17	Karen Nguyen	Northwest Territories	Price PLC	fr	1972-12-20
18	Connie Alvarado	Manitoba	Jensen-Cooper	en	1974-10-18
19	Heidi Freeman	Northwest Territories	Singh, Esparza and Santos	en	1951-06-07
20	Morgan Buchanan	Northwest Territories	Rollins Inc	fr	1971-07-31

```
In [79]: police_assurance<-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/poli
```

```
In [80]: head(police_assurance)
```

numeropol	debut_pol	fin_pol	cout1	cout2	cout3	cout4	cout5	cout6	cout7	nbsin
1	1999-11-10	2000-10-16	NA	NA	NA	NA	NA	NA	NA	0
1	2000-10-17	2000-11-09	NA	NA	NA	NA	NA	NA	NA	0
1	2000-11-10	2001-11-09	243.8571	NA	NA	NA	NA	NA	NA	1
5	1996-01-03	1996-03-27	NA	NA	NA	NA	NA	NA	NA	0
5	1996-03-28	1997-01-02	NA	NA	NA	NA	NA	NA	NA	0
5	1997-01-03	1998-01-02	NA	NA	NA	NA	NA	NA	NA	0

Creéons une nouvelle variable `ann_mois_sinistre` où nous conservons seulement le mois et l'année de la date `debut_pol`

```
In [82]: police_assurance$ann_mois_sinistre<-format(as.Date(police_assurance$debut_pol), "%Y-%m"
```

Ensuite nous faisons une somme sur toutes les variables (cout1@cout7). Il est important d'ajouter l'argument `na.rm=T` afin de prendre en considération les `na`

```
In [83]: police_assurance$somme_couts<-apply(police_assurance[,4:10],1 ,sum, na.rm=TRUE)
```

Ensuite nous faisons un sommaire (somme) de tous les coûts totaux groupé par mois **ET** année

```
In [84]: df_q6<-aggregate(police_assurance$somme_couts,
                          by=list(ann_mois_sinistre=police_assurance$ann_mois_sinistre), FUN=sum
```

```
In [85]: head(df_q6)
```

ann_mois_sinistre	x
1995-01	117.4658
1995-02	67543.4286
1995-03	4860.8261
1995-04	5738.0932
1995-05	5449.1056
1995-06	13850.9193

On voit que maintenant nos données vont du 1995-01 au 2004-05

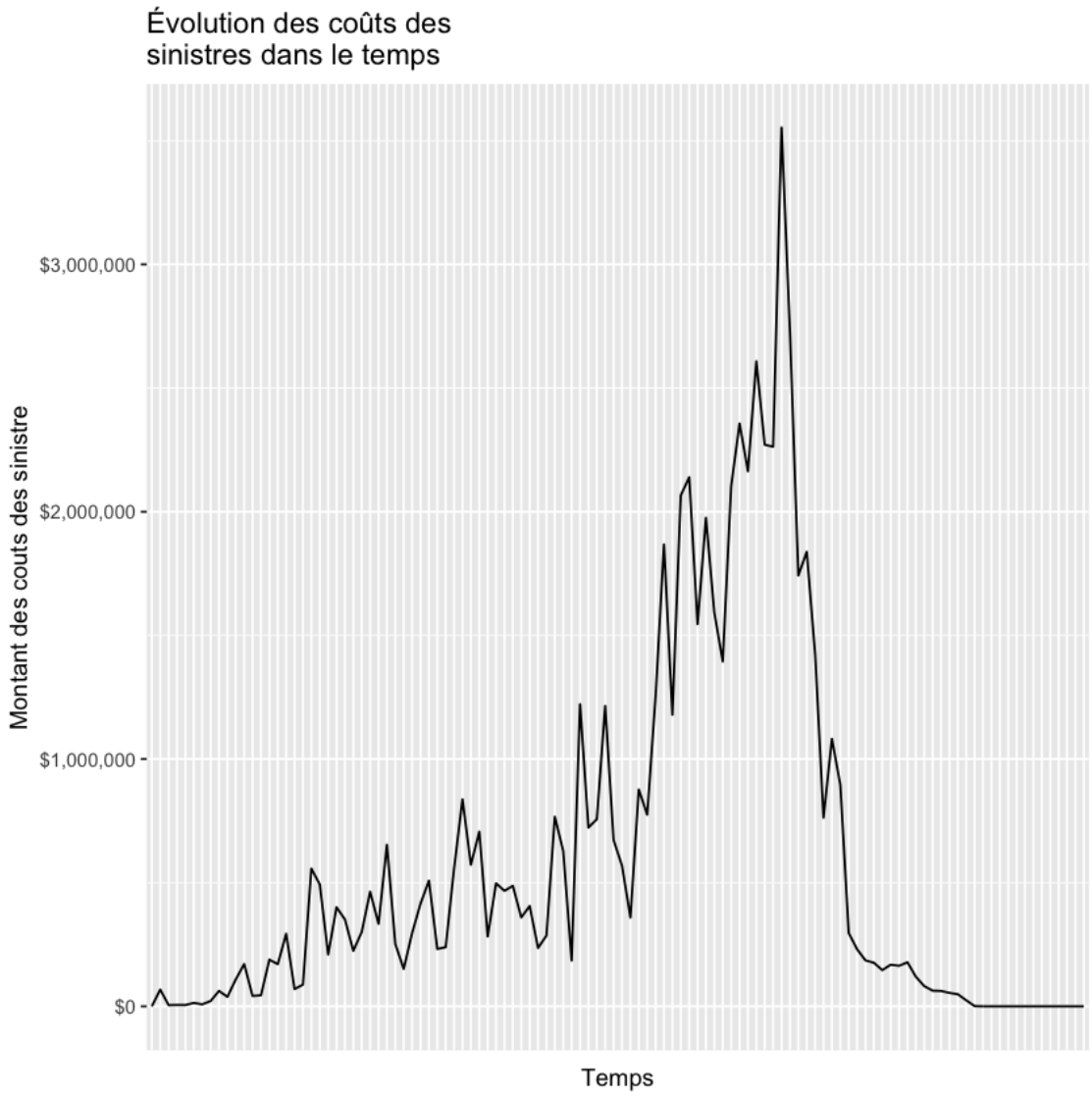
In [86]:
```
max(police_assurance$ann_mois_sinistre)
```

'2004-05'

In [87]:
```
library(scales)
```

In [ ]:
```
# detach("package:scales", unload=TRUE)
```

In [88]:
```
ggplot(data=df_q6, aes(x=ann_mois_sinistre, y=x, group=1)) +
    geom_line() +
    xlab("Temps") + ylab("Montant des couts des sinistre") +
    scale_y_continuous(labels = dollar)+ # c'est là que sert le package scale
    ggtitle("Évolution des coûts des \nsinistres dans le temps")+
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank())
```



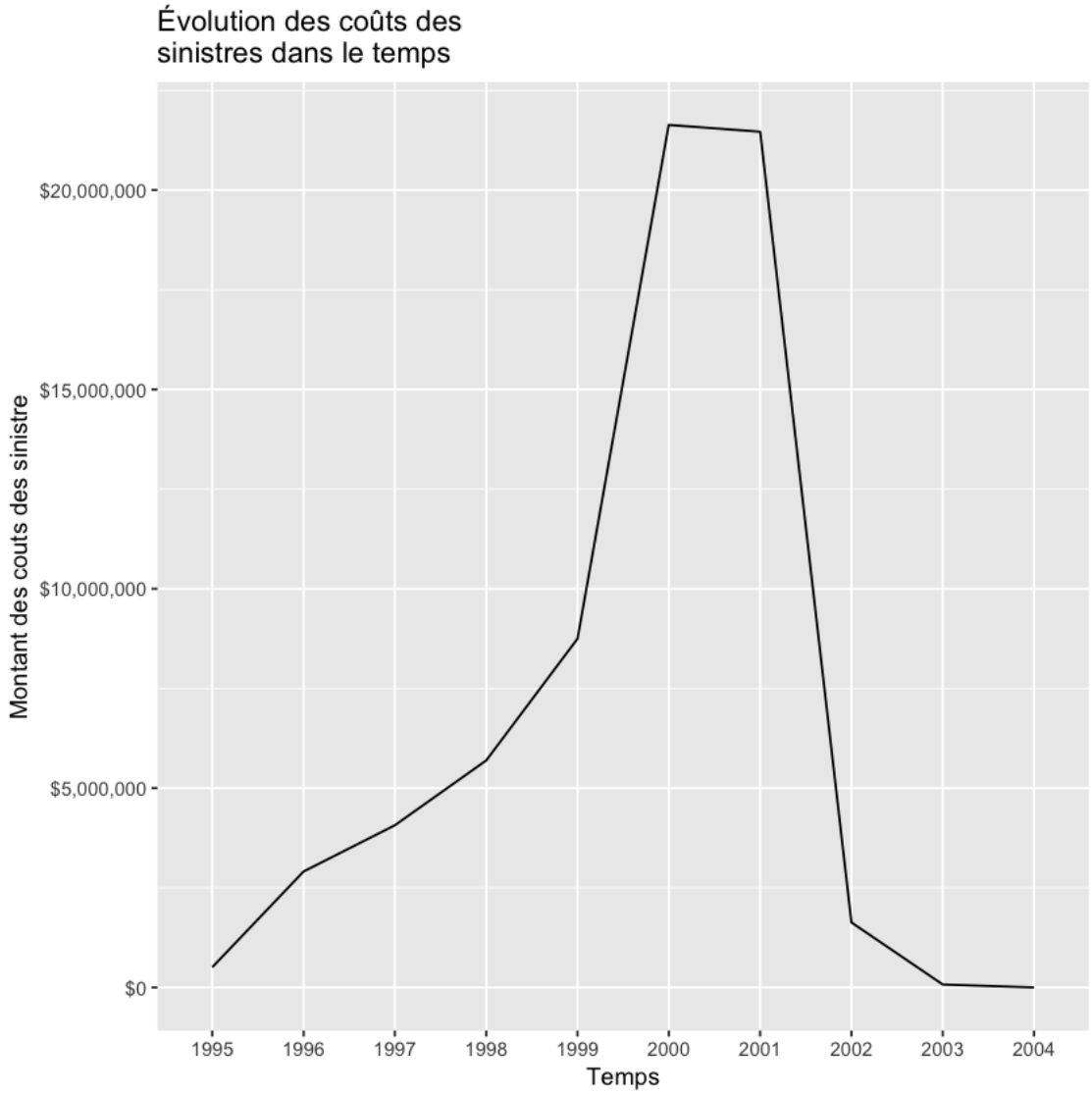
3.2 b)

Faites maintenant le même exercice en regroupant les coûts de sinistres par année. Vous devriez avoir le graphique suivant:

```
In [89]: police_assurance$sinistre_ann<-format(as.Date(police_assurance$debut_pol), "%Y")

In [90]: df_q6_b<-aggregate(police_assurance$somme_couts, by=list(sinistre_ann=police_assurance$

In [91]: ggplot(data=df_q6_b, aes(x=sinistre_ann, y=x, group=1)) +
             geom_line() +
             xlab("Temps") + ylab("Montant des couts des sinistre") +
             scale_y_continuous(labels = dollar)+
             ggtitle("Évolution des coûts des \nsinistres dans le temps")
```



4 Q4

4.1 a)

Faites le même exercice que la question 3b), mais cette fois, séparez les coûts en deux catégories; * les coûts de sinistres annuels pour les francophones * les coûts de sinistres annuels pour les anglophones

Faite un graphique qui contient deux lignes, une première qui représente les coûts de sinistre sur le temps pour les francophones, et l'autre ligne pour les anglophones.

```
In [92]: df_q4<-aggregate(police_assurance$somme_couts, by=list(sinistre_ann=police_assurance$si
```

```
In [93]: head(df_q4)
```

sinistre_ann	numeropol	x
1999	1	0.0000
2000	1	243.8571
1996	5	0.0000
1997	5	0.0000
1998	5	0.0000
1995	13	0.0000

```
In [94]: head(police_assurance)
```

numeropol	debut_pol	fin_pol	cout1	cout2	cout3	cout4	cout5	cout6	cout7	nbsin	a
1	1999-11-10	2000-10-16	NA	NA	NA	NA	NA	NA	NA	0	1
1	2000-10-17	2000-11-09	NA	NA	NA	NA	NA	NA	NA	0	2
1	2000-11-10	2001-11-09	243.8571	NA	NA	NA	NA	NA	NA	1	2
5	1996-01-03	1996-03-27	NA	NA	NA	NA	NA	NA	NA	0	1
5	1996-03-28	1997-01-02	NA	NA	NA	NA	NA	NA	NA	0	1
5	1997-01-03	1998-01-02	NA	NA	NA	NA	NA	NA	NA	0	1

```
In [95]: head(donnes_demo)
```

name	province	company	langue	date_naissance	agee	age_p
Shane Robinson	Nova Scotia	May Ltd	fr	1944-10-20	72	24
Courtney Nguyen	Saskatchewan	Foley, Moore and Mitchell	en	1985-12-09	31	24
Lori Washington	Yukon Territory	Robinson-Reyes	fr	1970-01-27	47	28
Sarah Castillo	Alberta	Wood, Brady and English	fr	2000-08-23	16	16
Jeffrey Garcia	Nunavut	Berger-Thompson	en	1969-10-25	47	20
Colleen Coleman	Saskatchewan	Simmons-Smith	en	1984-10-16	32	23

```
In [97]: library(dplyr)
```

```
In [98]: df_join<-left_join(df_q4,donnes_demo[, c("numeropol","langue")],by = "numeropol")
         head(df_join)
```

sinistre_ann	numeropol	x	langue
1999	1	0.0000	fr
2000	1	243.8571	fr
1996	5	0.0000	en
1997	5	0.0000	en
1998	5	0.0000	en
1995	13	0.0000	fr

Ensuite nous regroupons le tout par année de sinistre **et** langue parlée en faisant une somme sur la variable x de l'ancien data frame df_join

```
In [99]: df_join_sum<-aggregate(df_join$x,
                          by=list(sinistre_ann=df_join$sinistre_ann,
                                  langue=df_join$langue), FUN=sum)
```

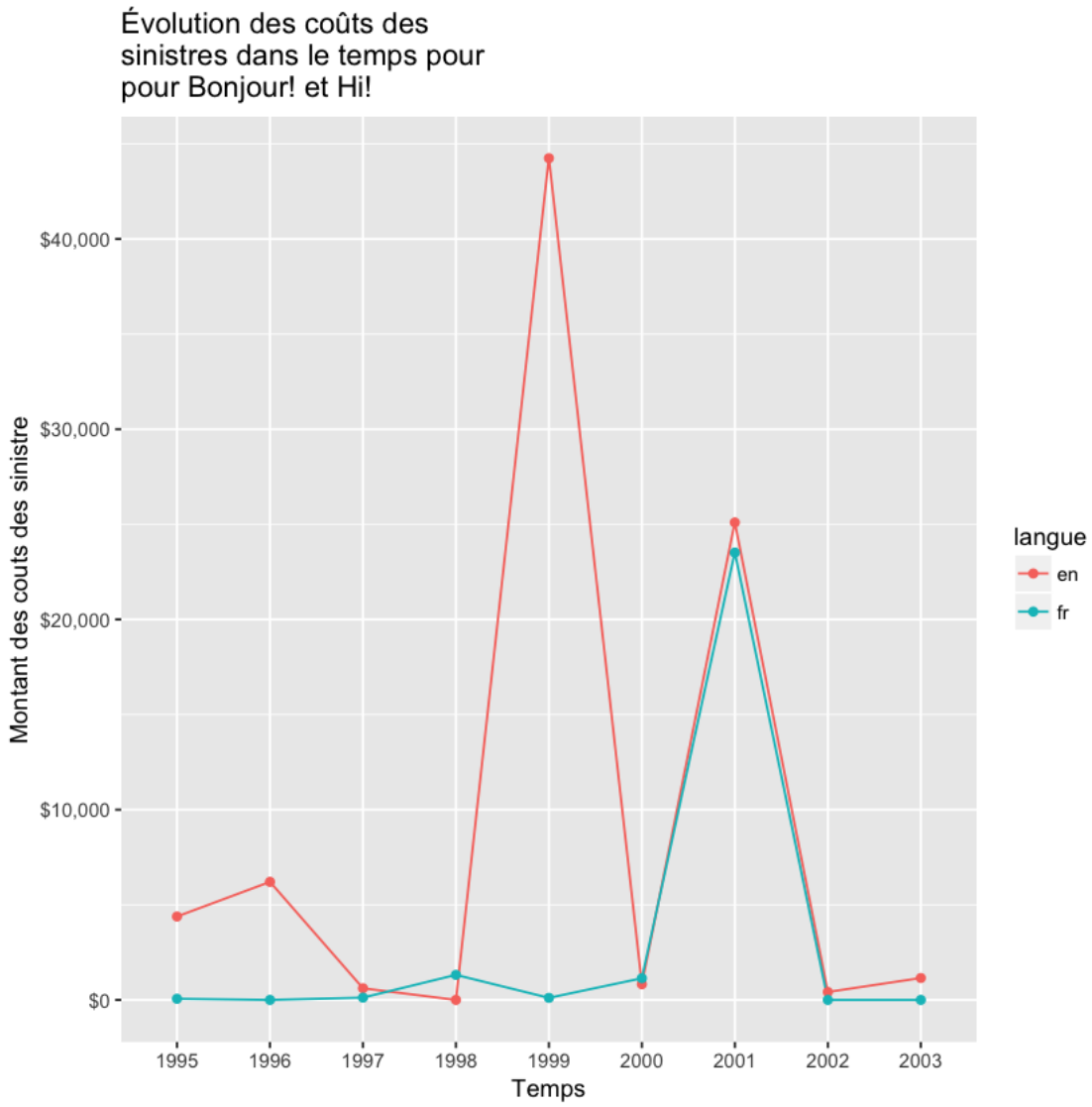
Remarquez la nouvelle variable x avec le total par année et par langue

```
In [100]: df_join_sum
```

sinistre_ann	langue	x
1995	en	4384.31056
1996	en	6205.98758
1997	en	611.27950
1998	en	0.00000
1999	en	44245.03106
2000	en	825.40373
2001	en	25096.28571
2002	en	416.44720
2003	en	1151.08696
1995	fr	60.50932
1996	fr	0.00000
1997	fr	120.46584
1998	fr	1316.48447
1999	fr	108.07453
2000	fr	1138.55280
2001	fr	23513.50311
2002	fr	0.00000
2003	fr	0.00000

Maintenant nous pouvons faire notre graphique ou notyre variable d'intérêt x est assignée aux deux groupes de langue `group=langue`. Nous distinguons ces deux groupes par couleur `colour=langue`

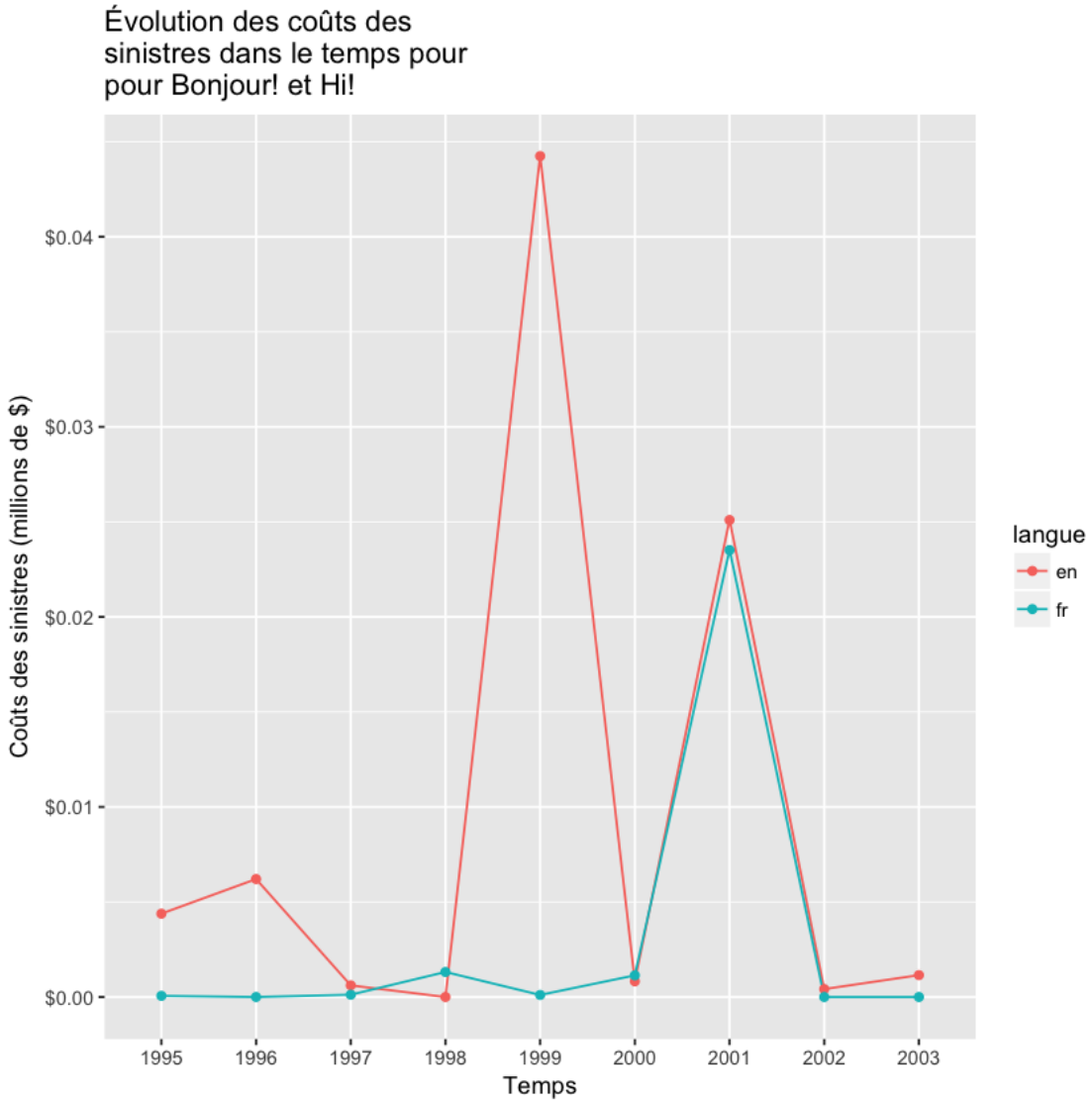
```
In [101]: ggplot(data=df_join_sum, aes(x=sinistre_ann, y=x, group=langue, colour=langue)) +
              geom_line() +
              geom_point()+    xlab("Temps") + ylab("Montant des couts des sinistre") +
              ggtitle("Évolution des coûts des \nsinistres dans le temps pour \npour Bonjour! et
              scale_y_continuous(labels = dollar)
```



On peut réduire le nombre de 0 sur notre axe des y

```
In [102]: million<-1000000

          ggplot(data=df_join_sum, aes(x=sinistre_ann, y=x/million, group=langue, colour=langue)
              geom_line() +
              geom_point()+    xlab("Temps") + ylab("Coûts des sinistres (millions de $)") +
              ggtitle("Évolution des coûts des \nsinistres dans le temps pour \npour Bonjour! et
              scale_y_continuous(labels = dollar)
```



5 Q5

On vous dit que la compagnie Discover, émettrice de cartes de crédit, a été achetée par le groupe Ironman. Faites la mise à jour de ces informations dans votre base de données. Mais n'oubliez pas de créer un backup de votre ancienne BD sous le format suivant `yyyy_mm_dd_HH_MM_SS.csv`(année, mois, jour, heure, minute et seconde).

```
In [107]: pmt_det<-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/pmt_details.
          head(pmt_det)
```

numeropol	cout_prime	credit_card_number	credit_card_provider	credit_card_expire	freq_pmt
1	1060.28	4.427476e+15	Voyager	04/23	12
5	1200.89	5.303389e+15	JCB 16 digit	08/26	1
13	940.54	3.528569e+15	Maestro	08/22	12
16	860.75	6.011570e+15	VISA 13 digit	03/23	1
22	790.17	5.262495e+15	Maestro	08/20	1
28	940.16	4.583364e+15	Discover	03/20	1

```
In [108]: date_heure<-Sys.time()
          date_heure
```

```
[1] "2018-04-01 17:15:19 EDT"
```

```
In [109]: date_heure<-gsub(":", "_", date_heure)
          date_heure<-gsub(" ", "_", date_heure)
          date_heure<-gsub("-", "_", date_heure)
```

```
In [110]: date_heure
```

'2018_04_01_17_15_19'

```
In [111]: nom_fichier<-paste("", date_heure, ".csv", sep = "")
          nom_fichier
```

'2018_04_01_17_15_19.csv'

```
In [112]: write.csv(pmt_det, nom_fichier)
```

```
In [113]: pmt_det$credit_card_provider <- replace(as.character(pmt_det$credit_card_provider),
                                                  pmt_det$credit_card_provider == "Discover", "I
```

```
In [114]: head(pmt_det)
```

numeropol	cout_prime	credit_card_number	credit_card_provider	credit_card_expire	freq_pmt
1	1060.28	4.427476e+15	Voyager	04/23	12
5	1200.89	5.303389e+15	JCB 16 digit	08/26	1
13	940.54	3.528569e+15	Maestro	08/22	12
16	860.75	6.011570e+15	VISA 13 digit	03/23	1
22	790.17	5.262495e+15	Maestro	08/20	1
28	940.16	4.583364e+15	Ironman	03/20	1

On vérifie qu'on a bien une valeur `Ironman`

```
In [115]: head(pmt_det[which(pmt_det$credit_card_provider=="Ironman"),])
```

	numeropol	cout_prime	credit_card_number	credit_card_provider	credit_card_expire	freq_pm
6	28	940.16	4.583364e+15	Ironman	03/20	1
15	69	720.57	4.530801e+15	Ironman	05/19	1
26	113	720.50	4.507907e+15	Ironman	03/21	1
32	126	960.24	5.127974e+15	Ironman	05/20	1
34	136	980.10	4.635091e+15	Ironman	04/20	1
43	172	930.86	5.408687e+15	Ironman	03/24	1

et que la valeur `Discover` n'éxiste plus

In [116]:
```
pmt_det[which(pmt_det$credit_card_provider=="Discover"),]
```

numeropol	cout_prime	credit_card_number	credit_card_provider	credit_card_expire	freq_pmt

6 Q6

6.1 a)

Faites un graphique du prix du bitcoin sur la période allant du 2016-12-04 au 2017-12-04. Vous pouvez lire ces données ici.

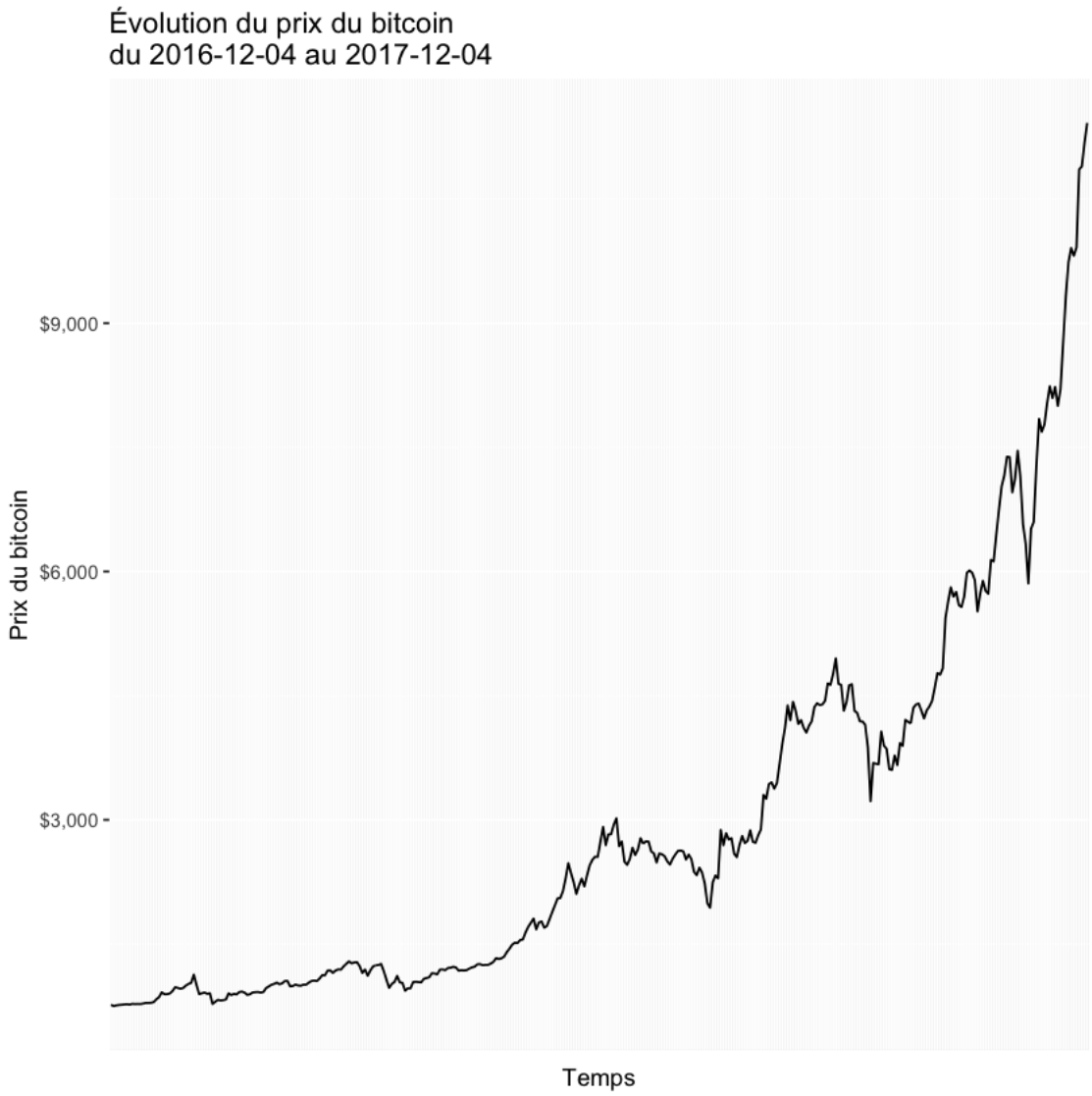
In [120]:
```
bitCoin<-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/bitcoin_pric
head(bitCoin)
```

Date	Close.Price
2016-12-04 0:00	766.46
2016-12-05 0:00	750.71
2016-12-06 0:00	758.81
2016-12-07 0:00	763.90
2016-12-08 0:00	766.75
2016-12-09 0:00	770.41

In [121]:
```
tail(bitCoin)
```

	Date	Close.Price
361	2017-11-29 0:00	9816.35
362	2017-11-30 0:00	9916.54
363	2017-12-01 0:00	10859.56
364	2017-12-02 0:00	10895.01
365	2017-12-03 0:00	11180.89
366	2017-12-04 14:52	11420.50

In [122]:
```
ggplot(data=bitCoin, aes(x=Date, y=Close.Price, group=1)) +
    geom_line() +
    xlab("Temps") + ylab("Prix du bitcoin") +
    scale_y_continuous(labels = dollar)+
    ggtitle("Évolution du prix du bitcoin \ndu 2016-12-04 au 2017-12-04")+
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank())
```



6.2 b)

Sauvegardez le graphique dans un fichier `.png` sous le format de 5" de largeur et 3" de hauteur dans le répertoire courant (*working directory*)

```
In [123]: ggsave("bitcoinProce.png",width = 5, height = 3)
```

6.3 c)

Calculer le rendement quotidien que vous avez fait depuis l'achat de votre bitcoin. On se rappelle que le rendement quotidien se calcule comme suit;

$$r = \frac{V_f - V_i}{V_i}$$

```
In [124]: head(bitCoin)
```

Date	Close.Price
2016-12-04 0:00	766.46
2016-12-05 0:00	750.71
2016-12-06 0:00	758.81
2016-12-07 0:00	763.90
2016-12-08 0:00	766.75
2016-12-09 0:00	770.41

```
In [125]: head(diff(bitCoin$Close.Price)/bitCoin$Close.Price[-length(bitCoin$Close.Price)])
```

1. -0.0205490175612556 2. 0.010789785669566 3. 0.00670787153569409 4. 0.00373085482392986 5. 0.00477339419628297 6. 0.00363442842123034

Remarquez que si l'on voulait insérer ce qu'on vient de faire `diff(...)` dans une nouvelle colonne de notre df `bitCoin`, ça n'aurait pas pu fonctionner car nous avons calculer 365 valleurs alors que notre df possède 366 observations. Puisqu'au temps $t = 0$, nous n'avons aucun rendement encore.

Alors il faut mettre le rendemnet au tempos $t = 0$ à `null`

```
In [126]: bitCoin$rate_return<-c(NA, diff(bitCoin$Close.Price)/bitCoin$Close.Price[-length(bitCo
```

Le package Quantmod contient une fonction *built in* qui calcul le rendement. Cette fonction est appelée `Delt`

```
In [128]: require(quantmod)
```

```
In [129]: bitCoin$rate_return<-Delt(bitCoin$Close.Price)
          head(bitCoin)
```

Date	Close.Price	rate_return
2016-12-04 0:00	766.46	NA
2016-12-05 0:00	750.71	-0.020549018
2016-12-06 0:00	758.81	0.010789786
2016-12-07 0:00	763.90	0.006707872
2016-12-08 0:00	766.75	0.003730855
2016-12-09 0:00	770.41	0.004773394

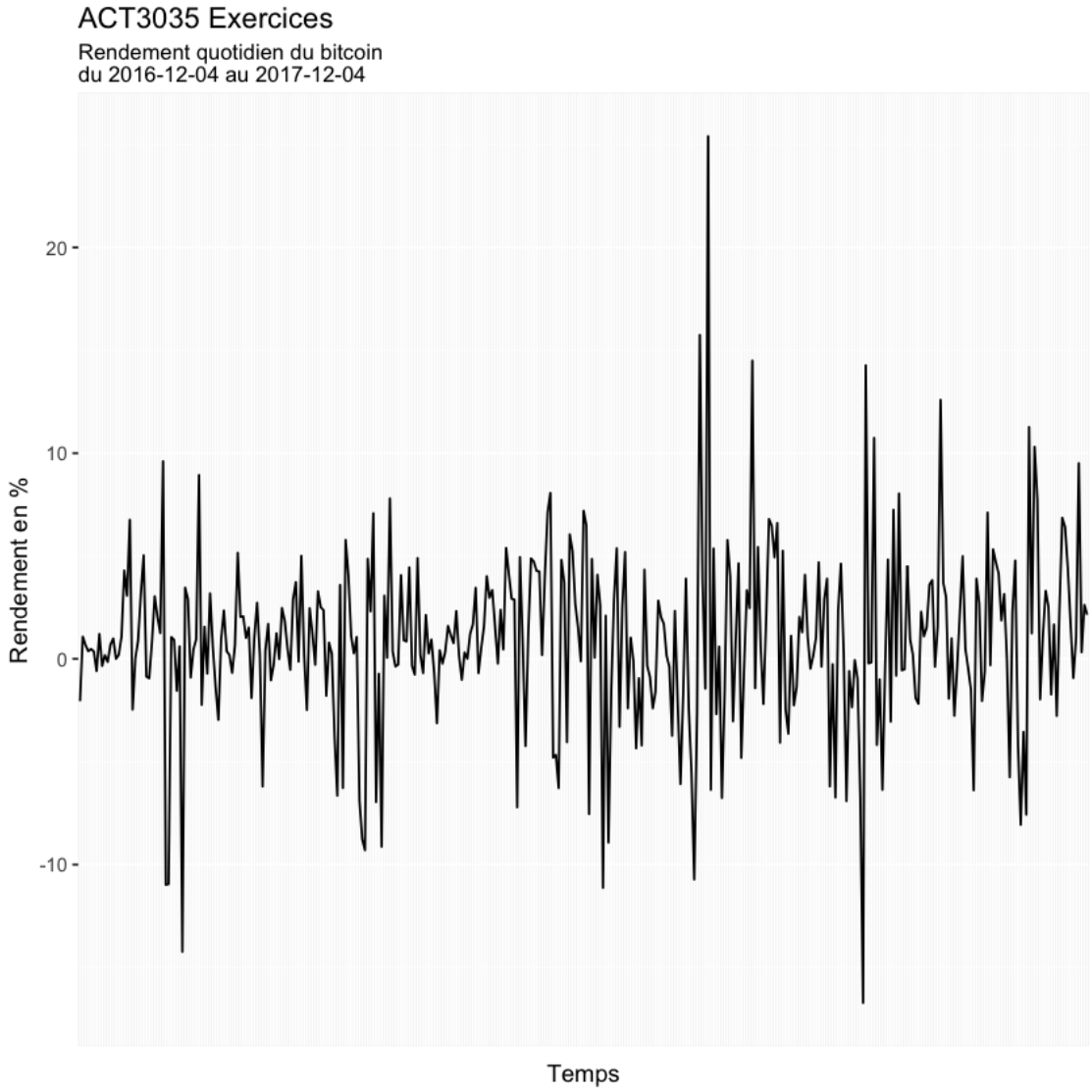
Remarquez qu'elle insère un rendement=na au temps $t = 0$

```
In [130]: bitCoin_2<-bitCoin[-1,] # on se débarasse de la première observation
          head(bitCoin_2)
```

	Date	Close.Price	rate_return
2	2016-12-05 0:00	750.71	-0.020549018
3	2016-12-06 0:00	758.81	0.010789786
4	2016-12-07 0:00	763.90	0.006707872
5	2016-12-08 0:00	766.75	0.003730855
6	2016-12-09 0:00	770.41	0.004773394
7	2016-12-10 0:00	773.21	0.003634428

Et on fait notre graphique

```
In [131]: ggplot(bitCoin_2, aes(x=bitCoin_2$Date, group=1)) +
            geom_line(aes(y=bitCoin_2$rate_return*100)) +
            labs(title="ACT3035 Exercices",
                 subtitle="Rendement quotidien du bitcoin \ndu 2016-12-04 au 2017-12-04",
                 y="Rendement en %")+ xlab("Temps")+
              theme(axis.text.x = element_blank(),
                    axis.ticks.x = element_blank())
```



7 Q7

Faites un graphique sur la corrélation entre les prix d'action venant des données suivantes

```
In [134]: library(ggcorrplot)
```

```
In [135]: df_app <-read.csv("https://raw.githubusercontent.com/nmeraihi/data/master/stocks_corre
          mat_corr<-cor(df_app)

In [136]: ggcorrplot(mat_corr, hc.order = TRUE,
              lab = TRUE)
```