Dates 2 : Intervalles, Composantes et Arrondissement

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!

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)
```

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).

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

```
## # 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
## # ℹ 5 more variables: start_date_default <date>,
## #   end_date_default <date>, start_date_typical <chr>, …
```

Le deuxième jeu de données contient des données mensuelles de 2015 à 2019 comparant l'incidence moyenne du paludisme pour 1000 personnes dans les villages ayant reçu des pulvérisations intradomiciliaires d'insecticide rémanent (PIIR) par rapport aux villages qui n'en ont pas reçu.

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

```
## # 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
```

La colonne `ir_case` montre l'incidence du paludisme dans les villages avec PIIR et `ir_control` montre l'incidence dans les villages sans PIIR. La colonne `date` contient le mois et un jour aléatoire. Le jeu de données inclut également les températures minimales et maximales moyennes mensuelles (`avg_min` et `avg_max`).

Le jeu de données final contient 1460 lignes de données meteorologiques quotidiennes pour la région d'Illovo de 2015 à 2019.

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

```
## # 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
```

Chaque ligne représente un seul jour et inclut des mesures de température minimale (`min_temp`) en degrés Celsius, de température maximale (`max_temp`) en degrés Celsius, et de précipitations (`rain`) en millimètres.

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`. Cela s'appelle parfois l'opérateur d'intervalle. 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
```

Cela signifie qu'il y avait six périodes de 5 jours dans cet intervalle. ::: r-practice

Lubridate weeks

Utilisez la fonction `weeks()` à la place de `days()` dans la méthode lubridate pour calculer la différence de temps en semaines entre les deux dates ci-dessous :

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

:::

Comparaison

Alors, quelle méthode est la meilleure ? Lubridate offre plus de flexibilité et de précision lorsqu'on travaille avec des dates dans R. Regardons un exemple simple pour comprendre pourquoi.

Commençons par définir deux dates espacées de 6 ans :

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

Si nous voulons calculer combien d'années se sont écoulées entre ces dates, comment procéderions-nous avec R de base ? Nous pourrions d'abord soustraire les deux dates, `date_2 - date_1`, puis diviser par un nombre moyen de jours, comme 365,25 (en tenant compte des années bissextiles) :

```
(date_2 - date_1)/365.25
```

```
## Time difference of 6.001369 days
```

Le résultat est proche de 6 ans mais pas précis!

Le résultat est toujours donné en "jours". Vous pouvez facilement convertir la différence de temps en une valeur numérique :

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

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

Diviser par 365 ou 366 donnera également des résultats imprécis :

```
(date_2 - date_1)/365
```

```
## Time difference of 6.005479 days
```

```
(date_2 - date_1)/366
```

```
## Time difference of 5.989071 days
```

Ce que vous devez faire est de prendre en compte qu'il y a seulement deux années bissextiles (deux jours supplémentaires) entre ces dates et de soustraire d'abord ces deux jours :

```
(date_2 - date_1 - 2)/365
```

```
## Time difference of 6 days
```

Mais ce sera une chose pénible à faire en pratique lorsqu'on travaille avec des données réelles. Avec les intervalles de lubridate, le processus est plus simple, car les années

bissextiles sont prises en compte pour vous :

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

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

La différence est légère, mais lubridate est clairement le gagnant dans cette situation.

Bien que nous ne les couvrions pas dans ce cours, {lubridate} est également excellent pour gérer les irrégularités temporelles telles que les fuseaux horaires et les changements d'heure d'été.

Lubridate intervals

Pouvez-vous appliquer la fonction d'intervalle de lubridate à notre ensemble de données IRS ? Créez une nouvelle colonne appelée "spraying_time" et, en utilisant l'opérateur %-% de lubridate, calculez le nombre de jours entre `start_date_default` et `end_date_default`.

::: side- note Lubridate distingue techniquement entre les périodes et les durées. Vous pouvez en savoir plus ici :::

Note: I've kept the code unchanged since code is language-independent.

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 :

```
        irs %>%
          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`.

```
irs %>%
  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()`.

```
irs %>%
  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
```

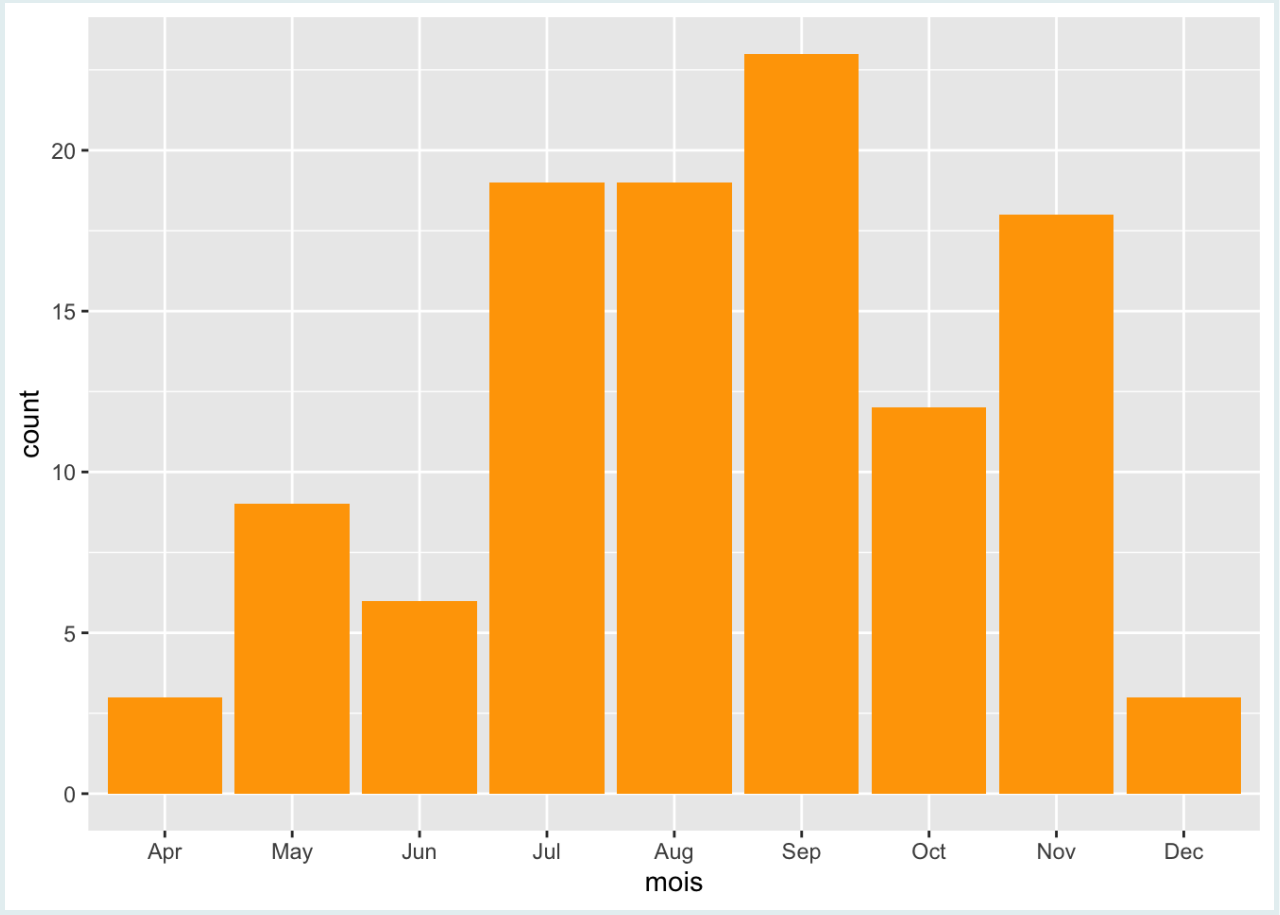
Extraction des jours de la semaine

Créez une nouvelle variable appelée `jour_semaine_debut` 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 des raisons pour lesquelles vous pourriez vouloir extraire des composants de date spécifiques est lorsque vous voulez visualiser vos données.

Par exemple, disons que nous voulions visualiser les mois où les pulvérisations commencent, nous pouvons le faire en créant une nouvelle variable mois avec la fonction `month()`, et en traçant un graphique à barres avec `geom_bar`.

```r
irs %>%
  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, aucune n'ayant lieu durant les trois premiers mois de l'année. Les auteurs de l'article dont ces données sont issues ont déclaré que les campagnes de pulvérisation visaient à se terminer juste avant la saison des pluies (novembre-avril) au Malawi. Cela correspond au schéma observé.

Visualisation des mois de fin d'arrosage

Utilisez le data frame `irs` pour créer un nouveau graphique montrant les mois de fin des campagnes de pulvérisation. Comparez ce graphique à celui du début des campagnes. Est-ce que les motifs sont similaires ?

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 <- as.Date("2012-03-17")
floor_date(ma_date, 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 <- as.Date("2020-01-03")
ceiling_date(ma_date, 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.

Pratique de l'arrondi des dates

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 :

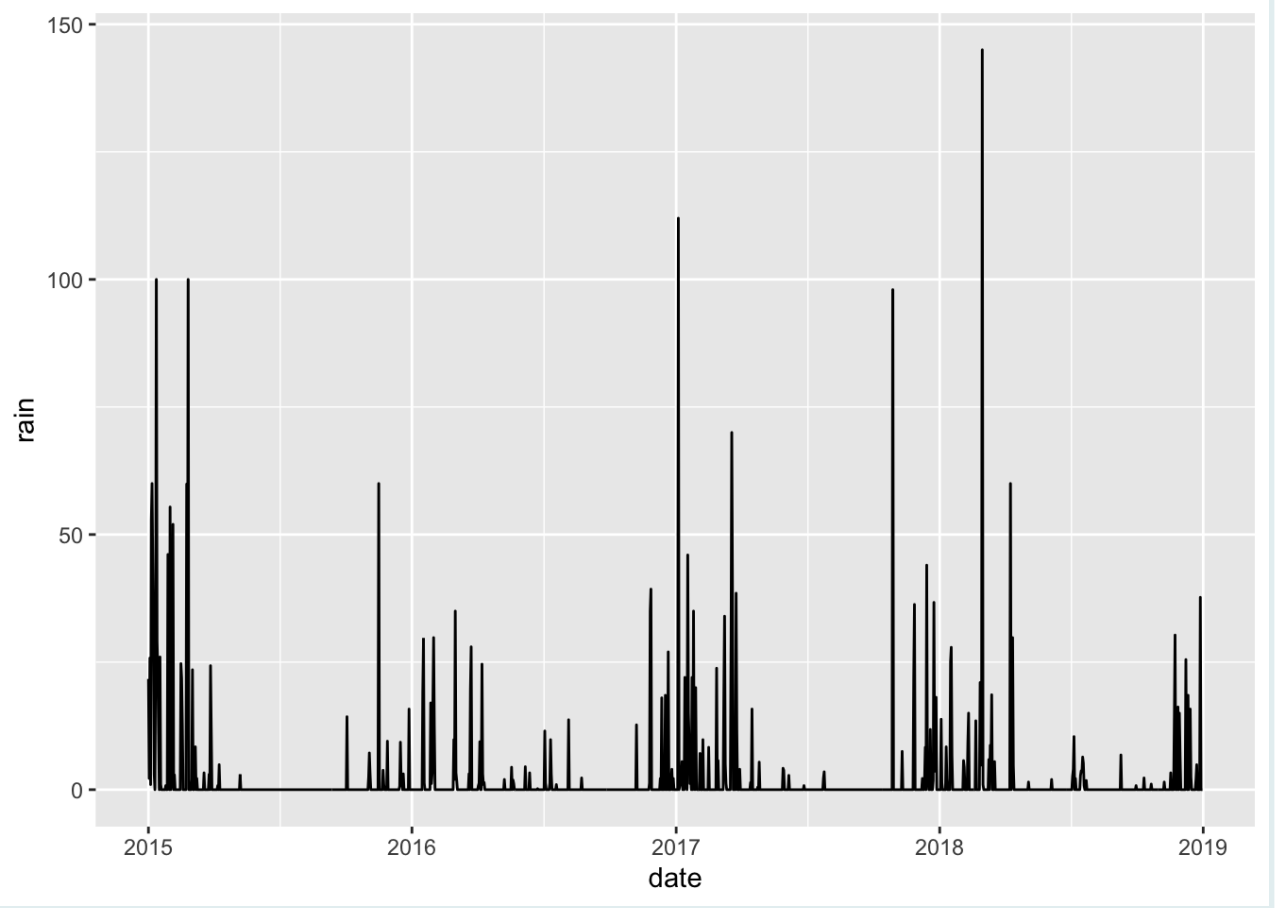
```
date_arrondie <- as.Date("2001-11-29")
floor_date(date_arrondie, unit="year")
```

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 meteorologiques.

```
meteo
```

Comme nous pouvons le voir, nos données météorologiques sont enregistrées quotidiennement, mais ce niveau de détail n'est pas idéal pour étudier comment les tendances météorologiques affectent la transmission du paludisme, qui suit un modèle saisonnier. Les données météorologiques quotidiennes peuvent être assez bruitées étant donné la variation significative d'un jour à l'autre :

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



En plus d'être visuellement en désordre, il est un peu difficile de voir les motifs saisonniers. L'agrégation mensuelle est une approche plus efficace pour capturer les variations saisonnières.

Si nous voulions tracer la moyenne des précipitations mensuelles, notre première tentative pourrait être d'utiliser la fonction `str_sub()` pour extraire les sept premiers caractères de notre date (le composant mois et année).

```
        meteo %>%
          mutate(mois_annee=str_sub(date, 1, 7))
```

Ensuite, nous regroupons par `mois_annee` pour calculer la moyenne des précipitations pour chaque mois :

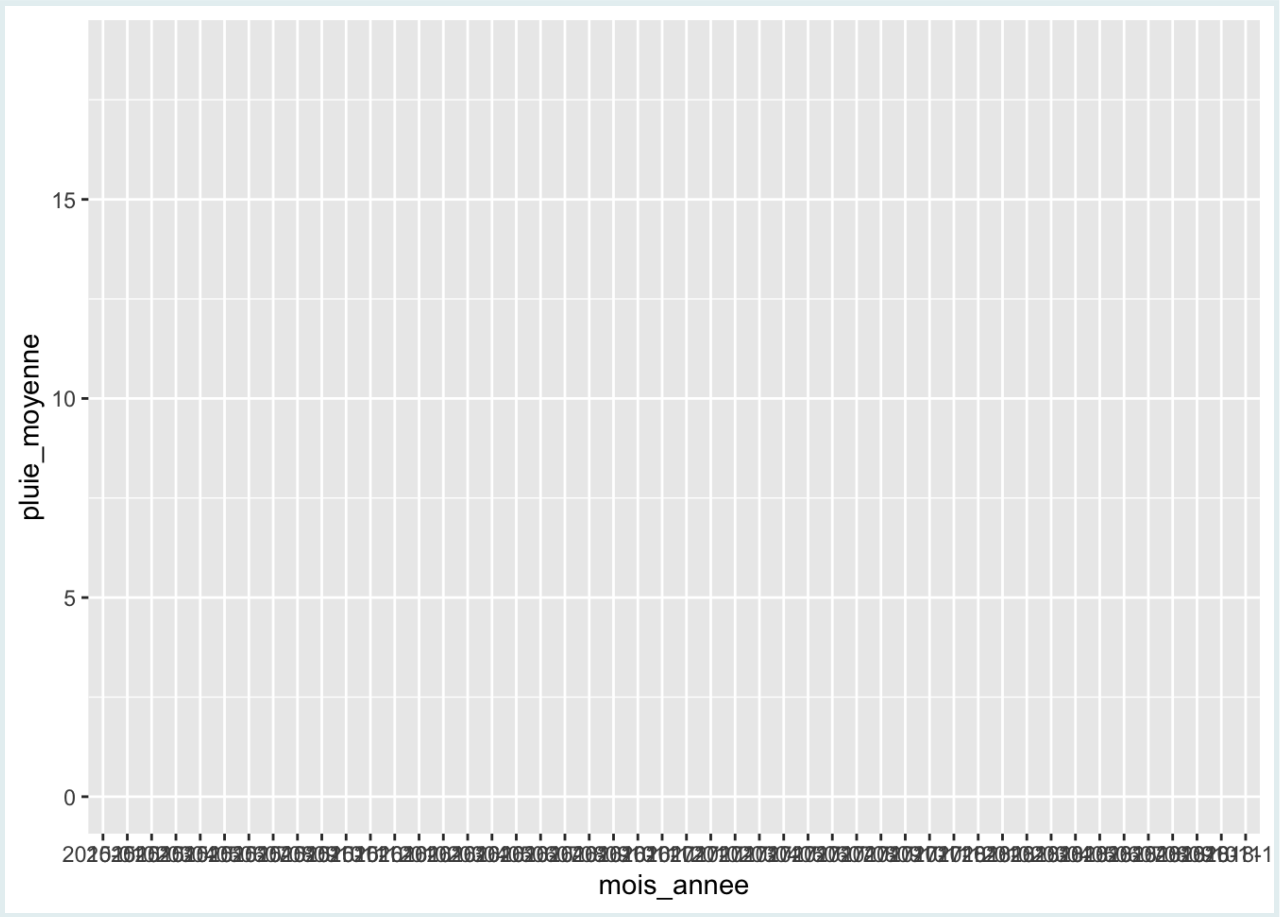
```
        resume_meteo_1 <-
          weather %>%
          mutate(mois_annee=str_sub(date, 1, 7)) %>%
          # regrouper et résumer
          group_by(mois_annee) %>%
          summarise(pluie_moyenne=mean(rain))
        resume_meteo_1
```

```
## # A tibble: 48 × 2
##    mois_annee pluie_moyenne
##    <chr>              <dbl>
##  1 2015-01          18.6
##  2 2015-02          10.4
##  3 2015-03           2.69
##  4 2015-04           0.19
##  5 2015-05           0.0968
##  6 2015-06           0
##  7 2015-07           0
##  8 2015-08           0
##  9 2015-09           0
## 10 2015-10           0.461
## # ℹ 38 more rows
```

Cependant, nous rencontrons un problème lors de la tentative de tracer ces données. Notre variable `mois_annee` est maintenant un caractère, et non une date. Cela signifie qu'elle n'est pas continue. Tracer un graphique linéaire avec une variable non continue ne fonctionne pas :

```
        resume_meteo_1 %>%
          ggplot() +
          geom_line(aes(mois_annee, pluie_moyenne))
```

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



La meilleure façon de procéder est d'abord d'arrondir nos dates au mois en utilisant la fonction `floor_date()`, puis de regrouper nos données par notre nouvelle variable `mois_annee`, et ensuite de calculer la moyenne mensuelle. Essayons-le maintenant.

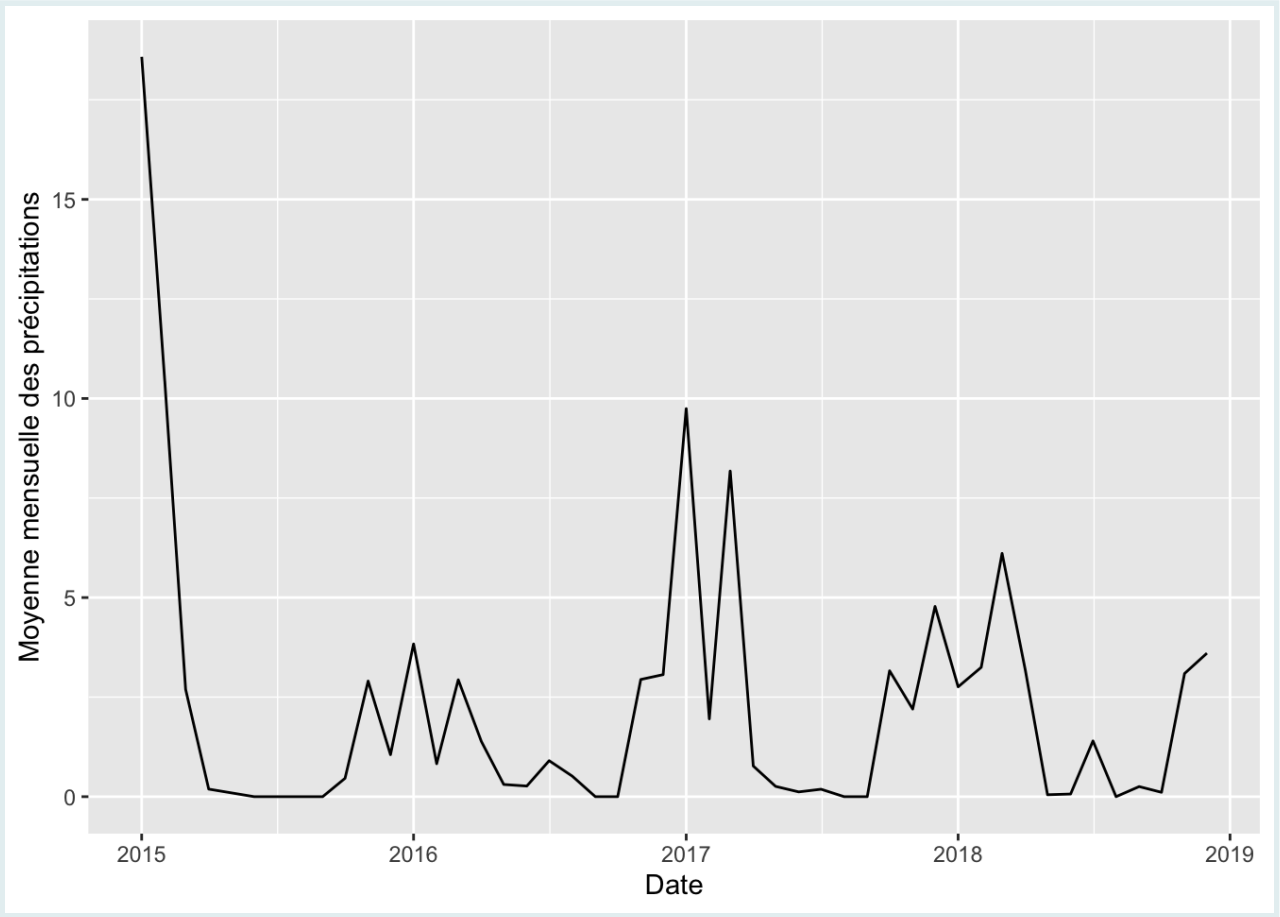
```
resume_meteo_2 <- weather %>%
  mutate(mois_annee=floor_date(date, unit="month")) %>%
  group_by(mois_annee) %>%
  summarise(pluie_moyenne=mean(rain))
resume_meteo_2
```

```
## # A tibble: 48 × 2
##    mois_annee pluie_moyenne
##    <date>             <dbl>
##  1 2015-01-01       18.6
##  2 2015-02-01       10.4
##  3 2015-03-01        2.69
##  4 2015-04-01        0.19
##  5 2015-05-01        0.0968
##  6 2015-06-01        0
##  7 2015-07-01        0
##  8 2015-08-01        0
##  9 2015-09-01        0
```

```
## 10 2015-10-01        0.461
## # ℹ 38 more rows
```

Maintenant, nous pouvons tracer nos données et nous aurons un graphique de la moyenne des précipitations mensuelles sur la période de 4 ans.

```
resume_meteo_2 %>%
  ggplot() +
  geom_line(aes(mois_annee, pluie_moyenne)) +
  labs(x="Date", y="Moyenne mensuelle des précipitations")
```



Cela semble bien meilleur ! Nous obtenons maintenant une image beaucoup plus claire des tendances saisonnières et des variations annuelles.

Tracer les températures minimales et maximales mensuelles moyennes

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

Conclusion

Cette leçon a couvert les compétences fondamentales pour travailler avec des dates en R - calculer des intervalles, extraire des composantes, arrondir et créer des visualisations de séries temporelles. Avec ces blocs de construction clés maintenant maîtrisés, vous pouvez désormais commencer à manipuler des données de date pour découvrir et analyser des modèles au fil du temps.

Corrigé

Lubridate weeks

```
        oct_31 <- as.Date("2023-10-31")
        jul_20 <- as.Date("2023-07-20")
        difference_temps <- oct_31 %--% jul_20
        difference_temps/weeks(1)
```

```
## [1] -14.71429
```

Intervalles avec Lubridate

```
        irs %>%
          mutate(temps_arrosage = interval(start_date_default,
end_date_default)/days(1)) %>%
          select(temps_arrosage)
```

```
## # A tibble: 112 × 1
##    temps_arrosage
##             <dbl>
##  1             10
##  2              5
##  3              0
##  4              0
##  5              0
##  6             11
##  7              0
##  8              0
##  9             19
## 10              9
## # ℹ 102 more rows
```

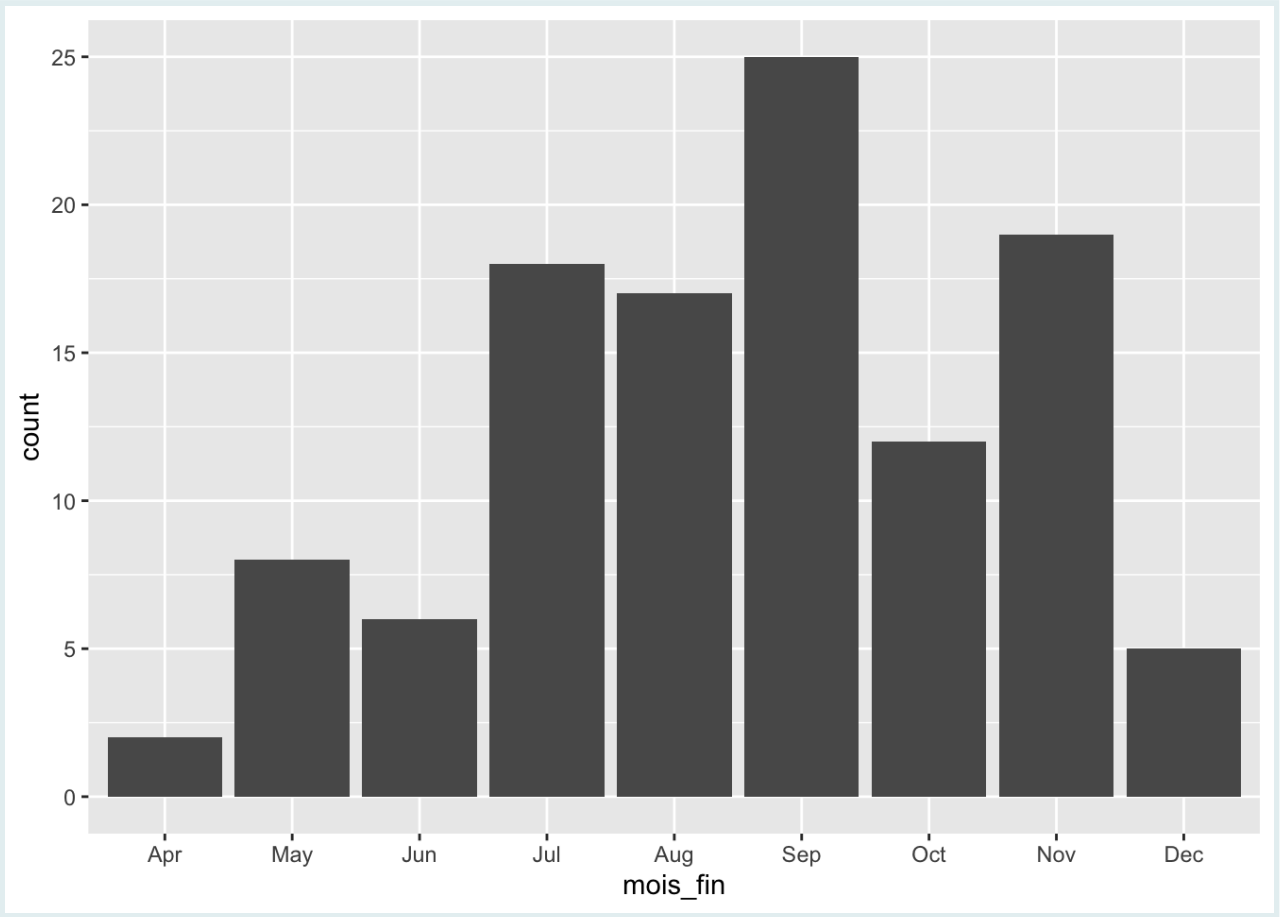
Extraction des jours de la semaine

```r
        irs %>%
          mutate(jour_semaine_debut = wday(start_date_default, label = TRUE))
%>%
          select(jour_semaine_debut)
```

```
## # A tibble: 112 × 1
##    jour_semaine_debut
##    <ord>
##  1 Mon
##  2 Tue
##  3 Tue
##  4 Tue
##  5 Tue
##  6 Thu
##  7 Tue
##  8 Tue
##  9 Wed
## 10 Wed
## # ℹ 102 more rows
```

Visualisation des mois de fin d'arrosage

```r
        irs %>%
          mutate(mois_fin = month(end_date_default, label = TRUE)) %>%
          ggplot(aes(x = mois_fin)) +
          geom_bar()
```



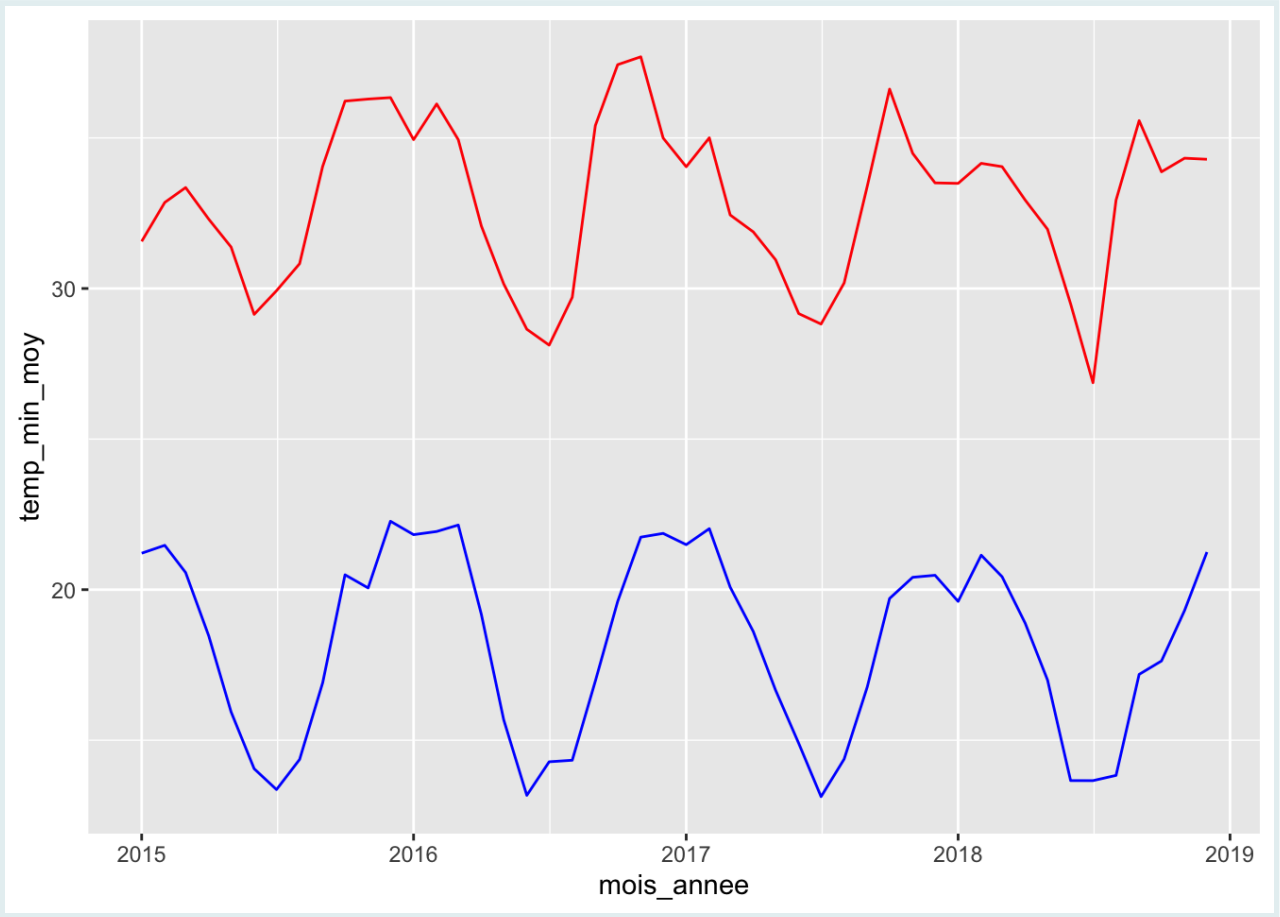
Pratique de l'arrondi des dates

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

```
## [1] "2001-01-01"
```

Tracer les températures minimales et maximales mensuelles moyennes

```
weather %>%
  mutate(mois_annee = floor_date(date, unit="month")) %>%
  group_by(mois_annee) %>%
  summarise(temp_min_moy = mean(min_temp),
            temp_max_moy = mean(max_temp)) %>%
  ggplot() +
  geom_line(aes(x = mois_annee, y = temp_min_moy), color = "blue") +
  geom_line(aes(x = mois_annee, y = temp_max_moy), color = "red")
```



Contributors

The following team members contributed to this lesson:

AMANDA MCKINLEY

R Developer and Instructor, the GRAPH Network

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement

GUY WAFEU

R Instructor and Public Health Physician
Committed to improving the quality of data analysis