Espace et Temps avec R

2 - Manipulation de données temporelles

Robin Cura, Hélène Mathian & Lise Vaudor

16/10/2018

École Thématique GeoViz 2018

Sommaire

Manipuler des données de type date-time

- Convertir une chaîne de caractères en date-time
- Arrondir des données date-time
- Durées, périodes, intervalles

Visualiser des données temporelles

- Visualisation simple
- Visualisation d'une tendance
- Visualisation de données agrégées
- Visualisation d'une périodicité, d'un intervalle

Manipulation de données temporelles avec R : `lubridate`

Pour manipuler des données temporelles avec R, on va utiliser le package `lubridate`, qui fait partie des packages du tidyverse et comprend des fonctions.

On utilisera aussi `dplyr` et `ggplot`. On peut donc charger directement le `tidyverse`...

```
library(tidyverse)
library(lubridate)
```

Convertir une chaîne de caractères en date-time

Date:

Il suffit de préciser l'ordre des éléments à travers le nom de la fonction:

```
ymd("2019/04_11")
```

```
## [1] "2019-04-11"
```

```
dmy("11 Avril 2019")
```

```
## [1] "2019-04-11"
```

```
mdy("April 11th, 2019")
```

```
## [1] "2019-04-11"
```

Date-time: Même principe!

```
ymd_hm("2019.04.11 14h37")
```

```
## [1] "2019-04-11 14:37:00 UTC"
```

```
ymd_hms("20190407143752")
```

```
## [1] "2019-04-07 14:37:52 UTC"
```

```
hms("14h37min52s")
```

```
## [1] "14H 37M 52S"
```

Extraire un des composants d'une date-time

Examinons par exemple la date-time suivante:

```
t
```

```
## [1] "2019-04-11 14:37:52 UTC"
```

On peut extraire un des composants de la date ou date-time à travers une série de fonctions du type `unitedetemps()`.

Par exemple:

```
date(t)
```

```
## [1] "2019-04-11"
```

```
hour(t)
```

```
## [1] 14
```

```
minute(t)
```

```
## [1] 37
```

```
second(t)
```

```
## [1] 52
```

Arrondir une date ou date-temps

Il est possible d'arrondir une date ou date-temps

- vers la valeur la plus proche (`round_date()`)
- vers le haut (`ceiling_date()`)
- vers le bas (`floor_date()`)

```
t
```

```
## [1] "2019-04-11 14:37:52 UTC"
```

```
round_date(t,"hour")
```

```
## [1] "2019-04-11 15:00:00 UTC"
```

On peut arrondir à l'unité de son choix

- `second`, `minute`, ou `hour`,
- `day`, `month`, ou `year`
- ou encore: `week`, `bimonth`, `quarter`, `season`, et `halfyear`

```
round_date(t,"day")
```

```
## [1] "2019-04-12 UTC"
```

```
round_date(t,"year")
```

```
## [1] "2019-01-01 UTC"
```

Calculer des durées ou périodes

```
t1 <- dmy("17/07/2018")
t2 <- dmy("17/04/2019")
diff <- t2-t1
diff
```

```
## Time difference of 274 days
```

`diff` correspond à une différence entre t1 et t2. Il s'agit d'un objet de classe `difftime`

```
# tps "physique"
as.duration(diff)
```

```
## [1] "23673600s (~39.14 weeks)"
```

```
# tps "social"
as.period(diff)
```

```
## [1] "274d 0H 0M 0S"
```

- durées: `dxxx()` (par exemple `ddays()` ou `dyears()`)

```
t1 + dyears(10)
```

```
## [1] "2028-07-14"
```

- périodes: fonctions `xxx()` (par exemple `days()` ou `months()`)

```
t1 + years(10)
```

```
## [1] "2028-07-17"
```

Créer des séquences et intervalles de temps

Création de séquences à travers les commandes:

- `dxxx(seq(...,...,...))`
- `xxx(seq(...,...,...))`

```
t0
```

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

```
t0 + dminutes(seq(from = 0, to = 45, by = 15))
```

```
## [1] "2018-01-01 00:00:00 UTC" "2018-01-01 00:15:00 UTC"
## [3] "2018-01-01 00:30:00 UTC" "2018-01-01 00:45:00 UTC"
```

Création d'intervalles:

```
itv <- interval(t0 + dminutes(seq(from = 0, to = 45, by = 15)),
                t0+dminutes(seq(from = 0, to = 45, by = 15)))
# commande equivalente: ...%--%...
```

Déterminer l'occurrence d'un événement dans un intervalle

Disposer d'un intervalle, cela permet de réaliser certaines opérations, comme (par exemple) déterminer si une date-time donnée fait partie de l'intervalle:

```
itv
```

```
## [1] 2018-01-01 00:00:00 UTC--2018-01-01 00:00:00 UTC
## [2] 2018-01-01 00:15:00 UTC--2018-01-01 00:15:00 UTC
## [3] 2018-01-01 00:30:00 UTC--2018-01-01 00:30:00 UTC
## [4] 2018-01-01 00:45:00 UTC--2018-01-01 00:45:00 UTC
```

```
t
```

```
## [1] "2018-01-01 00:17:45 UTC"
```

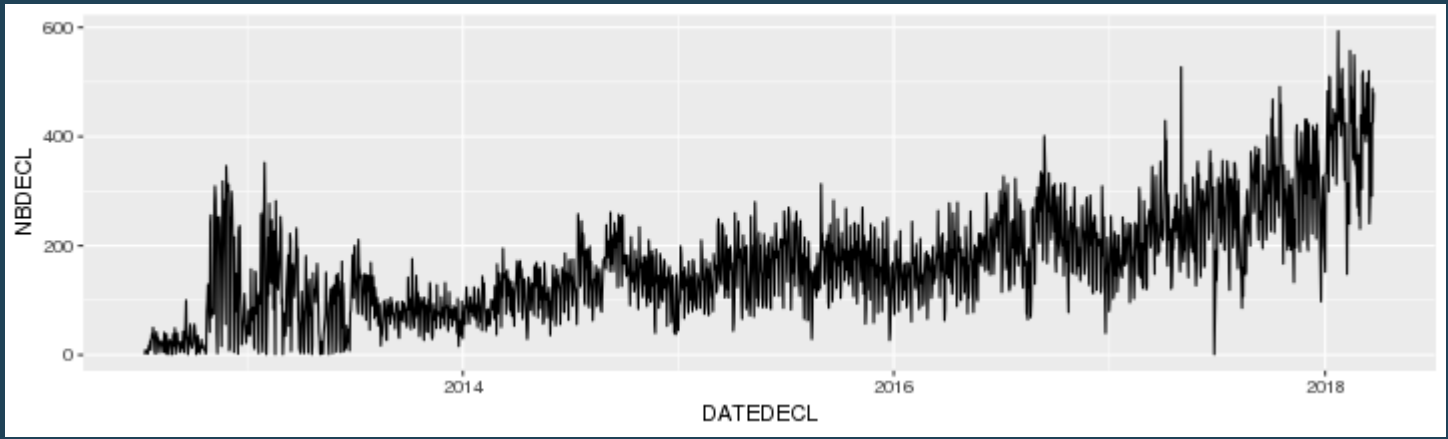
```
t %within% itv
```

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

Visualiser une série temporelle

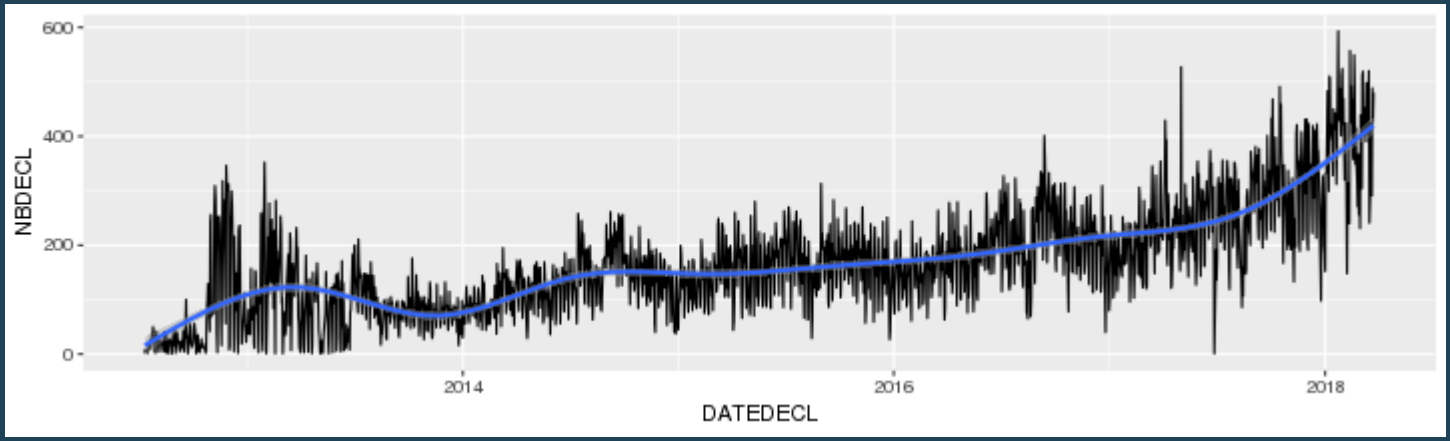
```
df_dmr <- readRDS("dans_ma_rue_clean.RDS")
df_dmr_parjour  <- df_dmr  %>%
  group_by(DATEDECL) %>%
  summarise(NBDECL=n())

ggplot(df_dmr_parjour, aes(x=DATEDECL, NBDECL))+
  geom_line()
```



Visualiser une tendance

```
ggplot(df_dmr_parjour, aes(x=DATEDECL, y=NBDECL))+
  geom_line()+
  geom_smooth(span=0.1)
```



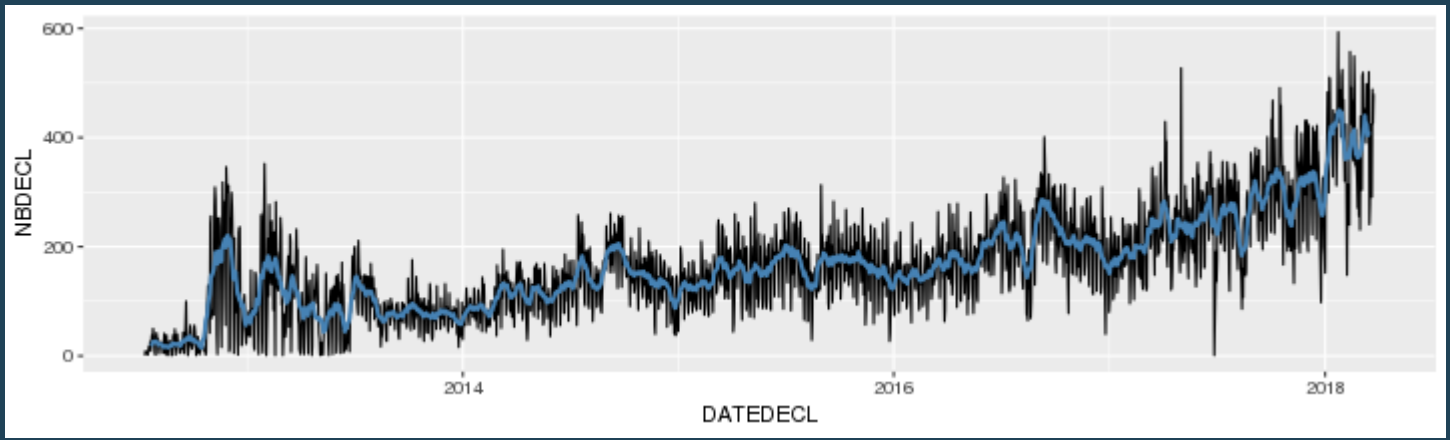
Ici il s'agit simplement d'une régression non-paramétrique => Il est difficile de définir exactement la "tendance centrale".

Visualiser une tendance

Pour visualiser une tendance il est également possible de représenter une moyenne mobile (moyenne ou autre métrique):

```
library(zoo)
df_dmr_parjour <- df_dmr_parjour %>%
          mutate(movavNBDECL = rollapply(NBDECL,15,mean,fill=NA))

ggplot(df_dmr_parjour, aes(x =DATEDECL, y=NBDECL))+
  geom_line()+
  geom_line(aes(y=movavNBDECL), col="steelblue", size=1)
```



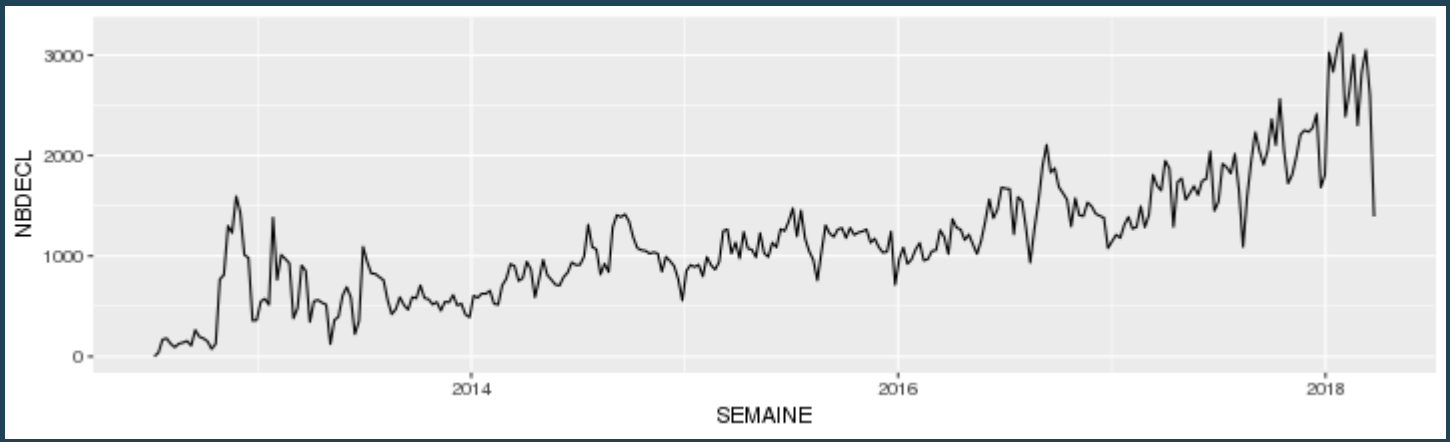
Agréger la donnée temporelle

Par exemple, ici, on agrège la donnée par semaine.

```
df_dmr_parsemaine <- df_dmr %>%
         mutate(SEMAINE = round_date(DATEDECL,"week")) %>%
         group_by(SEMAINE) %>%
         summarise(NBDECL = n())
```

La représentation des données s'en trouve de fait moins bruitée:

```
ggplot(df_dmr_parsemaine, aes(x=SEMAINE, y=NBDECL)) +
  geom_line()
```



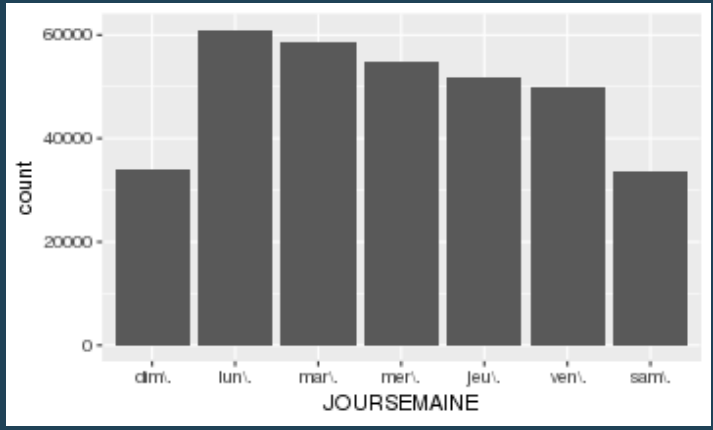
Visualiser une périodicité

Par exemple, pour considérer la périodicité du signal sur une semaine, on peut récupérer le jour de la semaine avec la fonction `wday()`.

```
df_dmr_joursemaine <- df_dmr %>%
  mutate(JOURSEMAINE =
           wday(DATEDECL,label = TRUE)
         )
```

On peut alors représenter le nombre total de déclarations d'incidents en fonction du jour de la semaine:

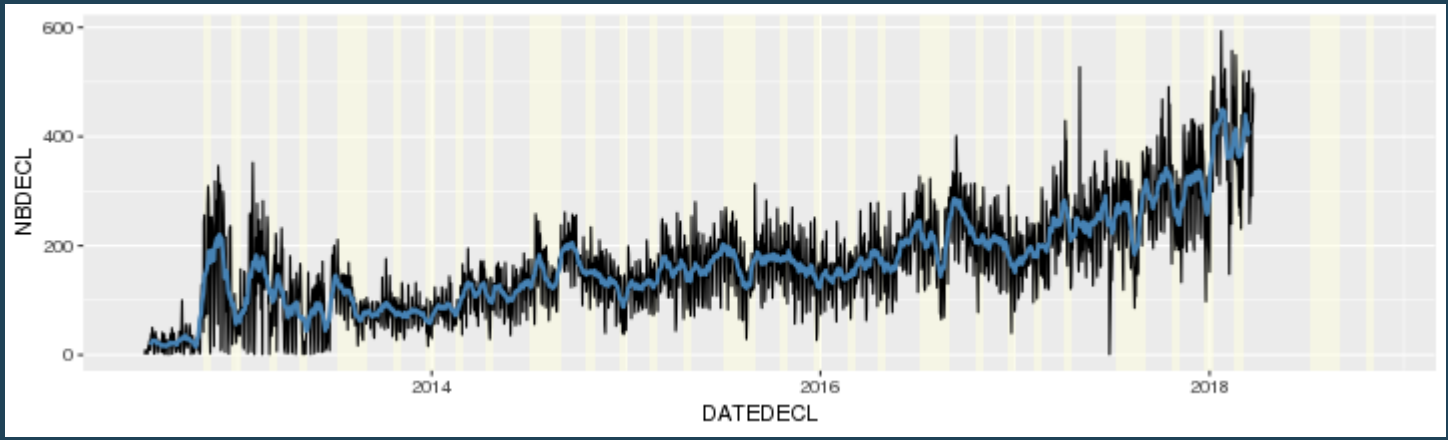
```
ggplot(df_dmr_joursemaine,
       aes(x = JOURSEMAINE)) +
  geom_bar()
```



Visualiser des intervalles particuliers

```
vacances <- readr::read_csv("data/vacances.csv") %>%
  mutate(Debut = as_datetime(dmy(Fin_des_cours)),
         Fin = as_datetime(dmy(Reprise_des_cours)))

ggplot(df_dmr_parjour,
            aes(x=DATEDECL, y=NBDECL))+
  geom_rect(data=vacances,inherit.aes=FALSE,
            aes(xmin=Debut, xmax=Fin,
                ymin=-Inf,ymax=+Inf),
            alpha=0.5, fill="lightyellow") +
  geom_line()+
  geom_line(aes(y=movavNBDECL), col="steelblue", size=1)
```



Voir aussi

Manipulation de séries temporelles avec R : `tibbletime` -->

https://github.com/business-science/tibbletime

Agrégation de séries temporelles avec R : `tsibble`

https://github.com/tidyverts/tsibble