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)
```

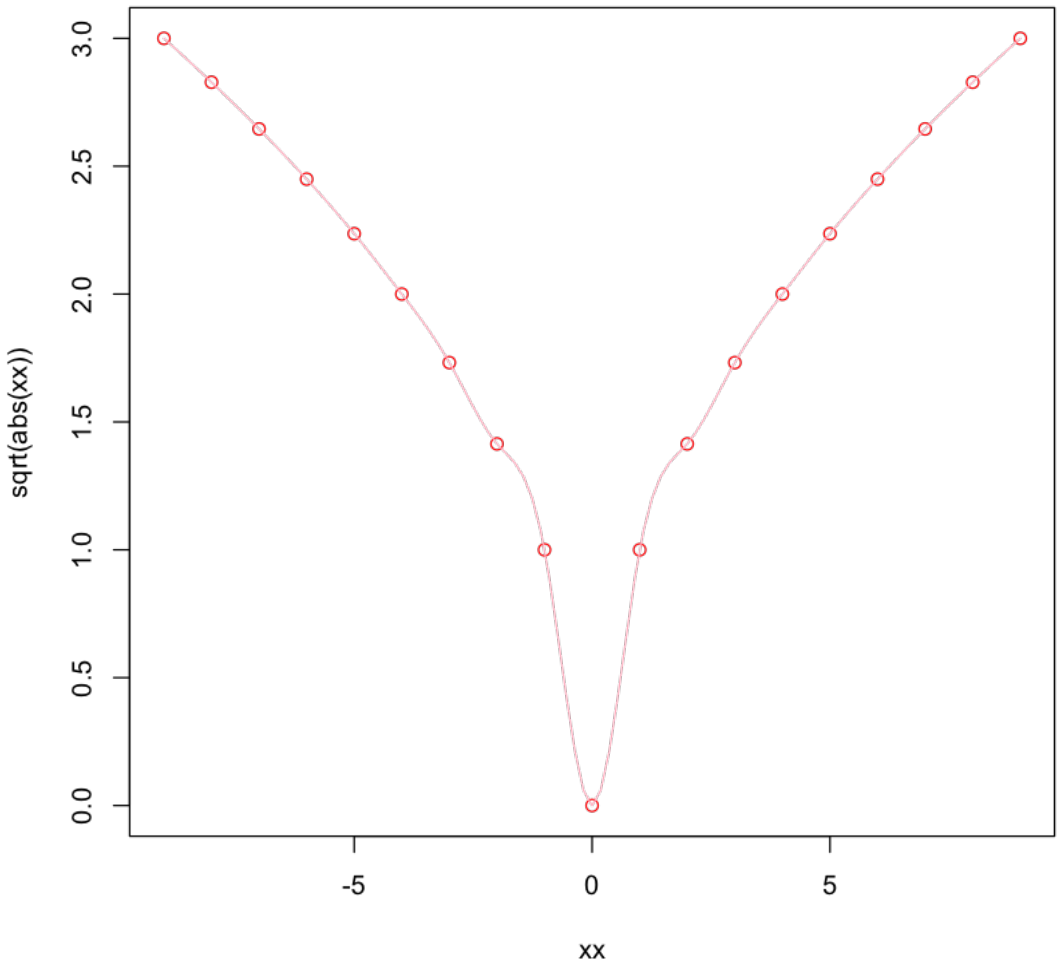
```
sqrt> require(stats) # for spline

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
```

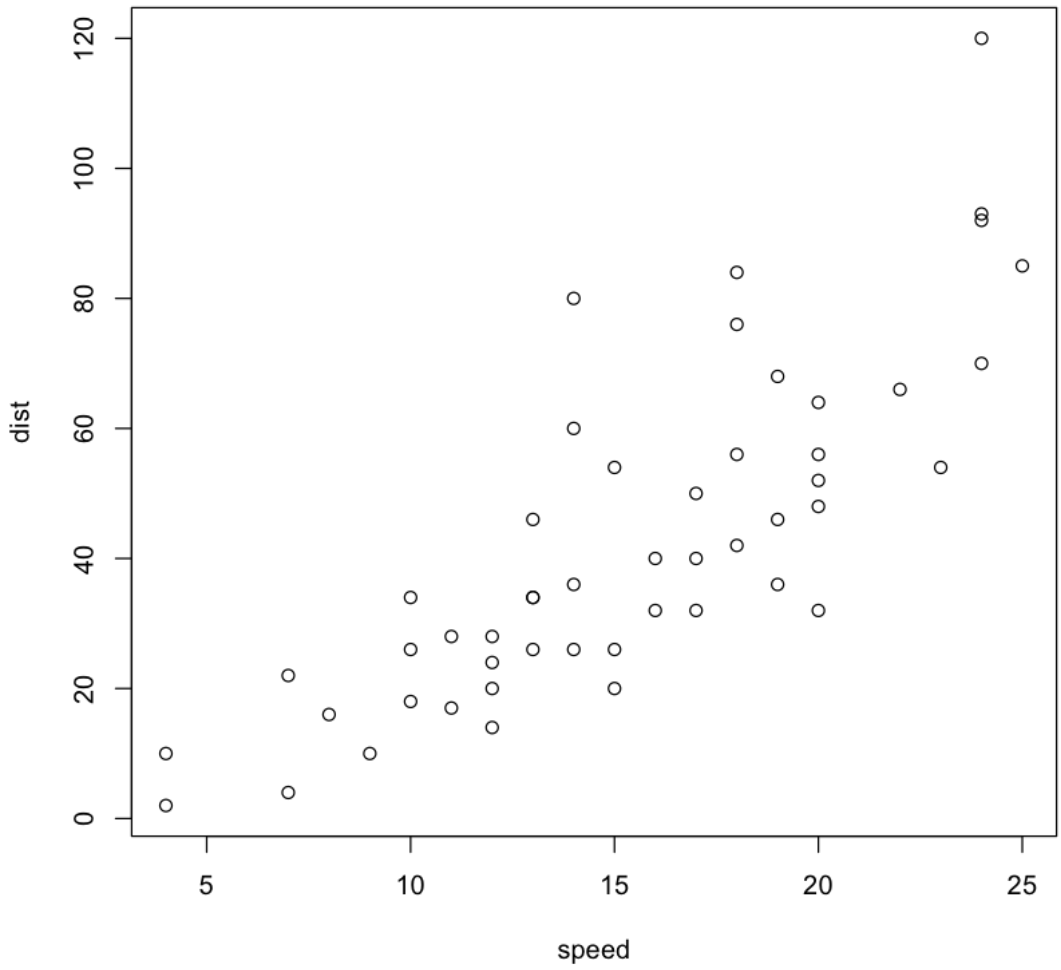
7.3 Aide sur les données

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