Dates 2: Intervals, Components and Rounding

Intro

You now have a solid understanding of how dates are stored, displayed, and formatted in R. In this lesson, you will learn how to perform simple analyses with dates, such as calculating the time between date intervals and creating time series graphs! These skills are crucial for anyone working with health data, as they are the basis to understanding temporal patterns such as the progression of diseases over time and the fluctuation in population health metrics across different periods.

Learning Objectives

- You know how to calculate intervals between dates
- You know how to extract components from date columns
- You know how to round dates
- You are able to create simple time series graphs

Packages

Please load the packages needed for this lesson with the code below:

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

Dataset

The two datasets we will be working with contain information related to indoor residual spraying (IRS) for malaria control efforts in Illovo, Malawi.

The first dataset details the start and end dates of mosquito spraying campaigns in different villages. More information about the dataset can be found in the previous lesson on dates.

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

The second dataset contains monthly data from 2015-2019 comparing the average incidence of malaria per 1000 people in villages that received indoor residual spraying (IRS) versus villages that did not.

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

The `ir_case` column shows malaria incidence in IRS villages and `ir_control` shows incidence in non-IRS villages. The `date` column contains the month and random day. The dataset also includes average monthly minimum and maximum temperatures (`avg_min` and `avg_max`).

The final dataset contains 1,460 rows of daily weather data for the Illovo region from 2015-2019.

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

Each row represents a single day and includes measurements of minimum temperature (`min_temp`) in degrees Celsius, maximum temperature (`max_temp`) in degrees Celsius, and rainfall (`rain`) in millimeters.

Calculating Date Intervals

To start off, we're going to look at two ways to calculate intervals, the first using the "-" operator in base R, and the second using the interval operator from the {lubridate} package. Let's take a look at both of these and compare.

Using the "-" operator

The first way to calculate time differences is using the "-" operator to subtract one date from another. Let's create two date variables and try it out!

```
date_1 <- as.Date("2000-01-01") # January 1st, 2000
date_2 <- as.Date("2000-01-31") # January 31st, 2000
date_2 - date_1
```

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

It's that simple! Here we can see that R outputs the time difference in days.

Using the interval operator from {lubridate}

The second way to calculate time intervals is by using the %--% operator from the {lubridate} package. This is sometimes called the interval operator. We can see here that the output is slightly different to the base R output.

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

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

Our output is an interval between two dates. If we want to know how long has passed in days, we have to use the `days()` function. The `(1)` here tells lubridate to count in increments of one day at a time.

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

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

Technically, specifying `days(1)` isn't actually necessary, we can also leave the parentheses empty (ie. `days()`)and get the same result because lubridate's default is to count in increments of 1. However, if we want to count in increments of 5 days for example, we can specify `days(5)` and the result returned to us will be 6, because 5*6=30.

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

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

This means there were six 5-day-periods in that interval.

Lubridate weeks

Use the `weeks()` function in place of `days()` in the lubridate method to calculate the time difference in weeks between the two dates below:

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

Comparison

So which of the methods is best? Lubridate provides more flexibility and accuracy when working with dates in R. Let's look at a simple example to see why.

First let's set two dates that are 6 years apart:

```
date_1 <- as.Date("2000-01-01") # January 1st, 2000
date_2 <- as.Date("2006-01-01") # January 1st, 2006
```

If we want to calculate how many years have passed between these dates, how would we proceed in base R? We might first subtract the two dates, `date_2 - date_1`, then divide by an average day count, like 365.25 (accounting for leap years):

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

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

The result is close to 6 years but not precise due to the averaging of leap years!

The result is still given in "days". You can convert the time difference to a numeric value easily:

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

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

Dividing by 365 or 366 will also give imprecise results:

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

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

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

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

What you need to do is take into account that there are just two leap years (two extra days) between those dates and subtract those two days out first:

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

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

But this will be a painful thing to do in practice when working with real data. With lubridate intervals, the process is more straightforward, as leap years are handled for you:

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

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

The difference is slight, but lubridate is clearly a winner in this situation.

Although we don't cover them in this course, {lubridate} is also great for dealing with time irregularities like time zones and daylight savings shifts.

Lubridate intervals

Can you apply lubridate's interval function to our IRS dataset? Create a new column called `spraying_time` and using lubridates `%--%` operator, calculate the number of days between `start_date_default` and `end_date_default`.

SIDE NOTE

Lubridate has a technical distinction between "intervals", "periods" and "durations". You can find out more here

Extracting Date Components

Sometimes during your data cleaning or analysis, you may need to extract a specific component of your date variable. A set of useful functions within the {lubridate} package allows you to do exactly this. For example, if we wanted to create a column with just the month that spraying started at each interval, we could use the `month()` function in the following way:

```
irs %>%
  mutate(month_start = month(start_date_default))%>%
  select(village, start_date_default, month_start)
```

As we can see here, this function returns the month as a number from 1-12. For our first observation, the spraying started during the fourth month, so in April. It's that simple! .If we want to have R display the month written out rather than the number underneath it, we can use the `label=TRUE` argument.

```
irs %>%
  mutate(month_start = month(start_date_default, label=TRUE))%>%
  select(village, start_date_default, month_start)
```

Likewise, if we wanted to extract the year, we would use the `year()` function.

```
irs %>%
  mutate(year_start = year(start_date_default)) %>%
  select(village, start_date_default, year_start)
```

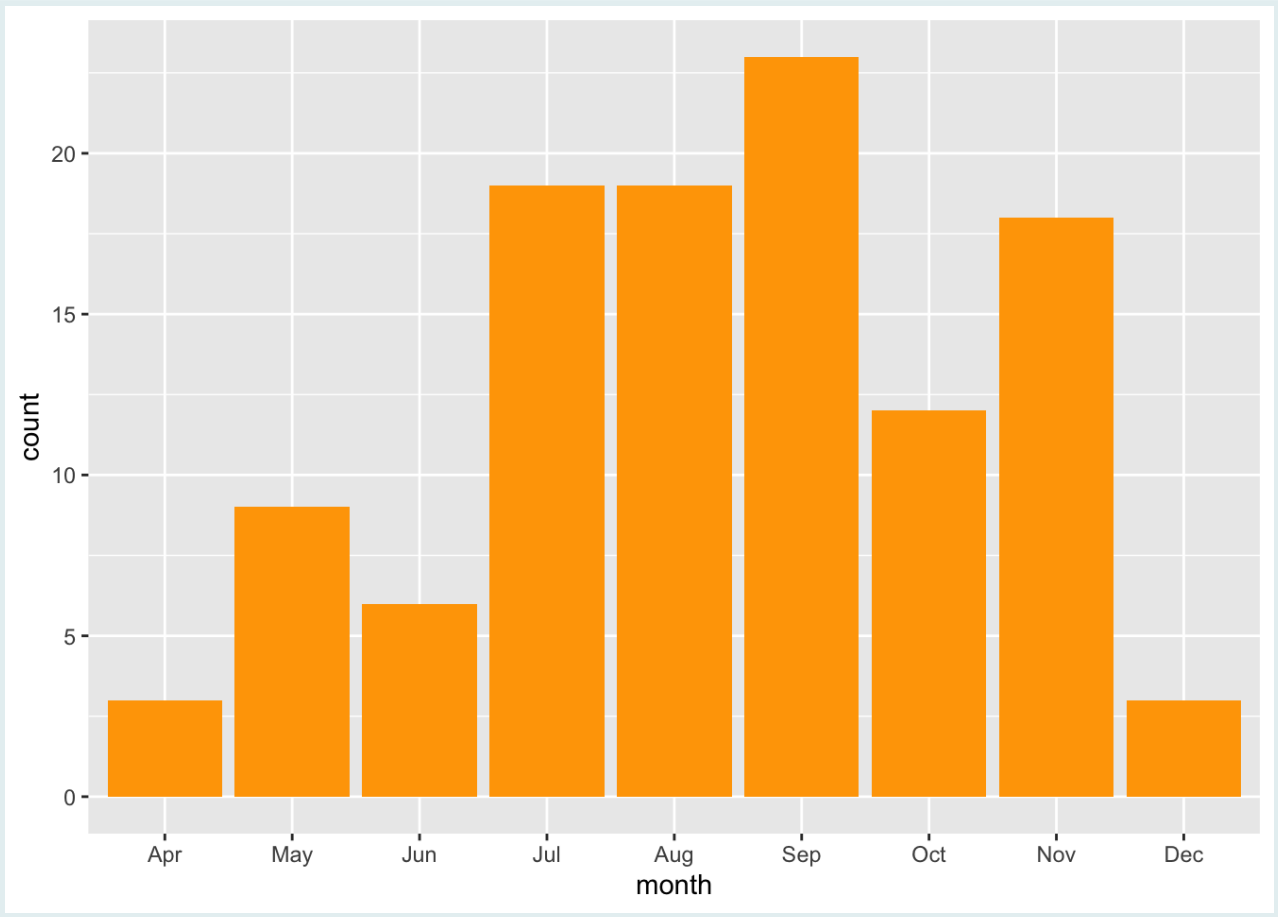
Extracting weekdays

Create a new variable called `wday_start` and extract the day of the week that the spraying started in the same way as above but with the `wday()` function. Try to display the days of the week written out rather than numerically.

One reason why you may want to extract specific date components is when you want to visualize your data.

For example, let's say we wanted to visualize the months when spraying starts, we can do this by creating a new month variable with the `month()` function, and plotting a bar graph with `geom_bar`.

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



Here we can see that most spraying campaigns started between July and November, with none occurring in the first three months of the year. The authors of the paper that this data was drawn from have stated that spraying campaigns aimed to finish just as the rainy season (November-April) in Malawi started. This fits the pattern observed.

Visualizing spray end months

Using the `irs` dataset, create a new graph showing the months when the spraying campaign ended and compare it to the graph of when they started. Do they have a similar pattern?

Rounding

Sometimes it's necessary to round our dates up or down if we want to analyze or vizualize our data in a meaningful way. First, let's see what we mean by rounding with a few simple examples.

Let's take the date March 17th 2012. If we wanted to round down to the nearest month, then we would use the `floor_date()` function from {lubridate} with the `unit="month"` argument.

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

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

As we can see, our date is now March 1st, 2012.

If we wanted to round up, we can use the `ceiling_date()` function. Let's try this on out on the date January 3rd 2020.

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

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

With `ceiling_date()`, January 3rd had been rounded up to February 1st.

Finally, we can also simply round without specifying up or down and the dates are automatically round to the nearest specified unit.

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

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

Here we can see that by rounding to the nearest month, November 3rd is round down to November 1st, and November 27th is round up to December 1st.

Rounding dates practice

We can also round up or down to the nearest year. What do you think the output would be if we round down the date November 29th 2001 to the nearest year:

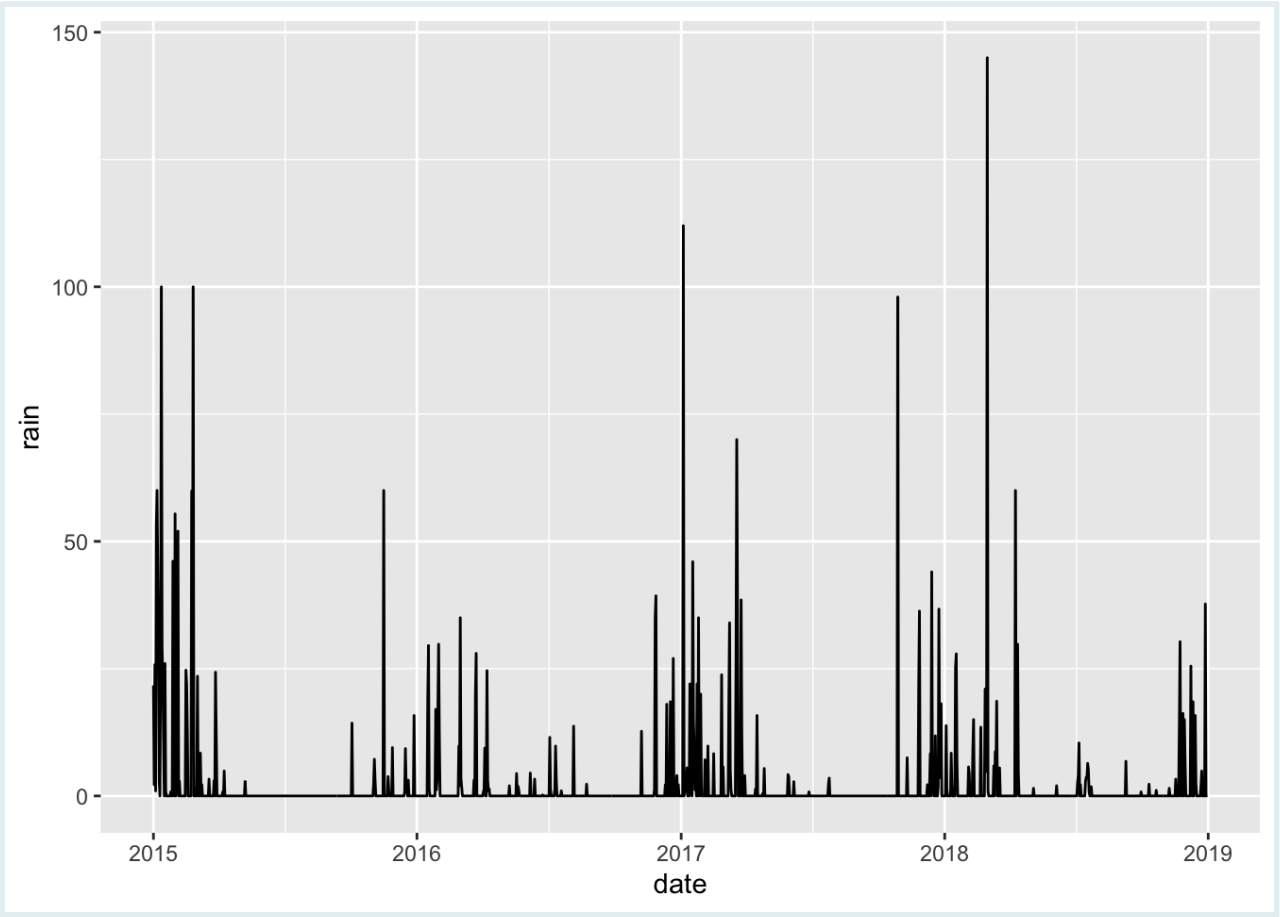
```
        date_round <- as.Date("2001-11-29")
        floor_date(date_round, unit="year")
```

Hopefully what we mean by rounding is a little bit more clear! So why could this be helpful with our data? Well, let's now turn to our weather data.

```
        weather
```

As we can see, our weather data is recorded daily, but this level of detail isn't ideal for studying how weather patterns affect malaria transmission, which follows a seasonal pattern. Daily weather data can be quite noisy given the significant variation from one day to the next:

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



Aside from being visually messy, it is a little bit difficult to see seasonal patterns. Monthly aggregation is a more effective approach for capturing seasonal variations and reducing noise.

If we wanted to plot the monthly average rainfall, our first attempt may be to use the `str_sub()` function to extract the first seven characters of our date (the month and year component).

```
weather %>%
  mutate(month_year=str_sub(date, 1, 7))
```

Next, we group by `month_year` to calculate the average rainfall for each month:

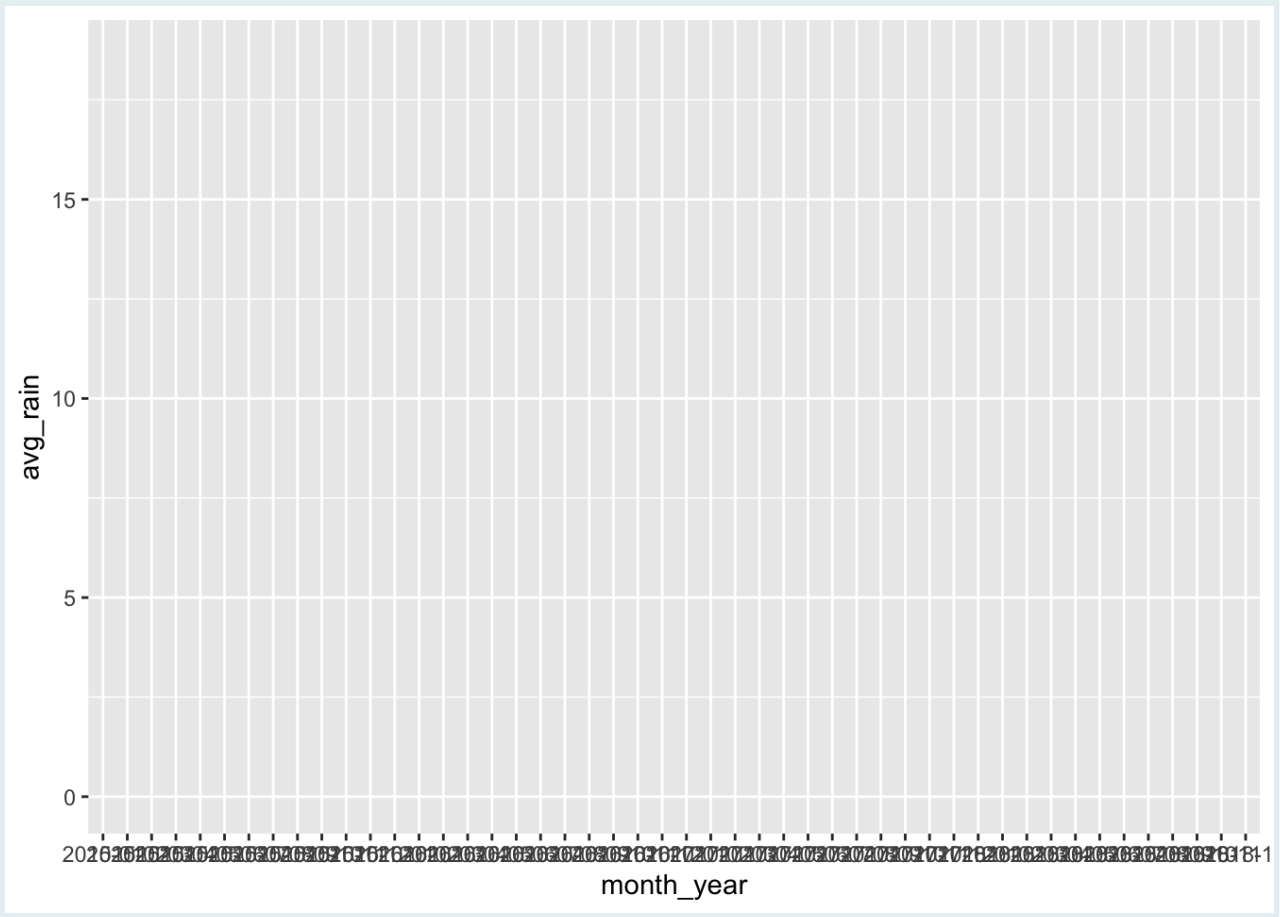
```
weather_summary_1 <-
  weather %>%
  mutate(month_year=str_sub(date, 1, 7)) %>%
  # group and summarise
  group_by(month_year) %>%
  summarise(avg_rain=mean(rain))
weather_summary_1
```

```
## # A tibble: 48 × 2
##    month_year avg_rain
##    <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
```

However, we encounter a problem when trying to plot this data. Our `month_year` variable is now a character, not a date. This means it's not continuous. Plotting a line graph with a non-continuous variable does not work:

```
weather_summary_1 %>%
  ggplot() +
  geom_line(aes(month_year, avg_rain))
```

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

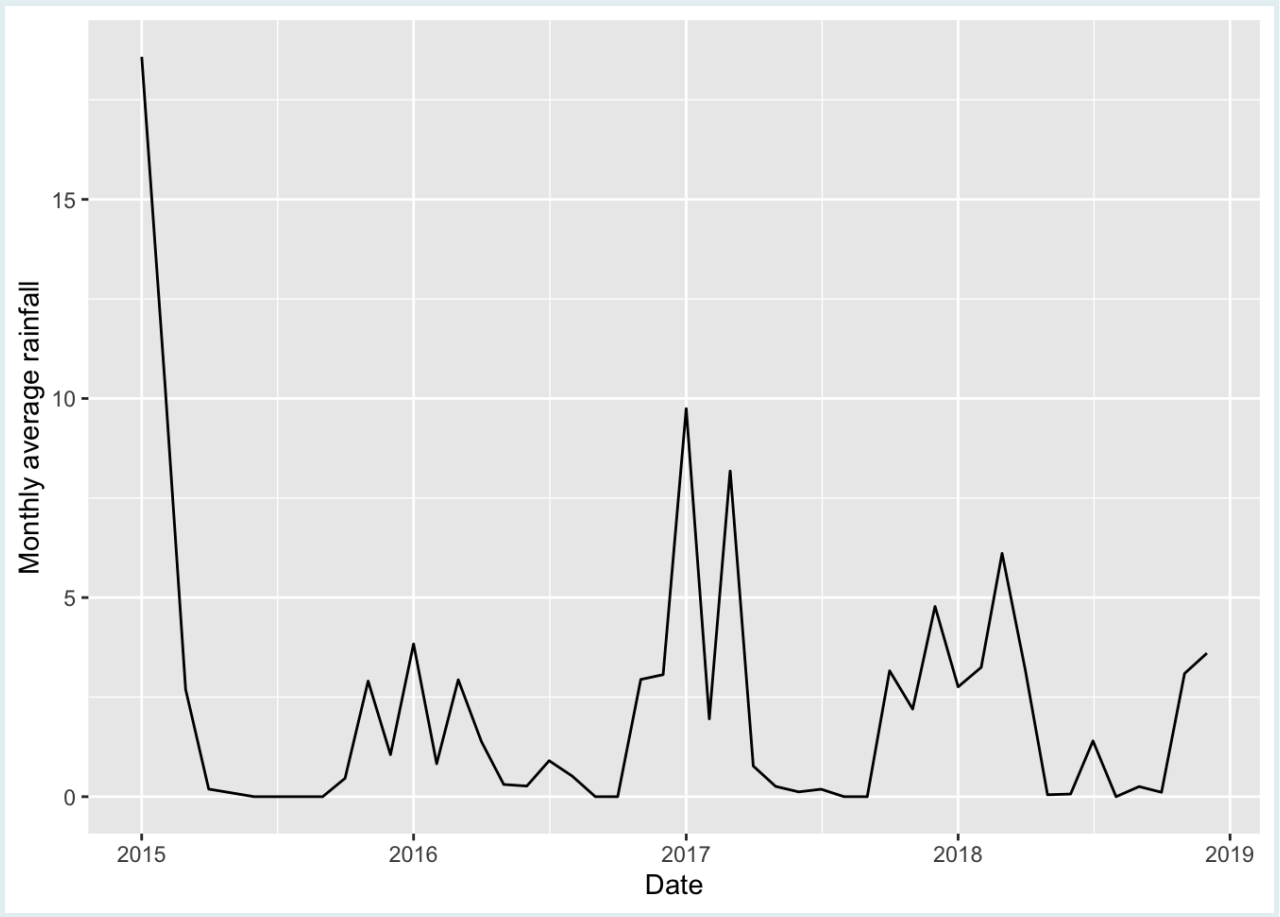


The best way to do this is to first round our dates down to the month using `floor_date()` function, then group our data by our new `month_year` variable, and then calculate the monthly average. Let's try it out now.

```
weather_summary_2 <- weather %>%
  mutate(month_year=floor_date(date, unit="month")) %>%
  group_by(month_year) %>%
  summarise(avg_rain=mean(rain))
weather_summary_2
```

Now we can plot our data and we'll have a graph of the average monthly rainfall over the 4 year spraying period.

```
weather_summary_2 %>%
  ggplot() +
  geom_line(aes(month_year, avg_rain)) +
  labs(x="Date", y="Monthly average rainfall")
```



That looks much better! Now we get a much clearer picture of seasonal trends and yearly variations.

Plot avg monthly min and max temperatures

Using the weather data, create a new line graph plotting the average monthly minimum and maximum temperatures from 2015-2019.

WRAP UP!

This lesson covered fundamental skills for working with dates in R - calculating intervals, extracting components, rounding, and creating time series visualizations. With these key building blocks now mastered, you can can now start to wrangle date data to uncover and analyze patterns over time.

Answer Key

Lubridate weeks

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

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

Lubridate intervals

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

```
## # A tibble: 112 × 1
##    spraying_time
##            <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
```

Extracting weekdays

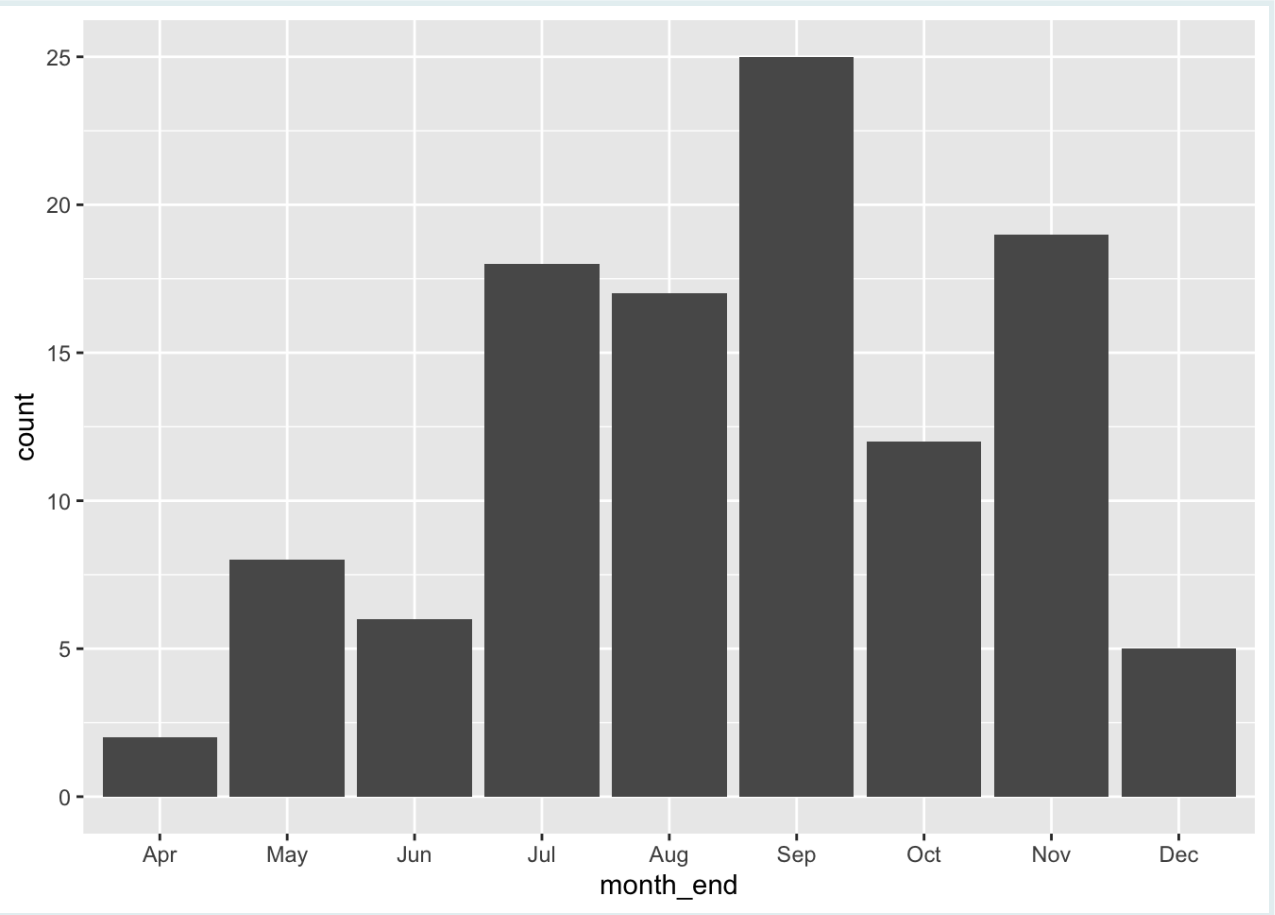
```
        irs %>%
          mutate(wday_start = wday(start_date_default, label = TRUE)) %>%
          select(wday_start)
```

```
## # A tibble: 112 × 1
##    wday_start
##    <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
```

Visualizing spray end months

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



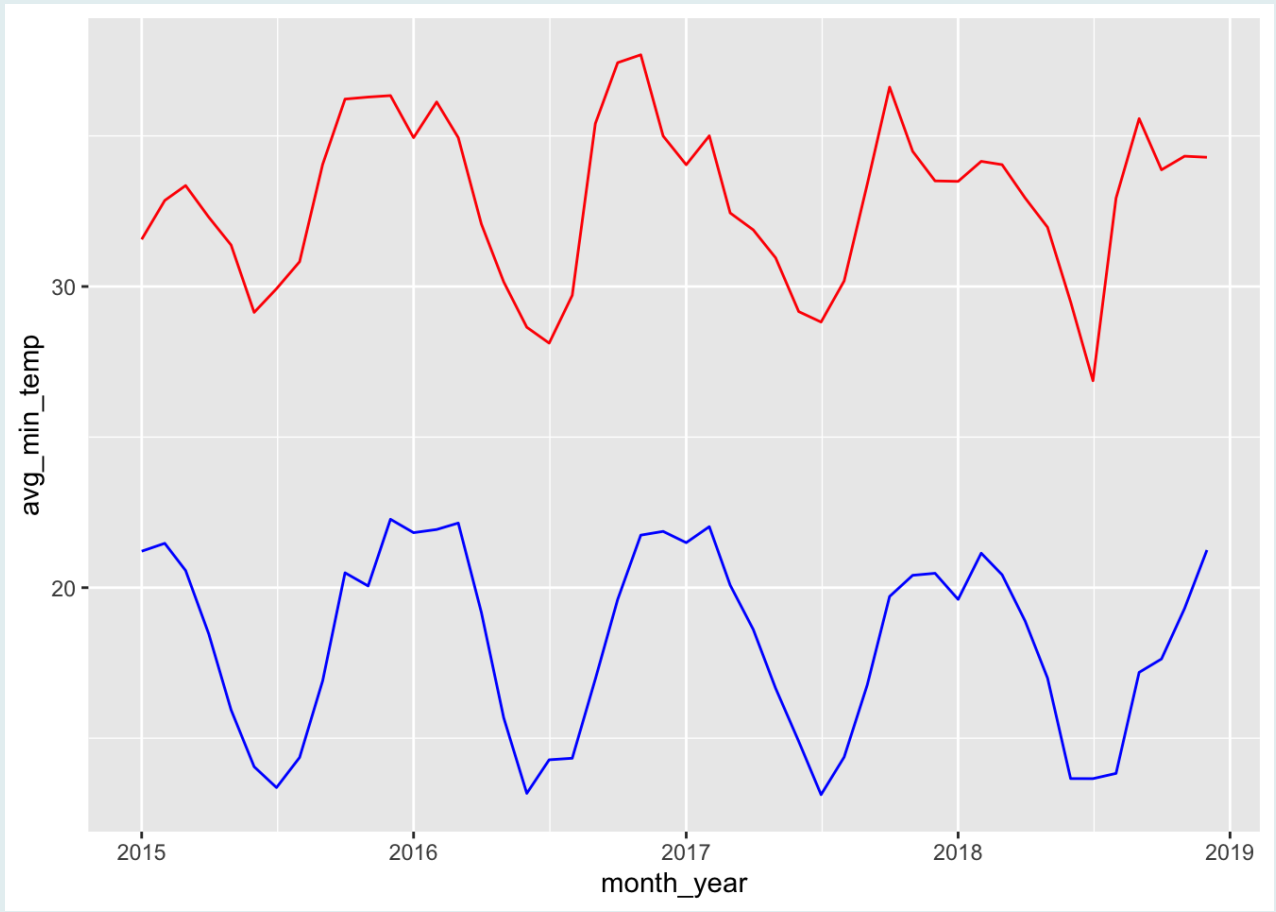
Rounding dates practice

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

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

Plot avg monthly min and max temperatures

```
weather %>%
  mutate(month_year = floor_date(date, unit="month")) %>%
  group_by(month_year) %>%
  summarise(avg_min_temp = mean(min_temp),
            avg_max_temp = mean(max_temp)) %>%
  ggplot() +
  geom_line(aes(x = month_year, y = avg_min_temp), color = "blue") +
  geom_line(aes(x = month_year, y = avg_max_temp), 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