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