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
```