Manipuler facilement vos formats date avec lubridate

Alioune SALL et Arona DIOP

2017-04-19

Table des matières

Introduction

L'analyse de données contenant des dates peut être un véritable défi si on n'est pas doté d'un outil performant. En effet, les données sur les dates peuvent être de différents formats, ce qui peut créer des erreurs si on n'a pas le programme adapté. Aussi, on pourrait être intéressé par le passage d'un fuseau horaire à un autre ou encore les zones qui appliquent l'heure d'été et celles qui ne l'appliquent pas. Les opérations arithmétiques avec les dates ne sont pas faciles. Cette difficulté résulte du fait que la durée d'une année, d'un mois, ... change en raison, entre autres, des secondes intercalaires.

Bien que le `R` de base permet de résoudre certains de ces problèmes, la syntaxe peut être difficile à retenir et elle peut changer selon la classe de l'objet utilisé. C'est pour remédier, en partie, à ces difficultés que le package `lubridate` a été créé. Il fournit également de nouveaux outils pour analyser les données de type date.

Différentes fonctionnalités de `lubridate`

Le package `lubridate` offre une panoplie de fonctionnalités qui rendent plus conviviales les opérations effectuées sur les données de type date. En effet, avec ses plus de 50 fonctions, `lubridate` constitue un outil très puissant et efficace pour manipuler les dates (temps, période, intervalle de temps, laps de temps, ...).

Dans cette section, il sera présenté les grandes fonctionnalités de ce package.

Analyse de données

`lubridate` vient avec des fonctions très intuitives. Ainsi, la fonction `today()` fournit l'information sur la date d'aujourd'hui et la commande `now()` apporte une précision supplémentaire quant à l'heure exacte.

```
today()
```

```
## [1] "2019-05-30"
```

```
now()
```

```
## [1] "2019-05-30 09:12:28 EDT"
```

Pour analyser une donnée de type date, il faut d'abord identifier l'ordre dans lequel l'année, le mois et le jour apparaissent dans la chaîne de caractères et ensuite mettre les lettres `y` (année), `m` (mois) et `d` (jours) dans le même ordre que notre chaîne de caractères. Par exemple avec cette procédure, nous pouvons obtenir l'une des fonctions suivantes : `ymd()`, `ydm()` , `dmy()` , `dym()` , etc.

Pour voir comment fonctionnent ces fonctions, nous allons essayer avec la date *2017-03-21*. Nous devons entourer la date par des guillemets.

```
ymd("2017-03-21")
```

```
## [1] "2017-03-21"
```

Bien vrai que la date *2017-03-21* est donnée dans un format assez standard, `lubridate` est aussi en mesure d'interpréter et de comprendre différents formats de date et d'heure. Nous utilisons `ydm()` pour analyser *2017 17 Mars*.

```
ydm("2017 17 Mars")
```

```
## [1] "2017-03-17"
```

Ci-après, quelques illustrations avec d'autres types de formats sont présentées :

```
ymd("2017 Mars, 17")
```

```
## [1] "2017-03-17"
```

```
myd("Mars, 2017, 1")
```

```
## [1] "2017-03-01"
```

```
dmy("13082017")
```

```
## [1] "2017-08-13"
```

```
dmy("150118")
```

```
## [1] "2018-01-15"
```

Les fonctions de type `ymd()` existent également avec les heures, les minutes et les secondes. En effet, de manière similaire, en combinant les lettres "h" (heure), "m" (minute) et "s" (seconde), nous pouvons obtenir les fonctions `hms()` , `ms()` , `hm()` , etc.

```
hms("05:34:43")
```

```
## [1] "5H 34M 43S"
```

```
hm("10.44")
```

```
## [1] "10H 44M 0S"
```

Il est possible d'analyser à la fois des formats contenant des dates et heures. Pour ce faire, il suffit de regrouper les fonctions dates et les fonctions heures à l'aide de `"_"`. Pour lire les dates avec un certain fuseau horaire, fournissez le nom officiel de ce fuseau horaire dans l'argument `tz`. Pour avoir plus de détails sur les noms des fuseaux horaires, visitez le lien suivant http://en.wikipedia.org/wiki/List_of_tz_database_time_zones.

```r
ymd_hms("2017/05/12 00:34:43")
```

```
## [1] "2017-05-12 00:34:43 UTC"
```

```r
ymd_hms("2017/09/21 10:34:43", tz = "America/Toronto")
```

```
## [1] "2017-09-21 10:34:43 EDT"
```

```r
ymd_hms("2017/01/29 5:34:43", tz = "Africa/Abidjan")
```

```
## [1] "2017-01-29 05:34:43 GMT"
```

Ces fonctions créent un objet `POSIXct date-time` qui correspond à la date décrite par la chaîne de caractères référencée en argument. Elles reconnaissent automatiquement les séparateurs couramment utilisés pour enregistrer les dates : `"/"` ; `"-"` ; `","` ; `"."` et `""` (c'est-à-dire aucun séparateur).

Quand une fonction `mdy()` est appliquée à un vecteur de dates, `lubridate` supposera que toutes les dates stockées dans ce vecteur ont le même ordre (dans notre exemple, l'ordre est mois -> jour -> année) et ceci fonctionne quels que soient les séparateurs utilisés.

```r
x <- c('February 20 2016',
"01/14/2017",
"110415",
'Feb 20th 19')
mdy(x, locale = "English")
```

```
## [1] "2016-02-20" "2017-01-14" "2015-11-04" "2019-02-20"
```

Cet exemple a aussi servi à illustrer l'argument `locale`, qui permet d'utiliser dans les noms de mois une autre langue que celle du système d'exploitation.

Manipulation de données

Une donnée de type *date-time* est une combinaison de différents éléments et chacun vient avec sa propre valeur. La plupart du temps, une date est composée par une valeur qui renseigne sur l'année, une autre valeur sur le mois, celle qui indique le jour et ainsi de suite. Avec `lubridate`, nous pouvons extraire n'importe laquel de ces composants en utilisant la fonction qui porte son nom (voir illustration sur le tableau 1). Ces fonctions rendent simple l'analyse de n'importe quel objet de type *date-time*.

Table 1: Extraction des composants d'une donnée de type date.

Composants	Fonction pour l'extraire
Année	year()
Mois	month()
Semaine	week()
Jour de l'année	yday()
Jour du mois	mday()
Jours de semaine	wday()
Heure	hour()
Minute	minute()
Seconde	second()
Fuseau horaire	tz()

Pour illustrer ce principe, nous utilisons l'exemple suivant qui utilise l'heure exacte du système au moment de l'exécution :

```r
maintenant<-now()
maintenant
```

```
## [1] "2019-05-30 09:12:28 EDT"
```

```r
year(maintenant)
```

```
## [1] 2019
```

```r
month(maintenant)
```

```
## [1] 5
```

```r
day(maintenant)
```

```
## [1] 30
```

```r
wday(maintenant)
```

```
## [1] 5
```

L'argument `label` contient les noms de certains composants en anglais.

```r
month(maintenant,label = TRUE)
```

```
## [1] May
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
```

```r
month(maintenant,label = TRUE ,abbr = FALSE)
```

```
## [1] May
## 12 Levels: January < February < March < April < May < June < ... < December
```

```r
wday(maintenant,label = TRUE, abbr = FALSE)
```

```
## [1] Thursday
## 7 Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < ... < Saturday
```

Opérations arithmétiques

Effectuer des opérations arithmétiques avec des données de type *date-time* est plus compliquée que de l'arithmétique appliquée sur des nombres. Qu'est-ce qui complique l'arithmétique avec les dates ? En effet, il y a beaucoup de choses à prendre à la fois en compte notamment :

- les conditions astronomiques,
- l'inclinaison de la Terre sur son axe par rapport au soleil,
- l'heure d'été,
- les années bissextiles,
- la seconde intercalaire, également appelée saut de seconde ou seconde additionnelle,
- etc.

Faire des calculs sur des dates tout en prenant en compte un de ces paramètres pourrait être une opération très lourde et fastidieuse. Heureusement, avec `lubridate`, ces calculs peuvent être effectués avec précision et facilité.

Si la date de remise de ce TP était le 1er janvier 2017 et que nous voudrions savoir quel jour il ferait un an à partir de cette date, nous pourrions simplement ajouter une année.

```
ymd(20170101) + years(1)
```

```
## [1] "2018-01-01"
```

```
ymd(20170101) + dyears(1)
```

```
## [1] "2018-01-01"
```

Reproduisons les opérations précédentes en prenant maintenant comme date de remise de ce TP le 1er janvier 2016.

```
ymd(20160101) + years(1)
```

```
## [1] "2017-01-01"
```

```
ymd(20160101) + dyears(1)
```

```
## [1] "2016-12-31"
```

Nous remarquons qu'en ajoutant 365 jours à la date du 1er janvier 2016, nous obtenons le 31 décembre 2016. Ceci est dû au fait que 2016 est une année bissextile (en anglais : leap year).

```
leap_year(2016)
```

```
## [1] TRUE
```

```
leap_year(2017)
```

```
## [1] FALSE
```

En effet à des moments différents, la longueur des mois, des semaines, des jours, des heures et même des minutes varieront également. Ils sont considérés comme étant des unités relatives de temps. En revanche, les secondes ont toujours une longueur constante. Par conséquent, elles sont des unités de temps exactes. `lubridate` permet d'effectuer des opérations avec des unités relatives de temps et celles exactes grâce à l'introduction de quatre nouveaux objets liés au temps :

- `instants`

Les instants sont des moments précis. La fonction `now()` retourne des objets de cette classe. Avec l'aide de la fonction `is.instant()`, nous pouvons tester si une date représente un instant ou pas.

```
maintenant
```

```
## [1] "2019-05-30 09:12:28 EDT"
```

```
is.instant(maintenant)
```

```
## [1] TRUE
```

```
is.instant(234)
```

```
## [1] FALSE
```

- `intervals`

Les `intervals` sont des intervalles de temps qui commencent et finissent à des instants bien déterminés (parce qu'ils sont liés à des dates spécifiques). Les intervalles conservent des informations complètes sur un intervalle de temps. Les fonctions pour travailler avec l'objet `intervals` sont `is.interval`, `as.interval`, `interval`, `int_start`, `int_end`,`int_shift`, `int_flip`, `int_aligns`, `int_overlaps` et `%within%`. Les intervalles peuvent également être manipulés avec les fonctions suivantes : `intersect`, `union`, et `setdiff`.

```
date_depart <- dmy_hm("27/12/2016 5:45", tz="Africa/Dakar")
date_arrive <- mdy_hm("dec 28, 2017  19:30", tz="America/Toronto", locale = "English")
duree <- interval(date_depart, date_arrive)
duree
```

```
## [1] 2016-12-27 05:45:00 GMT--2017-12-29 00:30:00 GMT
```

- `periods`

Les `periods` mesurent la variation de temps entre deux instants. Ils fournissent des prédictions robustes sur la date et prennent même en compte le passage à l'heure d'été, les années bissextiles, la seconde intercalaire. Les fonctions pour travailler avec les `periods` incluent `is.period`, `as.period` et `period`.

Pour plus de commodité, nous pouvons créer une période qui utilise les fonctions suivantes : `seconds`, `minutes`, `hours`, `days`, `weeks`, `months`, ou `years`.

```
as.period(duree)
```

```
## [1] "1y 0m 1d 18H 45M 0S"
```

- `durations`

Les `durations` mesurent le temps exact qui s'est écoulé entre deux instants. Cependant lorsque, l'un des évènements suivants : le passage à l'heure d'été, les années bissextiles ou la seconde intercalaire survient entre les deux instants, le résultat obtenu devient moins précis. Les fonctions pour travailler avec les `durations` sont `is.duration`, `as.duration` et `duration`. Pour plus de commodité, nous pouvons également créer une durée en utilisant les fonctions suivantes : `dseconds`, `dminutes`, `dhours`, `ddays`, `dweeks`, ou `dyears`.

Apport par rapport au R de base

La manipulation des dates avec le `R` de base est parfois difficile voire impossible pour réaliser certaines tâches. De plus, dans les cas où `R` est capable de le faire, le code est long, ce qui le rend difficile à être reproduit. `lubridate` est une librairie qui nous permet de pallier à ces insuffisances. Les exemples suivants nous permettront de saisir la différence de ce package avec les fonctionnalités de `R` de base.

Exemple 1 : création de date

- avec R de base

```
dateR  <-  as.POSIXct("01-01-2017", format  =  "%d-%m-%Y", tz="UTC")
dateR
```

```
## [1] "2017-01-01 UTC"
```

- avec `lubridate`

```
dateLubridate  <-  dmy("01-01-2017", tz="UTC")
dateLubridate
```

```
## [1] "2017-01-01 UTC"
```

Exemple 2 : changer la valeur du mois

- avec `R` de base

```
dateR  <-  as.POSIXct(format(dateR, "%Y-2-%d"),  tz  =  "UTC")
dateR
```

```
## [1] "2017-02-01 UTC"
```

- avec `lubridate`

```
month(dateLubridate)  <-  2
dateLubridate
```

```
## [1] "2017-02-01 UTC"
```

`lubridate` ne nous simplifie pas seulement les codes, mais nous permet de faire plus que le `R` de base. En effet, dans l'exemple qui suit , `lubridate` nous permet d'avoir directement l'âge en tenant compte des années bissextiles.

Exemple 3 : Calcul d'âge exact

- tentative avec `R`

Essayons de calculer l'âge avec le `R` de base en divisant le nombre de jours par 365

```
anni <- as.Date("31/12/2020", format="%d/%m/%Y")
bd <- as.Date("01/01/2000", format="%d/%m/%Y")
age <- as.numeric((anni-bd)/365)
age
```

```
## [1] 21.0137
```

Cette valeur n'est pas exacte puisque l'individu n'a pas encore fêté son 21^{e} anniversaire, cela est dû au fait que, en moyenne, une année dure 365.25 et non 365.

```
anni <- as.Date("31/12/2020", format="%d/%m/%Y")
bd <- as.Date("01/01/2000", format="%d/%m/%Y")
age <- as.numeric((anni-bd)/365.25)
age
```

```
## [1] 20.99932
```

Si on voudrait maintenant connaitre l'âge au 01-01-1903 d'un individu né en le 01-01-1900, on aurait :

```
anni <- as.Date("01/01/1903", format="%d/%m/%Y")
bd <- as.Date("01/01/1900", format="%d/%m/%Y")
age <- as.numeric((anni-bd)/365.25)
age
```

```
## [1] 2.997947
```

Or cet individu a exactement 3 ans le 01/01/1903.

Ces différences sont dues au fait que certaines années sont de 365 jours et d'autres 366. Pour avoir le nombre d'années exact, la fonction `time_length` de `lubridate` nous permet de représenter la différence en années tout en prenant en compte les années bissextiles.

- Solution avec `lubridate`

```
bd <- ymd("1900-01-01")
anni <- ymd("1903-01-01")
age <- time_length(interval(bd, anni), "years")
age
```

```
## [1] 3
```

Forces et faiblesses

Forces

Dans tout le document, il a été question de montrer les caractères intuitifs des fonctions du package `lubridate` . En effet, il permet une manipulation simple des dates, comme une arithmétique avec les nombres, ce qui le rend simple à utiliser. Selon les auteurs de cet formidable outil qui sont *Garrett Grolemund* et *Hadley Wickham* , *"`lubridate` has a consistent, memorable syntax, that makes working with dates fun instead of frustrating."*

Faiblesses

Mise à jour 30 mai 2019 :

> **Les dysfonctionnements dans certains cas spécifiques des fonctions `dmy_h` et `duration` qui avaient été relevés en 2017 sont maintenant corrigés (`lubridate` version 1.7.4).**

Text original accompagné des sorties mises à jour (maintenant non problématiques) :

Comme toute oeuvre humaine, ce puissant outil présente quelques failles qui mériteraient d'être corrigées dans les prochaines versions. En effet, nous remarquons qu'il existe un problème avec certaines fonctions :

- dysfonctionnement dans la fonction `dmy_h`

Il semble avoir un dysfonctionnement dans la création de cette fonction `dmy_h` .Cette fonction retourne un sulcature exact pour le *jour* et le *mois*. Par contre, elle donne une mauvaise information sur l'*année* et elle récupère d'ailleurs les deux dernières positions de l'année pour les affecter à la place des heures et positionne les heures à la place réservée aux minutes.

```
dmy_h("05-07-2017 13")
```

```
## [1] "2017-07-05 13:00:00 UTC"
```

- problème dans le traitement des valeurs décimales

On constate une limitation du mécanisme d'analyse de l'unité. Les unités fractionnaires ne sont pas encore reconnues. Le résultat retourné est incohérent lorsqu'on utilise les exemples suivants :

```
duration("0 hours 30 min")
```

```
## [1] "1800s (~30 minutes)"
```

```
duration("1.5 hours")
```

```
## [1] "5400s (~1.5 hours)"
```

Bibliographie

- Référence du package `lubridate` :https://cran.r-project.org/web/packages/lubridate/lubridate.pdf
- G.Grolemund, H. Wickham. *Dates and Times Made Easy with lubridate*, Journal of statistical software. April 2011, Volume 40, Issue 3. https://www.jstatsoft.org/article/view/v040i03
- https://www.r-statistics.com/2012/03/do-more-with-dates-and-times-in-r-with-lubridate-1-1-0/
- http://biostat.mc.vanderbilt.edu/wiki/pub/Main/ColeBeck/datestimes.pdf
- https://rpubs.com/davoodastaraky/lubridate
- https://github.com/hadley/lubridate/issues?q=is%3Aissue+is%3Aopen+label%3Abug