ETC1010: Data Modelling and Computing

Lecture 3B: Dates and Times

Dr. Nicholas Tierney & Professor Di Cook

EBS, Monash U.

2019-08-16

Art by Allison Horst

Overview

- Working with dates
- Constructing graphics

Reminder re the assignment:

- Due 5pm **today**
- Submit by one person in the assignment group
- ED > assessments > upload your `Rmd`, and `html`, files.
- **One per group**
- **Remember to name your files as described in the submission**

The challenges of working with dates and times

- Conventional order of day, month, year is different across location
  - Australia: DD-MM-YYYY
  - America: MM-DD-YYYY
  - ISO 8601: YYYY-MM-DD

PUBLIC SERVICE ANNOUNCEMENT:

OUR DIFFERENT WAYS OF WRITING DATES AS NUMBERS CAN LEAD TO ONLINE CONFUSION. THAT'S WHY IN 1988 ISO SET A GLOBAL STANDARD NUMERIC DATE FORMAT.

THIS IS ***THE*** CORRECT WAY TO WRITE NUMERIC DATES:

2013-02-27

THE FOLLOWING FORMATS ARE THEREFORE DISCOURAGED:

02/27/2013 02/27/13 27/02/2013 27/02/13
20130227 2013.02.27 27.02.13 27-02-13
27.2.13 2013. II. 27. $\frac{27}{2}$-13 2013.158904109
MMXIII-II-XXVII MMXIII $\frac{\text{LVII}}{\text{CCCLXV}}$ 1330300800
((3+3)×(111+1)-1)×3/3-1/3^3 2013 

10/11011/1101 02/27/20/13 $\overset{2\ 3\ 1\ 4}{0\ 1\ 2\ 3\ 7}$ $\overset{5}{}$ $\overset{6\ 7}{}$ $\overset{8}{}$

The challenges of working with dates and times

- Number of units change:
  - Years do not have the same number of days (leap years)
  - Months have differing numbers of days. (January vs February vs September)
  - Not every minute has 60 seconds (leap seconds!)
- Times are local, for us. Where are you?
- Timezones!!!

The challenges of working with dates and times

- Representing time relative to it's type:
  - What day of the week is it?
  - Day of the month?
  - Week in the year?
- Years start on different days (Monday, Sunday, ...)

The challenges of working with dates and times

- Representing time relative to it's type:
  - Months could be numbers or names. (1st month, January)
  - Days could be numbers of names. (1st day....Sunday? Monday?)
  - Days and Months have abbreviations. (Mon, Tue, Jan, Feb)

The challenges of working with dates and times

- Time can be relative:
  - How many days until we go on holidays?
  - How many working days?

Lubridate

- Simplifies date/time by helping you:
  - Parse values
  - Create new variables based on components like month, day, year
  - Do algebra on time

Parsing dates & time zones using ymd ( )

`ymd()` can take a character input

```
ymd("20190810")
```

```
## [1] "2019-08-10"
```

`ymd()` can also take other kinds of separators

```
ymd("2019-08-10")
```

```
## [1] "2019-08-10"
```

```
ymd("2019/08/10")
```

```
## [1] "2019-08-10"
```

yeah, wow, I was actually surprised this worked

```
ymd("??2019-.-08//10---")
```

```
## [1] "2019-08-10"
```

Change the letters, change the output

```
mdy("10/15/2019")
```

```
## [1] "2019-10-15"
```

`mdy()` expects month, day, year.

`dmy()` expects day, month, year.

```
dmy("10/08/2019")
```

```
## [1] "2019-08-10"
```

Add a timezone

If you add a time zone, what changes?

```
ymd("2019-08-10", tz = "Australia/Melbourne")
```

```
## [1] "2019-08-10 AEST"
```

What happens if you try to specify different time zones?

```
ymd("2019-08-10",
    tz = "Africa/Abidjan")
```

```
## [1] "2019-08-10 GMT"
```

```
ymd("2019-08-10",
    tz = "America/Los_Angeles")
```

```
## [1] "2019-08-10 PDT"
```

A list of acceptable time zones can be found here (google wiki timezone database)

Timezones another way:

```
today()
```

```
## [1] "2019-08-16"
```

```
today(tz = "America/Los_Angeles")
```

```
## [1] "2019-08-15"
```

```
now()
```

```
## [1] "2019-08-16 07:02:57 AEST"
```

```
now(tz = "America/Los_Angeles")
```

```
## [1] "2019-08-15 14:02:57 PDT"
```

date and time: ymd_hms()

```
ymd_hms("2019-08-10 10:05:30",
        tz = "Australia/Melbourne")
```

```
## [1] "2019-08-10 10:05:30 AEST"
```

```
ymd_hms("2019-08-10 10:05:30",
        tz = "America/Los_Angeles")
```

```
## [1] "2019-08-10 10:05:30 PDT"
```

Extracting temporal elements

- Very often we want to know what day of the week it is
- Trends and patterns in data can be quite different depending on the type of day:
  - week day vs. weekend
  - weekday vs. holiday
  - regular saturday night vs. new years eve

Many ways of saying similar things

- Many ways to specify day of the week:
  - A number. Does 1 mean... Sunday, Monday or even Saturday???
  - Or text or or abbreviated text. (Mon vs. Monday)

Many ways of saying similar things

- Talking with people we generally use day name:
  - Today is Friday, tomorrow is Saturday vs Today is 5 and tomorrow is 6.
- But, doing data analysis on days might be useful to have it represented as a number:
  - e.g., Saturday - Thursday is 2 days (6 - 4)

The Many ways to say Monday (Pt 1)

```
wday("2019-08-12")
```

```
## [1] 2
```

```
wday("2019-08-12", label = TRUE)
```

```
## [1] Mon
## Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

The Many ways to say Monday (Pt 2)

```
wday("2019-08-12", label = TRUE, abbr = FALSE)
```

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

```
wday("2019-08-12", label = TRUE, week_start = 1)
```

```
## [1] Mon
## Levels: Mon < Tue < Wed < Thu < Fri < Sat < Sun
```

Similarly, we can extract what month the day is in.

```
month("2019-08-10")
```

```
## [1] 8
```

```
month("2019-08-10", label = TRUE)
```

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

```
month("2019-08-10", label = TRUE, abbr = FALSE)
```

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

Fiscally, it is useful to know what quarter the day is in.

```
quarter("2019-08-10")
```

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

```
semester("2019-08-10")
```

```
## [1] 2
```

Similarly, we can select days within a year.

```
yday("2019-08-10")
```

```
## [1] 222
```

Our Turn:

- Open rstudio.cloud and check out Lecture 3B and follow along.

Example: pedestrian sensor

Melbourne pedestrian sensor portal:

- Contains hourly counts of people walking around the city.
- Extract records for 2018 for the sensor at Melbourne Central
- Use lubridate to extract different temporal components, so we can study the pedestrian patterns at this location.

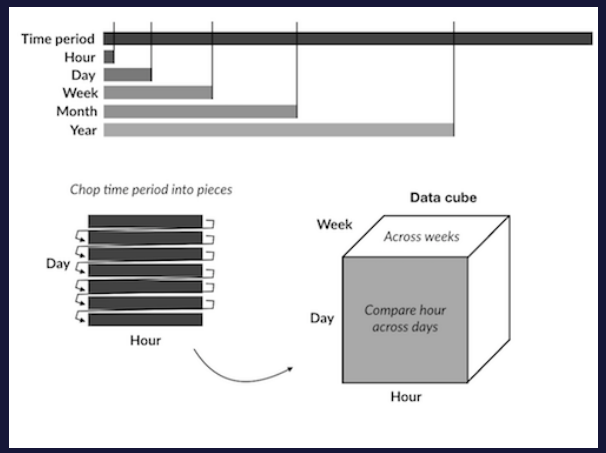
```
library(rwalkr)
walk_all <- melb_walk_fast(year = 2018)
library(dplyr)

walk <- walk_all %>% filter(Sensor == "Melbourne Central")

write_csv(walk, path = "data/walk_2018.csv")
```

```
walk <- readr::read_csv("data/walk_2018.csv")
walk
```

```
## # A tibble: 8,760 x 5
##    Sensor            Date_Time           Date        Time Count
##    <chr>             <dttm>              <date>     <dbl> <dbl>
##  1 Melbourne Central 2017-12-31 13:00:00 2018-01-01     0  2996
##  2 Melbourne Central 2017-12-31 14:00:00 2018-01-01     1  3481
##  3 Melbourne Central 2017-12-31 15:00:00 2018-01-01     2  1721
##  4 Melbourne Central 2017-12-31 16:00:00 2018-01-01     3  1056
##  5 Melbourne Central 2017-12-31 17:00:00 2018-01-01     4   417
##  6 Melbourne Central 2017-12-31 18:00:00 2018-01-01     5   222
##  7 Melbourne Central 2017-12-31 19:00:00 2018-01-01     6   110
##  8 Melbourne Central 2017-12-31 20:00:00 2018-01-01     7   180
##  9 Melbourne Central 2017-12-31 21:00:00 2018-01-01     8   205
## 10 Melbourne Central 2017-12-31 22:00:00 2018-01-01     9   326
## # … with 8,750 more rows
```

Let's think about the data structure.

- The basic time unit is hour of the day.
- Date can be decomposed into
  - month
  - week day vs weekend
  - week of the year
  - day of the month
  - holiday or work day



What format is walk in?

```
walk
```

```
## # A tibble: 8,760 x 5
##    Sensor            Date_Time           Date        Time Count
##    <chr>             <dttm>              <date>     <dbl> <dbl>
##  1 Melbourne Central 2017-12-31 13:00:00 2018-01-01     0  2996
##  2 Melbourne Central 2017-12-31 14:00:00 2018-01-01     1  3481
##  3 Melbourne Central 2017-12-31 15:00:00 2018-01-01     2  1721
##  4 Melbourne Central 2017-12-31 16:00:00 2018-01-01     3  1056
##  5 Melbourne Central 2017-12-31 17:00:00 2018-01-01     4   417
##  6 Melbourne Central 2017-12-31 18:00:00 2018-01-01     5   222
##  7 Melbourne Central 2017-12-31 19:00:00 2018-01-01     6   110
##  8 Melbourne Central 2017-12-31 20:00:00 2018-01-01     7   180
##  9 Melbourne Central 2017-12-31 21:00:00 2018-01-01     8   205
## 10 Melbourne Central 2017-12-31 22:00:00 2018-01-01     9   326
## # … with 8,750 more rows
```

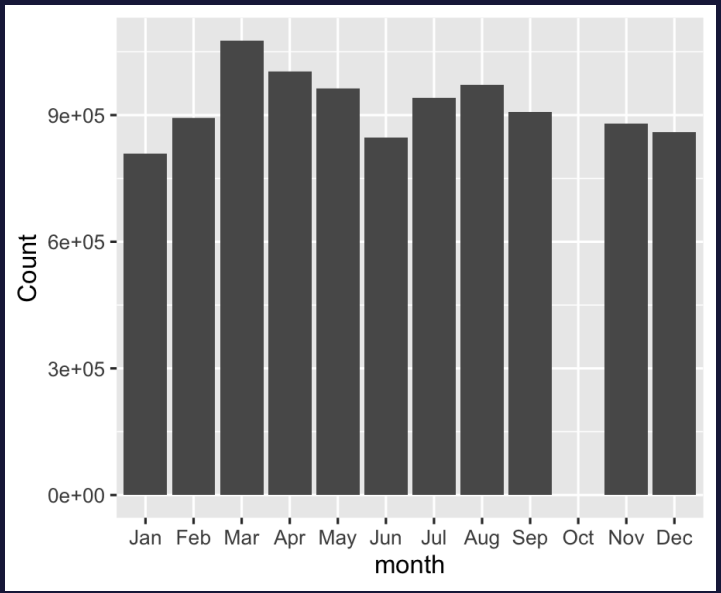
Create variables with these different temporal components.

```
walk_tidy <- walk %>%
  mutate(month = month(Date,  label = TRUE, abbr = TRUE),
         wday = wday(Date, label = TRUE, abbr = TRUE, week_start = 1))
walk_tidy
```

```
## # A tibble: 8,760 x 7
##    Sensor            Date_Time           Date        Time Count month wday
##    <chr>             <dttm>              <date>     <dbl> <dbl> <ord> <ord>
##  1 Melbourne Central 2017-12-31 13:00:00 2018-01-01     0  2996 Jan   Mon
##  2 Melbourne Central 2017-12-31 14:00:00 2018-01-01     1  3481 Jan   Mon
##  3 Melbourne Central 2017-12-31 15:00:00 2018-01-01     2  1721 Jan   Mon
##  4 Melbourne Central 2017-12-31 16:00:00 2018-01-01     3  1056 Jan   Mon
##  5 Melbourne Central 2017-12-31 17:00:00 2018-01-01     4   417 Jan   Mon
##  6 Melbourne Central 2017-12-31 18:00:00 2018-01-01     5   222 Jan   Mon
##  7 Melbourne Central 2017-12-31 19:00:00 2018-01-01     6   110 Jan   Mon
##  8 Melbourne Central 2017-12-31 20:00:00 2018-01-01     7   180 Jan   Mon
##  9 Melbourne Central 2017-12-31 21:00:00 2018-01-01     8   205 Jan   Mon
## 10 Melbourne Central 2017-12-31 22:00:00 2018-01-01     9   326 Jan   Mon
## # … with 8,750 more rows
```

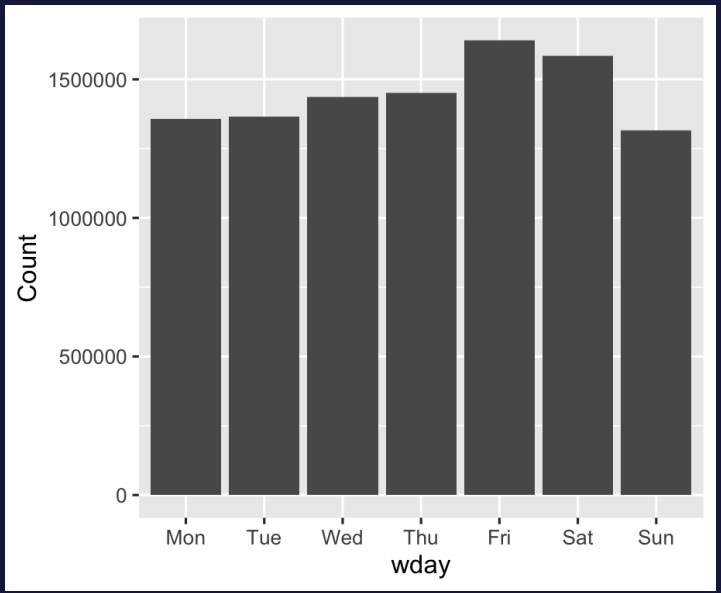
Pedestrian count per month

```
ggplot(walk_tidy,
       aes(x = month,
           y = Count)) +
  geom_col()
```



Pedestrian count per weekday

```
ggplot(walk_tidy,
       aes(x = wday,
           y = Count)) +
  geom_col()
```



What might be wrong with these interpretations?

- There might be a different number of days of the week over the year.
- This means that simply summing the counts might lead to a misinterpretation of pedestrian patterns.
- Similarly, months have different numbers of days.

Your Turn: Brainstorm with your table a solution, to answer these questions:

1. Are pedestrian counts different depending on the month?
2. Are pedestrian counts different depending on the day of the week?

What are the number of pedestrians per day?

```
walk_day <- walk_tidy %>%
  group_by(Date) %>%
  summarise(day_count = sum(Count, na.rm = TRUE))

walk_day
```

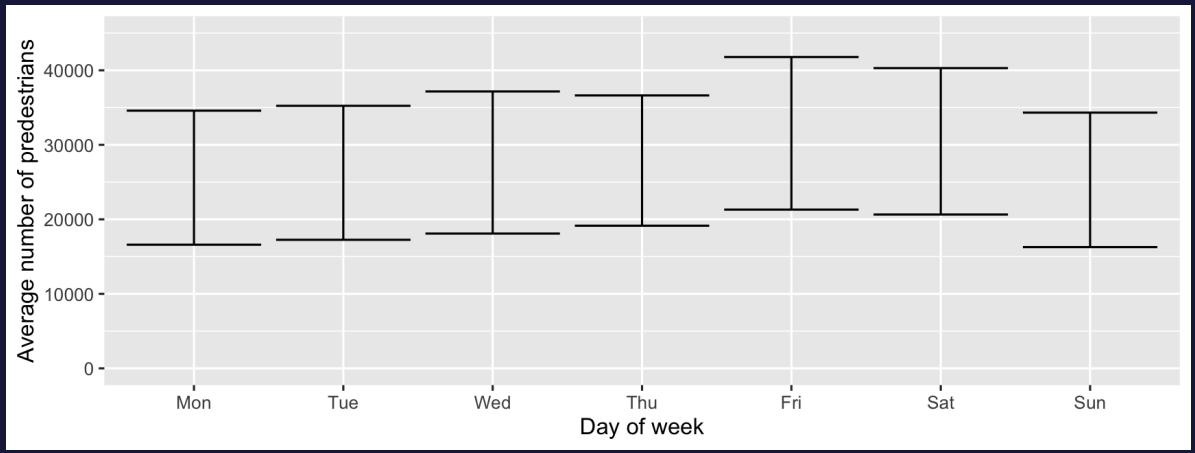
```
## # A tibble: 365 x 2
##    Date       day_count
##    <date>         <dbl>
##  1 2018-01-01     30832
##  2 2018-01-02     26136
##  3 2018-01-03     26567
##  4 2018-01-04     26532
##  5 2018-01-05     28203
##  6 2018-01-06     20845
##  7 2018-01-07     24052
##  8 2018-01-08     26530
##  9 2018-01-09     27116
## 10 2018-01-10     28203
## # … with 355 more rows
```

What are the mean number of people per weekday?

```
walk_week_day <- walk_day %>%
  mutate(wday = wday(Date, label = TRUE, abbr = TRUE, week_start = 1)) %>%
  group_by(wday) %>%
  summarise(m = mean(day_count, na.rm = TRUE),
            s = sd(day_count, na.rm = TRUE))

walk_week_day
```

```
## # A tibble: 7 x 3
##   wday       m      s
##   <ord>  <dbl>  <dbl>
## 1 Mon   25590.  8995.
## 2 Tue   26242.  8989.
## 3 Wed   27627.  9535.
## 4 Thu   27887.  8744.
## 5 Fri   31544. 10239.
## 6 Sat   30470.  9823.
## 7 Sun   25296.  9024.
```

```
ggplot(walk_week_day) +
  geom_errorbar(aes(x = wday, ymin = m - s, ymax = m + s)) +
  ylim(c(0, 45000)) +
  labs(x = "Day of week",
       y = "Average number of predestrians")
```

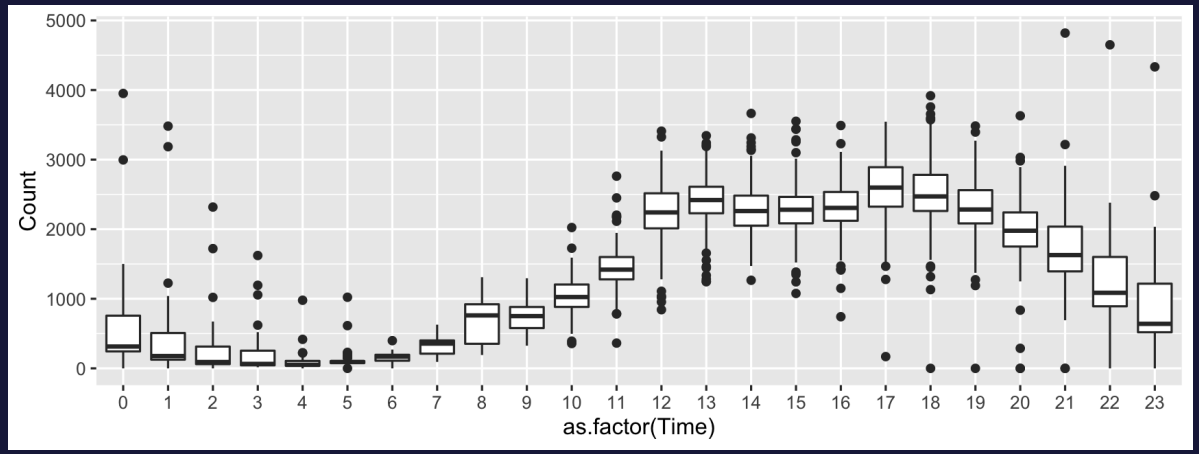


Distribution of counts

Side-by-side boxplots show the distribution of counts over different temporal elements.

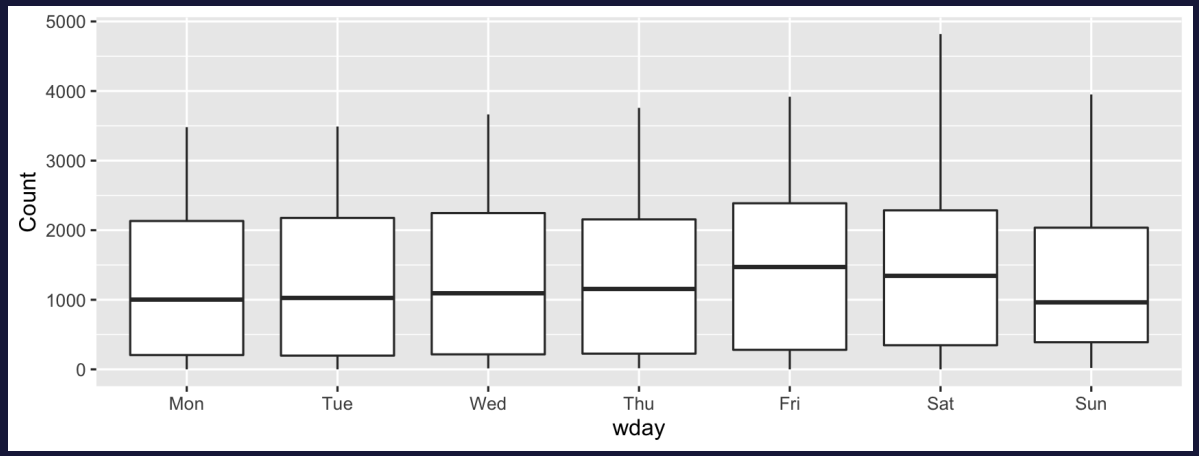
Hour of the day

```
ggplot(walk_tidy,
       aes(x = as.factor(Time), y = Count)) +
  geom_boxplot()
```



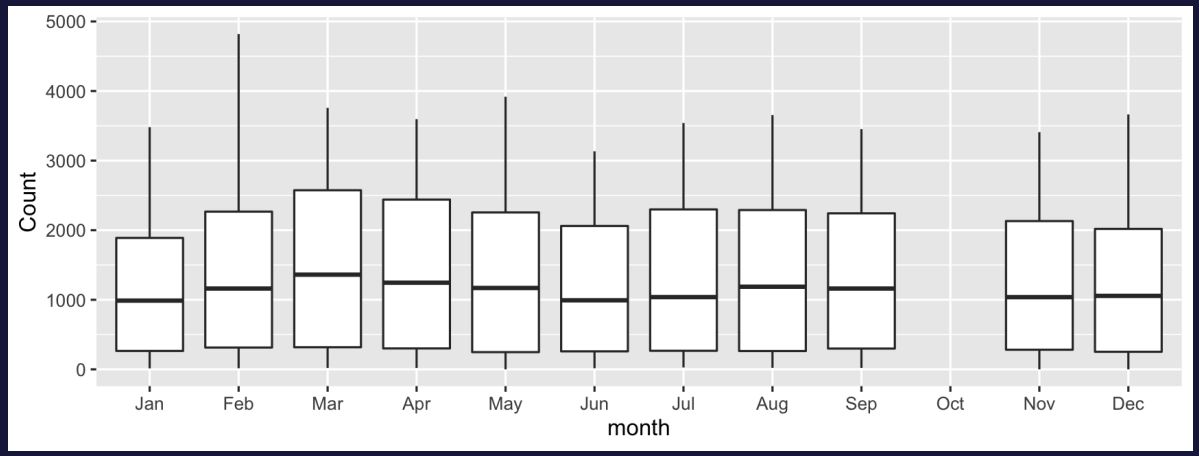
Day of the week

```
ggplot(walk_tidy,
       aes(x = wday,
           y = Count)) +
  geom_boxplot()
```



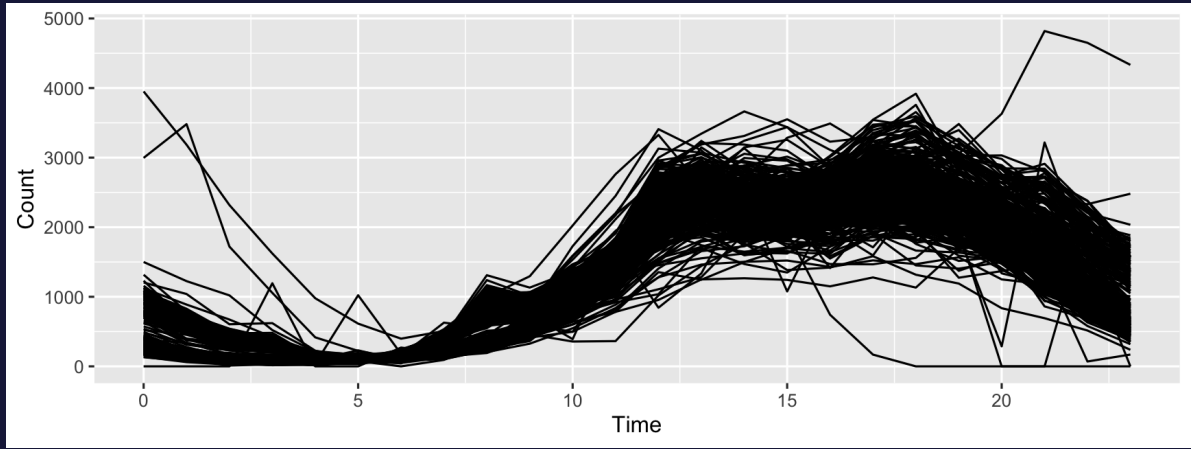
Month

```
ggplot(walk_tidy,
       aes(x = month,
           y = Count)) +
  geom_boxplot()
```



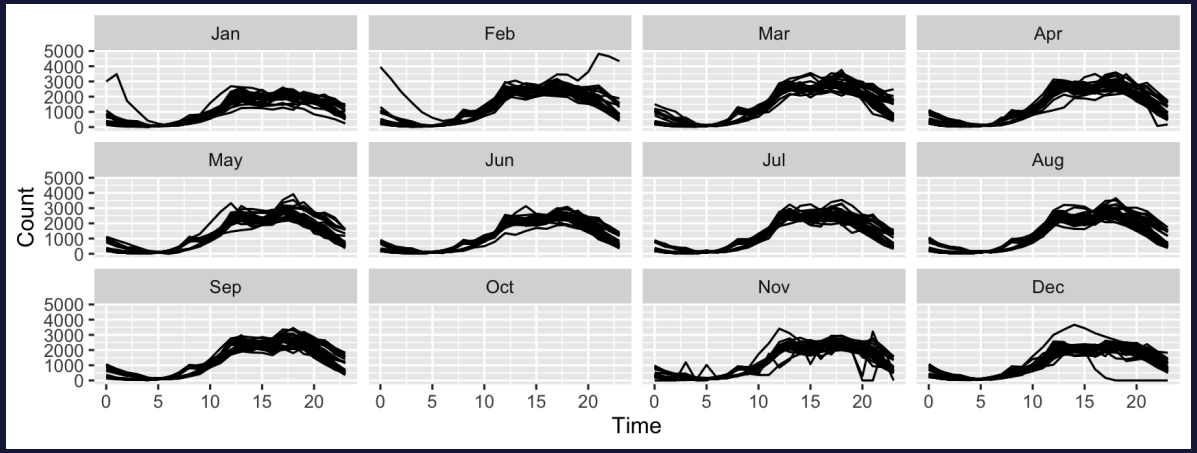
Time series plots: Lines show consecutive hours of the day.

```
ggplot(walk_tidy, aes(x = Time, y = Count, group = Date)) +
  geom_line()
```



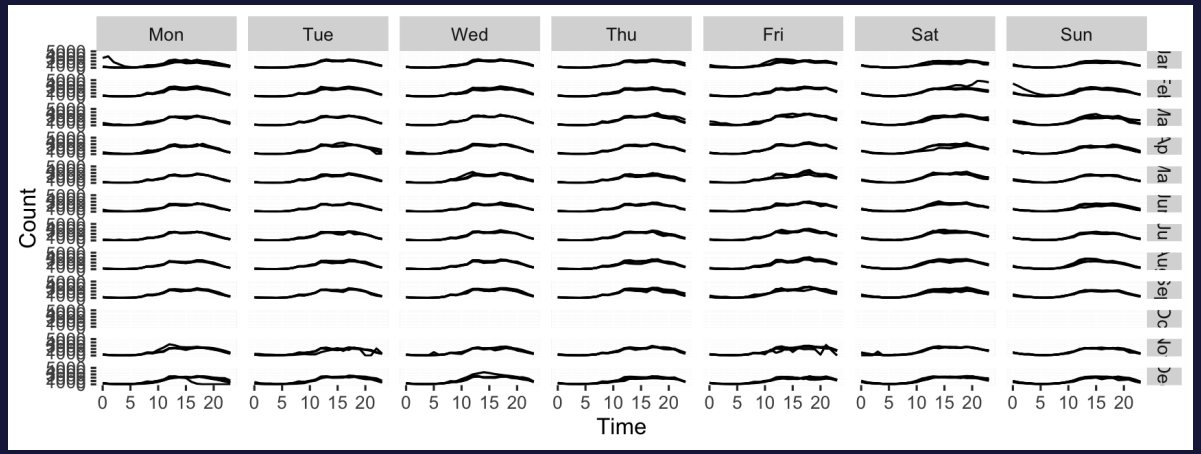
By month

```
ggplot(walk_tidy, aes(x = Time, y = Count, group = Date)) +
  geom_line() +
  facet_wrap( ~ month)
```



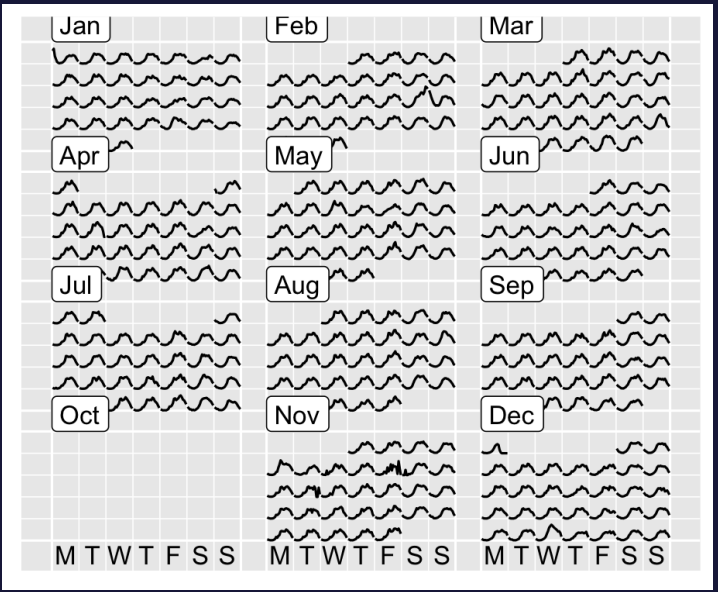
By week day

```
ggplot(walk_tidy, aes(x = Time, y = Count, group = Date)) +
  geom_line() +
  facet_grid(month ~ wday)
```



Calendar plots

```
library(sugrrants)

walk_tidy_calendar <-
  frame_calendar(walk_tidy,
                 x = Time,
                 y = Count,
                 date = Date,
                 nrow = 4)

p1 <- ggplot(walk_tidy_calendar,
       aes(x = .Time,
           y = .Count,
           group = Date)) +
  geom_line()

prettify(p1)
```



Holidays

```
library(tsibble)
library(sugrrants)
library(timeDate)
vic_holidays <- holiday_aus(2018, state =
vic_holidays
```

```
## # A tibble: 12 x 2
##    holiday          date
##    <chr>            <date>
##  1 New Year's Day   2018-01-01
##  2 Australia Day    2018-01-26
##  3 Labour Day       2018-03-12
##  4 Good Friday      2018-03-30
##  5 Easter Saturday  2018-03-31
##  6 Easter Sunday    2018-04-01
##  7 Easter Monday    2018-04-02
##  8 ANZAC Day        2018-04-25
##  9 Queen's Birthday 2018-06-11
## 10 Melbourne Cup    2018-11-06
## 11 Christmas Day    2018-12-25
## 12 Boxing Day       2018-12-26
```

pull-right[

]

Holidays

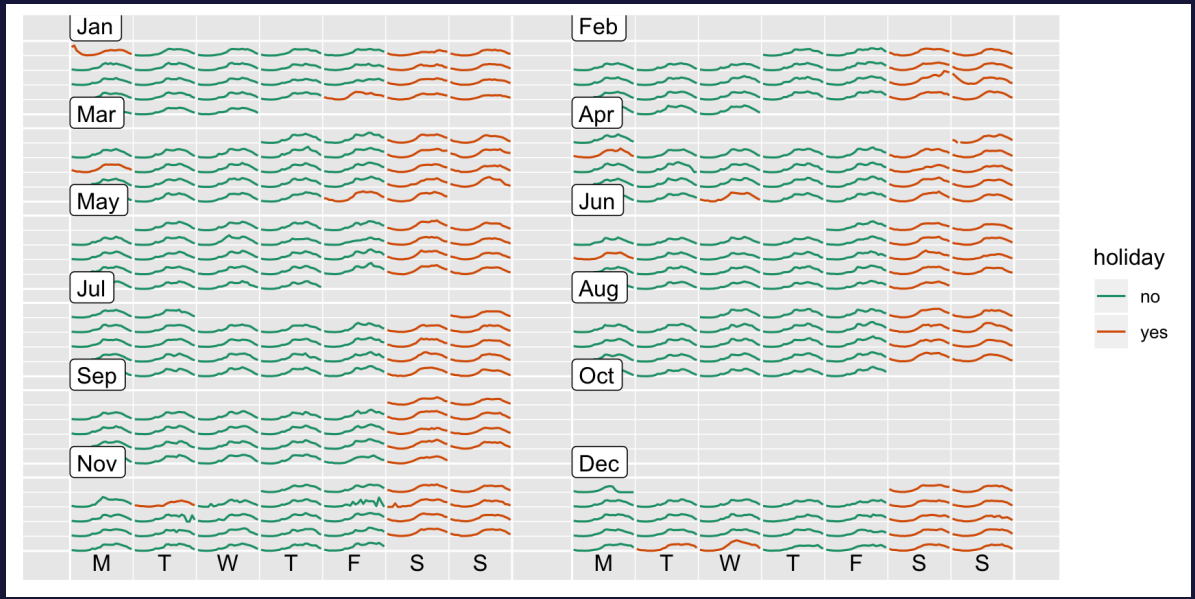
```
walk_holiday <- walk_tidy %>%
  mutate(holiday = if_else(condition = Date %in% vic_holidays$date,
                           true = "yes",
                           false = "no")) %>%
  mutate(holiday = if_else(condition = wday %in% c("Sat", "Sun"),
                           true = "yes",
                           false = holiday))
walk_holiday
```

```
## # A tibble: 8,760 x 8
##    Sensor            Date_Time           Date        Time Count month wday  holiday
##    <chr>             <dttm>              <date>     <dbl> <dbl> <ord> <ord> <chr>
##  1 Melbourne Central 2017-12-31 13:00:00 2018-01-01     0  2996 Jan   Mon   yes
##  2 Melbourne Central 2017-12-31 14:00:00 2018-01-01     1  3481 Jan   Mon   yes
##  3 Melbourne Central 2017-12-31 15:00:00 2018-01-01     2  1721 Jan   Mon   yes
##  4 Melbourne Central 2017-12-31 16:00:00 2018-01-01     3  1056 Jan   Mon   yes
##  5 Melbourne Central 2017-12-31 17:00:00 2018-01-01     4   417 Jan   Mon   yes
##  6 Melbourne Central 2017-12-31 18:00:00 2018-01-01     5   222 Jan   Mon   yes
##  7 Melbourne Central 2017-12-31 19:00:00 2018-01-01     6   110 Jan   Mon   yes
##  8 Melbourne Central 2017-12-31 20:00:00 2018-01-01     7   180 Jan   Mon   yes
##  9 Melbourne Central 2017-12-31 21:00:00 2018-01-01     8   205 Jan   Mon   yes
## 10 Melbourne Central 2017-12-31 22:00:00 2018-01-01     9   326 Jan   Mon   yes
```

Holidays

```
walk_holiday_calendar <- frame_calendar(data = walk_holiday,
                                        x = Time,
                                        y = Count,
                                        date = Date,
                                        nrow = 6)

p2 <- ggplot(walk_holiday_calendar,
       aes(x = .Time,
           y = .Count,
           group = Date,
           colour = holiday)) +
  geom_line() +
  scale_colour_brewer(palette = "Dark2")
```

Holidays


Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
M T W T F S S
M T W T F S S
holiday
no
yes

References

- suggrants
- tsibble
- lubridate
- dplyr
- timeDate
- rwalkr

Your Turn:

- Do the lab exercises
- Take the lab quiz
- Use the rest of the lab time to coordinate with your group on the first assignment.

Share and share alike