Dates 2 : Manipuler les dates dans R

GRAPH Network & WHO, supported by the Global Fund to fight HIV, TB & Malaria

October 2023

This document is a draft of a lesson made by the GRAPH Network, a non-profit headquartered at the University of Geneva Global Health Institute, in collaboration with the World Health Organization, under a Global Fund 2023 grant to create e-learning modules to build in-country data capacity for epidemiological and impact analysis for National HIV, TB and malaria programs

Introduction

Vous avez maintenant une bonne compréhension de la façon dont les dates sont stockées, affichées et formatées dans R. Dans cette leçon, vous apprendrez à effectuer des analyses simples avec les dates, telles que le calcul de l'intervalle de temps entre les intervalles de dates et la création de graphiques de séries chronologiques ! Ces compétences sont cruciales pour toute personne travaillant avec des données de santé publique, car elles sont la base de la compréhension des tendances temporelles telles que la propagation des maladies, les changements dans les indicateurs de santé au niveau de la population et l'impact des mesures préventives !

Objectifs d'apprentissage

- Vous savez comment calculer les intervalles entre les dates
- Vous savez comment extraire des composants des colonnes de date
- Vous savez comment arrondir les dates
- Vous êtes capable de créer des graphiques de séries chronologiques simples

Packages

Veuillez charger les packages nécessaires pour cette leçon avec le code ci-dessous :

```
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse,
               here,
               lubridate,
               ggplot2)
```

Données

Les premières données avec lesquelles nous travaillerons sont les données IRS de la leçon précédente. Consultez la première leçon sur les dates pour plus d'informations sur le contenu de ces données de pulvérisation intradomiciliaire d'insecticide (IRS).

```
données_pulvérisation <- read_csv(here("data/Illovo_data.csv"))
```

```
données_pulvérisation
```

```
## # A tibble: 5 × 9
##   village     target_spray sprayed coverage_p
##   <chr>              <dbl>   <dbl>      <dbl>
## 1 Mess                  87      64       73.6
## 2 Nkombedzi            183     169       92.4
## 3 B Compound            16      16      100
## 4 D Compound             3       2       66.7
## 5 Post Office            6       3       50
##   start_date_default end_date_default start_date_typical
##   <date>             <date>           <chr>
## 1 2014-04-07         2014-04-17       07/04/2014
## 2 2014-04-22         2014-04-27       22/04/2014
## 3 2014-05-13         2014-05-13       13/05/2014
## 4 2014-05-13         2014-05-13       13/05/2014
## 5 2014-05-13         2014-05-13       13/05/2014
## # ℹ 2 more variables: start_date_long <chr>,
## #   start_date_messy <chr>
```

Le deuxième ensemble de données que nous utiliserons contient des données de la même étude à partir de laquelle les données IRS ont été extraites. Ici, nous avons le taux d'incidence mensuel moyen du paludisme pour 1000 personnes de 2015 à 2019 pour les villages ayant reçu l'IRS (cas) et les villages n'ayant pas reçu l'IRS (témoins). Le jeu de données contient également la température minimale moyenne et la température maximale moyenne à Illovo pour chaque mois de 2015 à 2019.

```
données_météo_paludisme <- read_csv(here("data/Illovo_ir_weather.csv"))
```

```
données_météo_paludisme
```

```
## # A tibble: 5 × 5
##   date       ir_case ir_control avg_min avg_max
##   <date>       <dbl>      <dbl>   <dbl>   <dbl>
## 1 2015-01-10    42.9       19.6    21.2    31.6
## 2 2015-02-03    61.0       10.1    21.5    32.9
## 3 2015-03-11    74.1       56.8    20.6    33.4
## 4 2015-04-15    95.2       34.7    18.5    32.3
## 5 2015-05-05    89.8       31.9    15.9    31.4
```

Les variables incluses dans ces données sont les suivantes :

- `date` : Points temporels mensuels allant de 2015 à 2019, avec le jour du mois généré de manière aléatoire
- `cas_IRS` : Taux d'incidence moyen du paludisme pour 1000 personnes pour les villages ayant reçu l'IRS
- `témoins` : Taux d'incidence moyen du paludisme pour 1000 personnes pour les villages n'ayant pas reçu l'IRS
- `moyenne_min` : Température minimale moyenne mensuelle en degrés Celsius
- `moyenne_max` : Température maximale moyenne mensuelle en degrés Celsius

MÉTÉO

```
météo <- read_csv(here("data/Illovo_weather.csv"))
```

```
météo
```

```
## # A tibble: 5 × 4
##   date       min_temp max_temp  rain
##   <date>        <dbl>    <dbl> <dbl>
## 1 2015-01-01     21.5     29.9  21.7
## 2 2015-01-02     19.6     30.4   2.2
## 3 2015-01-03     21.6     29.9  25.8
## 4 2015-01-04     20       29.5   1
## 5 2015-01-05     20       32.2  53
```

Calcul des intervalles de date

Pour commencer, nous allons examiner deux façons de calculer les intervalles, la première en utilisant l'opérateur "-" en R de base, et la seconde en utilisant l'opérateur d'intervalle du package `lubridate`. Jetons un coup d'œil à ces deux méthodes et comparons-les.

Utilisation de l'opérateur "-"

La première façon de calculer les différences de temps consiste à utiliser l'opérateur "-" pour soustraire une date d'une autre. Créons deux variables de date et essayons !

```
date_1 <- as.Date("2000-01-01") # 1er janvier 2000
date_2 <- as.Date("2000-01-31") # 31 janvier 2000
date_2 - date_1
```

```
## Time difference of 30 days
```

C'est aussi simple que ça ! Ici, nous pouvons voir que R renvoie la différence de temps en jours.

Utilisation de l'opérateur d'intervalle du package `lubridate`

La deuxième façon de calculer des intervalles de temps consiste à utiliser l'opérateur `%--%` du package `lubridate`. Nous pouvons voir ici que la sortie est légèrement différente de la sortie de R de base.

```
date_1 %--% date_2
```

```
## [1] 2000-01-01 UTC--2000-01-31 UTC
```

Notre sortie est un intervalle entre deux dates. Si nous voulons savoir combien de jours se sont écoulés, nous devons utiliser la fonction `days()`. Le `(1)` ici indique à lubridate de compter par incréments d'un jour à la fois.

```
date_1 %--% date_2/days(1)
```

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

Techniquement, spécifier `days(1)` n'est pas vraiment nécessaire, nous pouvons également laisser les parenthèses vides (c'est-à-dire `days()`) et obtenir le même résultat, car la valeur par défaut de lubridate est de compter par incréments de 1. Cependant, si nous voulons compter par incréments de 5 jours par exemple, nous pouvons spécifier `days(5)` et le résultat retourné sera de 6, car 5*6=30.

```
date_1 %--% date_2/days(5)
```

```
## [1] 6
```

Utilisez les trois méthodes différentes pour calculer le temps entre les dates ci-dessous. Utilisez la fonction `weeks()` à la place de `days()` dans la méthode lubridate pour obtenir la réponse en semaines.

```
oct_31 <- as.Date("2023-10-31")
jul_20 <- as.Date("2023-07-20")
```

Comparaison

Alors, quelle est la meilleure méthode à utiliser ? Eh bien, vous pouvez probablement voir que lubridate offre plus de flexibilité, et grâce à la manière dont il gère les dates (les détails étant hors du champ de cette formation), il est plus précis !

Jetons un coup d'œil à un exemple de cela en calculant un intervalle de 6 ans. Avec R de base, les résultats sont donnés en jours. Si nous voulons obtenir le résultat en années, nous devons utiliser la fonction `as.numeric()` pour transformer les résultats en un nombre sans l'unité "jours" et diviser par 365,25. Nous utilisons 365,25 parce qu'il y a un jour supplémentaire tous les 4 ans (c'est-à-dire les années bissextiles), donc en moyenne, il y a 365,25 jours par an. Avec lubridate, nous pouvons simplement spécifier `years()`. Voyons comment chaque méthode se comporte dans l'exemple suivant.

```
date_1 <- as.Date("2000-01-01") # 1er janvier 2000
date_2 <- as.Date("2006-01-01") # 1er janvier 2006
as.numeric(date_2-date_1)/365.25
```

```
## [1] 6.001369
```

```
date_1 %--% date_2/years()
```

```
## [1] 6
```

D'accord, ils sont très similaires, mais pas exactement les mêmes ! Évidemment, la réponse correcte est que 6 ans se sont écoulés entre les deux dates données. Cependant, les dates peuvent devenir compliquées, car le nombre de jours dans une année, voire dans un mois, n'est pas toujours le même. Essayer de calculer des intervalles de dates avec R de base est une approximation plutôt qu'une réponse exacte. Dans l'exemple ci-dessus, nous n'avons pas pu obtenir exactement 6 ans en utilisant l'opérateur moins en R de base, même si nous divisons par 365, 365,25 ou 366.

```
as.numeric(date_2-date_1)/365
```

```
## [1] 6.005479
```

```
as.numeric(date_2-date_1)/365.25
```

```
## [1] 6.001369
```

```
as.numeric(date_2-date_1)/366
```

```
## [1] 5.989071
```

Lubridate, en revanche, peut gérer ces complexités, il donnera donc une réponse exacte. Les différences sont légères, mais en termes de flexibilité et de précision, lubridate est le grand gagnant !

Bien que nous ne couvrions pas les date-heures dans ce cours, il est bon de savoir que `lubridate` est particulièrement utile pour traiter les fuseaux horaires et les changements d'heure d'été.

Pouvez-vous appliquer cela à notre jeu de données de pulvérisation ? Créez une nouvelle variable appelée `durée_pulvérisation` et utilisez l'opérateur `%--%` de lubridate pour calculer le nombre de jours entre `start_date_default` et `end_date_default`.

Extraction des composants de la date

Parfois, lors de votre nettoyage ou de votre analyse de données, vous devrez peut-être extraire un composant spécifique de votre variable de date. Un ensemble de fonctions utiles dans le package `lubridate` vous permet de le faire exactement. Par exemple, si nous voulions créer une colonne avec seulement le mois où la pulvérisation a commencé à chaque intervalle, nous pourrions utiliser la fonction `month()` de la manière suivante :

```
données_pulvérisation %>%
  mutate(mois_début = month(start_date_default)) %>%
  select(village, start_date_default, mois_début)
```

```
## # A tibble: 5 × 3
##   village     start_date_default mois_début
##   <chr>       <date>                  <dbl>
## 1 Mess        2014-04-07                  4
## 2 Nkombedzi   2014-04-22                  4
## 3 B Compound  2014-05-13                  5
## 4 D Compound  2014-05-13                  5
## 5 Post Office 2014-05-13                  5
```

Comme nous pouvons le voir ici, cette fonction renvoie le mois sous forme de numéro de 1 à 12. Pour notre première observation, la pulvérisation a commencé au cours du quatrième mois, donc en avril. C'est aussi simple que ça ! Si nous voulons que R affiche le

mois écrit plutôt que le numéro en dessous, nous pouvons utiliser l'argument `label=TRUE`.

```
données_pulvérisation %>%
  mutate(mois_début = month(start_date_default, label=TRUE)) %>%
  select(village, start_date_default, mois_début)
```

```
## # A tibble: 5 × 3
##   village     start_date_default mois_début
##   <chr>       <date>             <ord>
## 1 Mess        2014-04-07         Apr
## 2 Nkombedzi   2014-04-22         Apr
## 3 B Compound  2014-05-13         May
## 4 D Compound  2014-05-13         May
## 5 Post Office 2014-05-13         May
```

De même, si nous voulions extraire l'année, nous utiliserions la fonction `year()`.

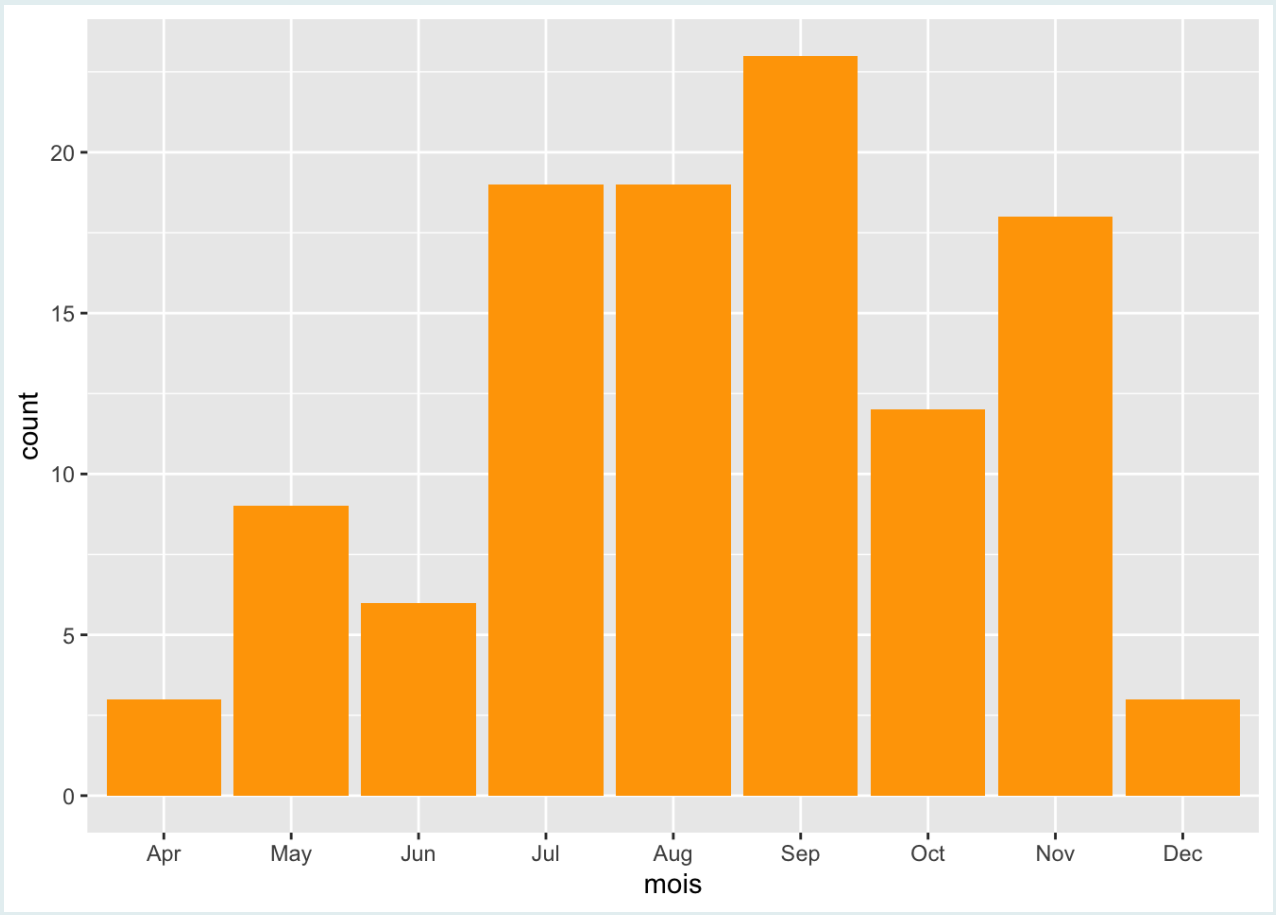
```
données_pulvérisation %>%
  mutate(année_début = year(start_date_default)) %>%
  select(village, start_date_default, année_début)
```

```
## # A tibble: 5 × 3
##   village     start_date_default année_début
##   <chr>       <date>                  <dbl>
## 1 Mess        2014-04-07               2014
## 2 Nkombedzi   2014-04-22               2014
## 3 B Compound  2014-05-13               2014
## 4 D Compound  2014-05-13               2014
## 5 Post Office 2014-05-13               2014
```

Créez une nouvelle variable appelée `jour_semaine_début` et extrayez le jour de la semaine où la pulvérisation a commencé de la même manière qu'indiqué ci-dessus, mais avec la fonction `wday()`. Essayez d'afficher les jours de la semaine écrits plutôt que numériquement.

Une raison pour laquelle vous pourriez vouloir extraire des composants de date spécifiques est lorsque vous souhaitez visualiser vos données, ce qui est très simple à faire dans R en utilisant le package `ggplot2` ! Par exemple, disons que nous voulions comparer les mois où la pulvérisation commence, nous pouvons le faire en créant une nouvelle variable de mois avec la fonction `month()` et en traçant un graphique à barres avec `geom_bar`.

```
données_pulvérisation %>%
  mutate(mois = month(start_date_default, label = TRUE)) %>%
  ggplot(aes(x = mois)) +
  geom_bar(fill = "orange")
```



Ici, nous pouvons voir que la plupart des campagnes de pulvérisation ont commencé entre juillet et novembre, sans aucune en janvier, février et mars. Les auteurs de l'article à partir duquel ces données ont été extraites ont déclaré que les campagnes de pulvérisation visaient à se terminer juste au début de la saison des pluies (novembre-avril) au Malawi. Cela était à la fois pour des raisons pratiques et en prévision d'une transmission plus élevée du paludisme. Nous pouvons voir ce schéma temporel de pulvérisation reflété dans notre graphique !

Créez un nouveau graphique représentant les mois où la campagne de pulvérisation s'est terminée et comparez-le au graphique de son commencement. Cela correspond-il à vos attentes compte tenu de la variable `durée_pulvérisation` que vous avez créée dans l'exercice de la section précédente ?

Arrondir

Parfois, il est nécessaire d'arrondir nos dates vers le haut ou vers le bas si nous voulons analyser ou visualiser nos données de manière significative. Tout d'abord, voyons ce que nous entendons par "arrondir" avec quelques exemples simples.

Prenons la date du 17 mars 2012. Si nous voulions arrondir vers le bas au mois le plus proche, alors nous utiliserions la fonction `floor_date()` de `lubridate` avec l'argument `unit="month"`.

```
ma_date_inf <- as.Date("2012-03-17")
floor_date(ma_date_inf, unit="month")
```

```
## [1] "2012-03-01"
```

Comme nous pouvons le voir, notre date est maintenant le 1er mars 2012.

Si nous voulions arrondir vers le haut, nous pouvons utiliser la fonction `ceiling_date()`. Essayons ceci avec la date du 3 janvier 2020.

```
ma_date_sup <- as.Date("2020-01-03")
ceiling_date(ma_date_sup, unit="month")
```

```
## [1] "2020-02-01"
```

Avec `ceiling_date()`, le 3 janvier a été arrondi au 1er février.

Enfin, nous pouvons également simplement arrondir sans spécifier vers le haut ou vers le bas, et les dates sont automatiquement arrondies à l'unité spécifiée la plus proche.

```
mes_dates <- as.Date(c("2000-11-03", "2000-11-27"))
round_date(mes_dates, unit="month")
```

```
## [1] "2000-11-01" "2000-12-01"
```

Ici, nous pouvons voir qu'en arrondissant au mois le plus proche, le 3 novembre est arrondi au 1er novembre, et le 27 novembre est arrondi au 1er décembre.

Nous pouvons également arrondir vers le haut ou vers le bas à l'année la plus proche. Que pensez-vous que sera la sortie si nous arrondissons vers le bas la date du 29 novembre 2001 à l'année la plus proche :

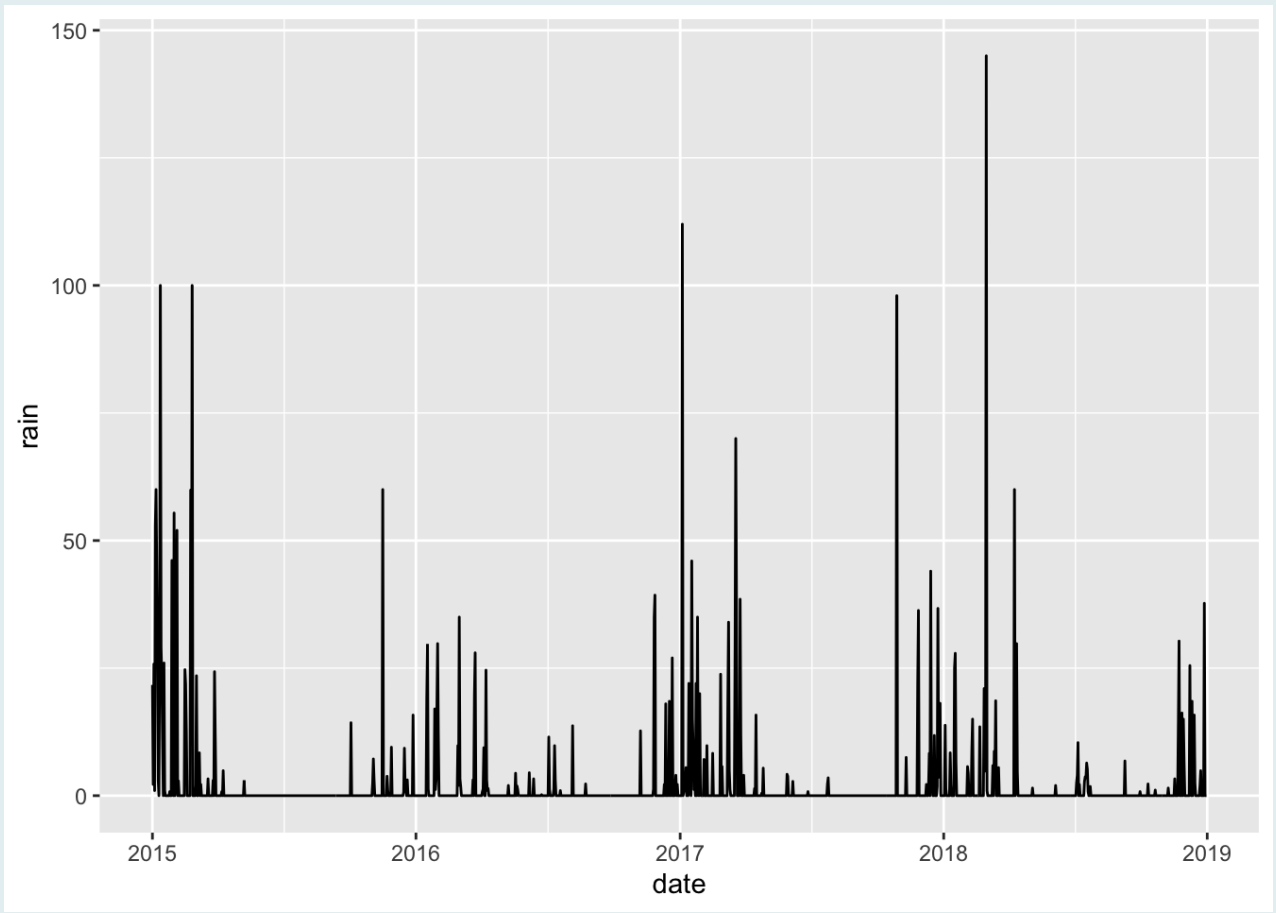
```
date_arrondie <- as.Date("2001-11-29")
floor_date(date_arrondie, unit="year")
```

J'espère que ce que nous entendons par "arrondir" est un peu plus clair ! Alors, pourquoi cela pourrait-il être utile avec nos données ? Eh bien, passons maintenant à nos données météorologiques.

```
météo
```

Comme nous pouvons le voir, nos données météorologiques sont enregistrées quotidiennement, mais ce niveau de détail n'est pas idéal pour étudier comment les schémas météorologiques affectent la transmission du paludisme, qui suit un schéma saisonnier. Les données météorologiques quotidiennes peuvent être assez bruyantes compte tenu de la variation significative d'un jour à l'autre. Par exemple, un graphique des précipitations quotidiennes ne serait pas très informatif. Essayons pour voir :

```
météo %>%
  ggplot()+
  geom_line(aes(date, rain))
```



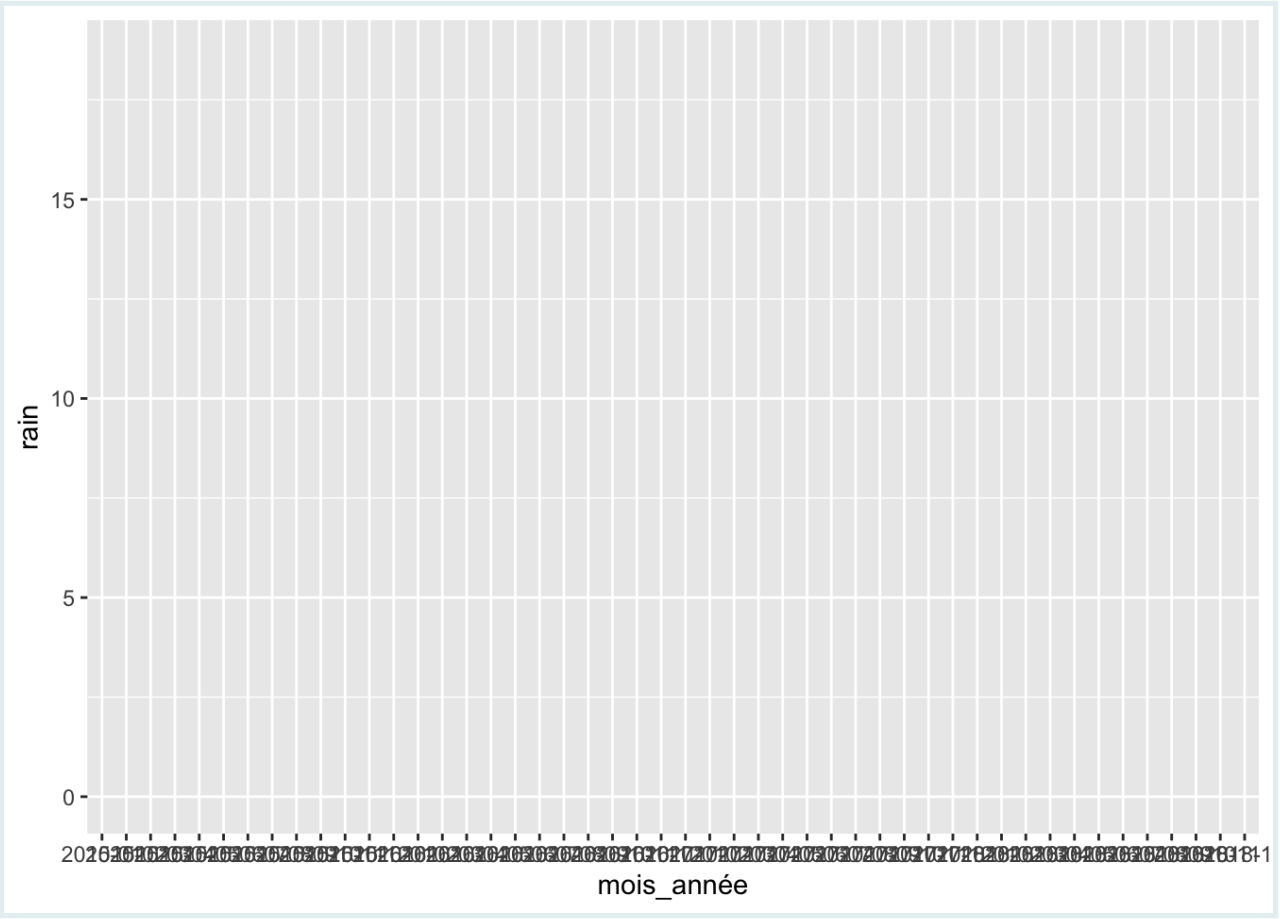
Outre le fait qu'il soit visuellement confus, ce graphique ne permet pas d'illustrer efficacement les tendances saisonnières. L'agrégation mensuelle est une approche plus efficace pour capturer les variations saisonnières et réduire le bruit. Si nous voulions représenter les précipitations mensuelles moyennes, notre première tentative pourrait être d'utiliser la fonction `str_sub()` pour extraire les sept premiers caractères de notre date (le mois et l'année).

```
météo_mauvaise <- météo %>%
  mutate(mois_année=str_sub(date, 1, 7)) %>%
  group_by(mois_année) %>%
  summarise(rain=mean(rain))
météo_mauvaise
```

Cependant, si nous essayons de le représenter, notre nouvelle variable `mois_année` n'est plus une variable de date, nous ne pouvons donc pas représenter des graphiques dans le temps car elle n'est pas continue !

```
météo_mauvaise %>%
  ggplot() +
  geom_line(aes(mois_année, rain))
```

```
## `geom_line()`: Each group consists of only one
## observation.
## ℹ Do you need to adjust the group aesthetic?
```

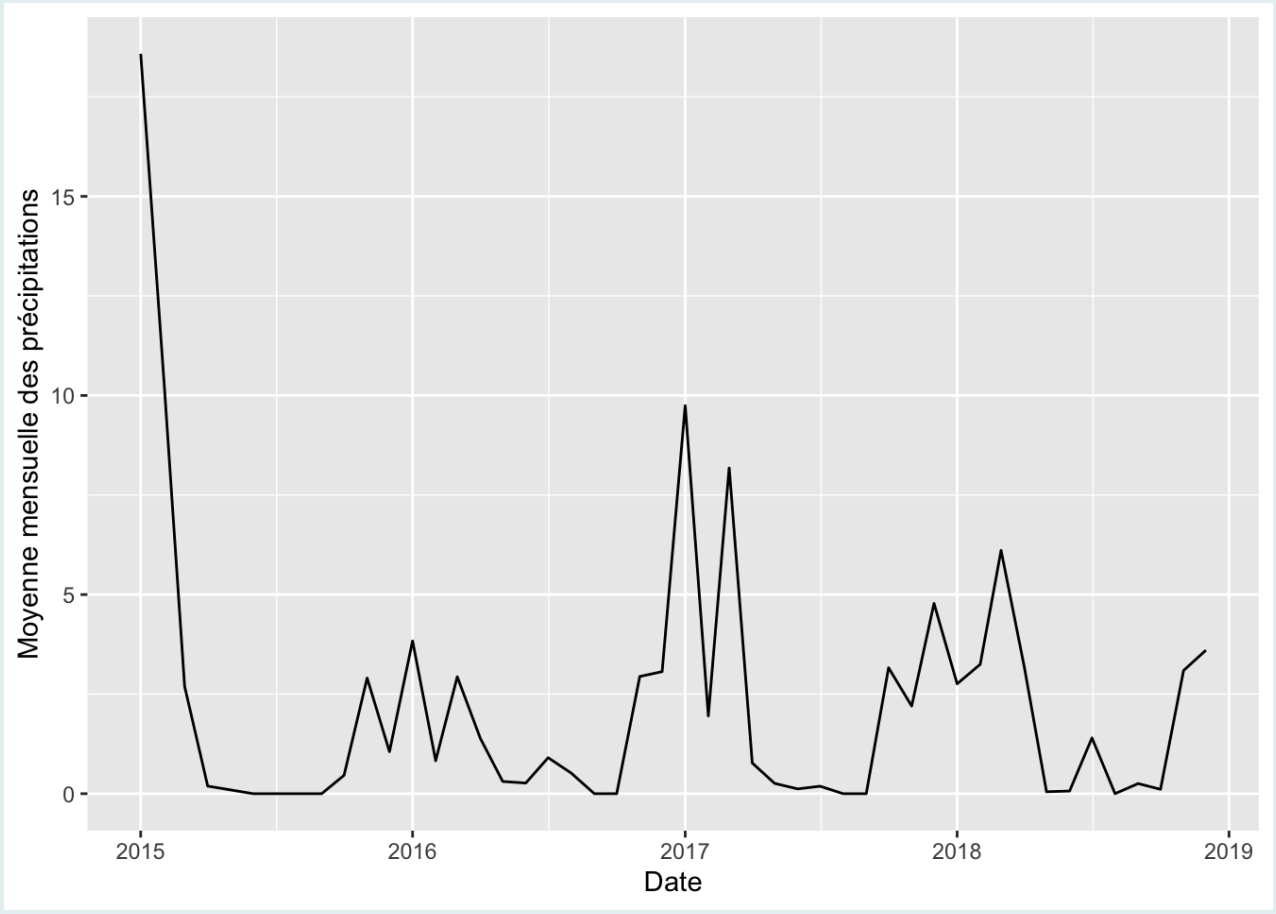


La meilleure façon de faire cela est d'abord d'arrondir nos dates au mois à l'aide de la fonction `floor_date()`, puis de regrouper nos données par notre nouvelle variable `mois_année`, et enfin de calculer la moyenne mensuelle. Essayons maintenant.

```
météo <- météo %>%
  mutate(mois_année=floor_date(date, unit="month")) %>%
  group_by(mois_année) %>%
  summarise(rain=mean(rain))
météo
```

Maintenant nous pouvons représenter nos données et nous aurons un graphique de la pluviométrie mensuelle moyenne sur la période de pulvérisation de 4 ans.

```
météo %>%
  ggplot() +
  geom_line(aes(mois_année, rain)) +
  labs(x="Date", y="Moyenne mensuelle des précipitations")
```



Cela semble beaucoup mieux ! Maintenant, nous obtenons une image beaucoup plus claire des tendances saisonnières et des variations annuelles.

À l'aide des données météorologiques, créez un nouveau graphique représentant les températures minimales et maximales moyennes mensuelles de 2015 à 2019.

WRAP UP!

[XXX NICE WRAP UP MESSAGE OR SUMMARY IF NEEDED HERE XXX]

Answer Key

```
données_pulvérisation %>%
  mutate(jour_semaine_début = wday(start_date_default, label=TRUE)) %>%
  select(village, start_date_default, jour_semaine_début)
```

```
## # A tibble: 5 × 3
##   village     start_date_default jour_semaine_début
##   <chr>       <date>             <ord>
## 1 Mess        2014-04-07         Mon
## 2 Nkombedzi   2014-04-22         Tue
## 3 B Compound  2014-05-13         Tue
## 4 D Compound  2014-05-13         Tue
## 5 Post Office 2014-05-13         Tue
```

Contributors

The following team members contributed to this lesson:

(make sure to update the contributor list accordingly!)