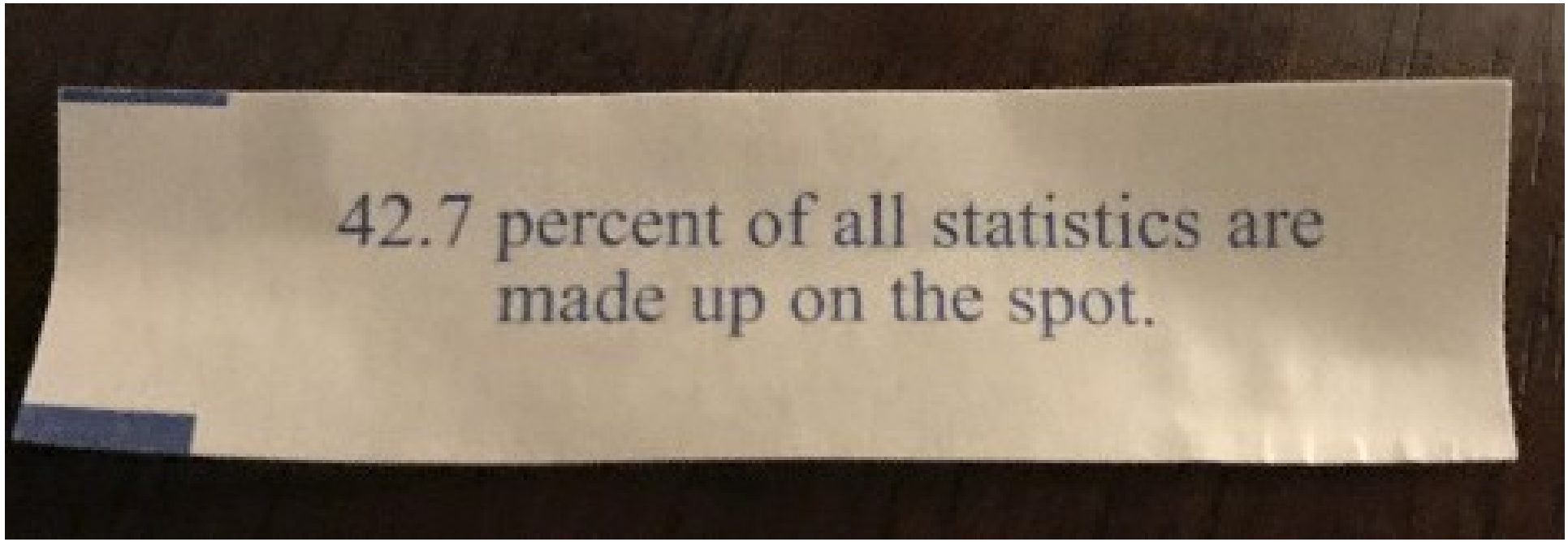
Data Analytics
CS301
Dates and Times

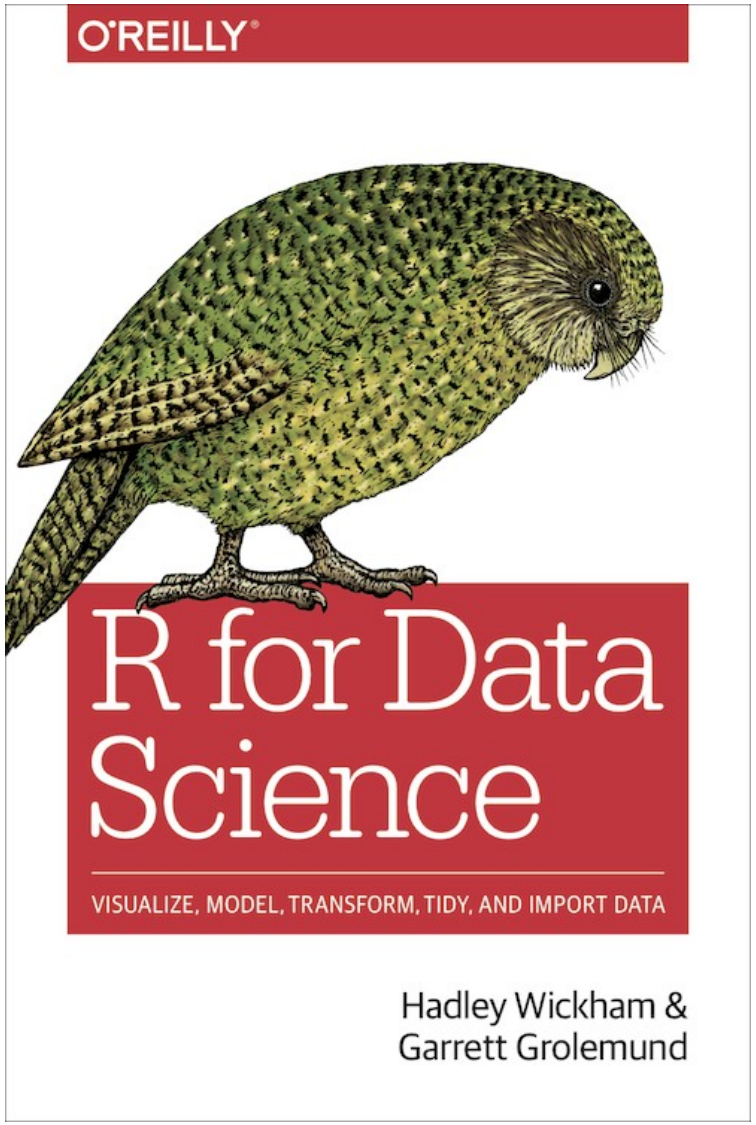
Fall 2018

Oliver Bonham-Carter



Seriously?!

Where in the Web?
Where in the Book?



- Note the chapter differences!
- Book:
  - Chap 13
- Web:
  - Chap 16

- Dates and Times with Lubridate

Dates and Times in R

- How do we deal when time or dates are a part of our analysis?
- How do we determine if our data spreads across a leap year?
- What if we measure our observations using a minute-by-minute time frame for some series of years? If there is a leap year, is there a problem?

Libraries

```
# Remember. You do not need to
reinstall these libraries each time
you use them; only import them

#install.packages("lubridate")

library(lubridate)

library(tidyverse)

library(nycflights13)
```

What time is it?

```
# Show today's data
today()
# Show time and data for right now
now()
TodayData <- today()
TimeNow <- now()
```

- What is the ***type of*** these variables?
- Now, try adding some seconds to the TimeNow variable. Can you add some days or years to this variable?

Three Ways to Create Date and Time

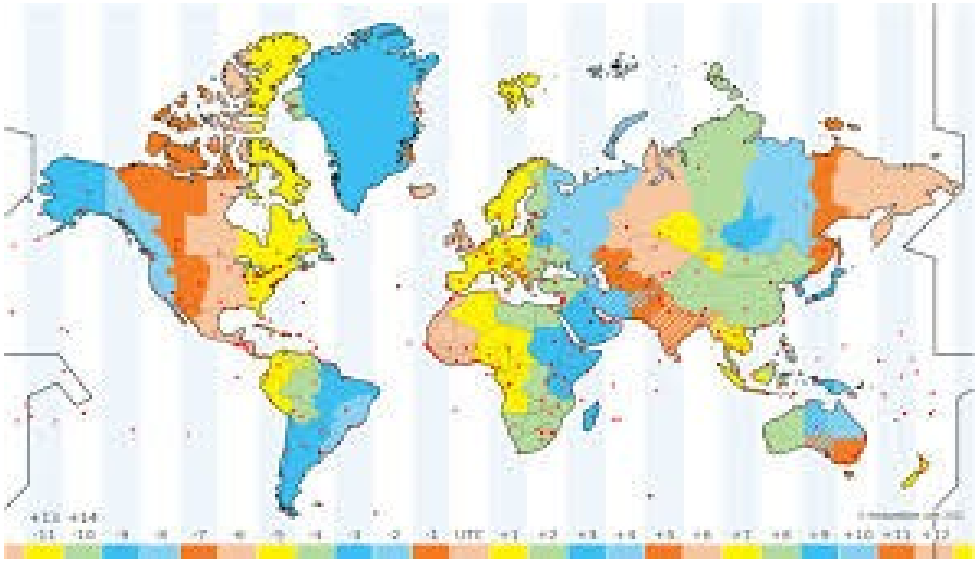
- Depending on what you want to do with your code, you can work with dates:
- From a string.
- From individual date-time components.
- From an existing date/ time object.

Date Strings

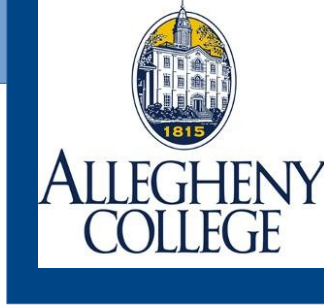
```
# Use the built-in code provided by lubridate to automatically format dates.

# Specify the order of the components: year, month and day, then arrange “y”, “m”, and “d” in the same order.

ymd("2017-10-06")

mdy("October 6th 2017")

dmy("06-Oct-2017")

ymd(20171006)
```

Date and Time Strings

```
# Add the time in with the date and
Coordinated Universal Time (UTC)

ymd_hms("2017-10-06 18:10:42")

mdy_hm("10/06/2017 08:10")

# Specify the timezone of the time

ymd(20171006, tz = "UTC")
```



as_datetime() and as_date()




```
# Switch between a date-time and a date
today()
# "2018-10-16"
as_datetime(today())
# "2018-10-16 UTC"
now()
# "2018-10-16 02:57:16 GMT"
as_date(now())
# "2018-10-16"
```

Use the lubridate function to parse each of the following dates:

```
# Consider these!
d1 <- "January 1, 2010"
d2 <- "2015-Mar-07"
d3 <- "06-Jun-2017"
d4 <- c("August 19 (2015)", "July 1 (2015)")
d5 <- "12/30/14" # Dec 30, 2014
```

Commands to try:
dym(), mdy() and ymd(),

Parsing Specific Details

```
now() %>% ymd_hms()
datetime <- ymd_hms("2018-10-16 12:01:13")
year(datetime)
month(datetime)
mday(datetime) # (day of the month)
yday(datetime) # (day of the year)
wday(datetime) # (day of the week)
hour(datetime)
minute(datetime)
second(datatime)
```

Parsing Specific Details

```
datetime <- ymd_hms("2018-10-16 12:01:13")
# Get the month as a String
month(datetime, label = TRUE) # short string
month(datetime, label = TRUE, abbr = FALSE) # long


# Get the day of the week as a string
wday(datetime, label = TRUE, abbr = FALSE)
```

Parse Time and Date to Make Time Table

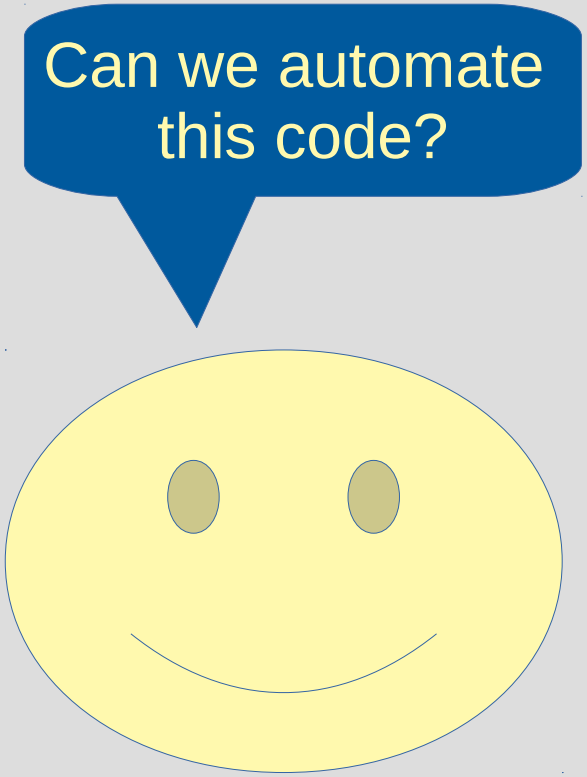
```
# load libraries if this has not been done

#library(tidyverse)

#library(nycflights13)

#View(flights)

timeTable <- flights %>% select(year,
month, day, hour, minute)

#what is this new table?

View(timeTable)
```

Further Formatting

```
# Format a new column using mutate
with the time data.


flights %>% select(year, month, day,
hour, minute) %>%  mutate(departure =
 make_datetime(year, month, day,
hour, minute))
```

Further Formatting

```
flights  %>%
select(year, month, day, hour, minute)  %>%
mutate(departure = make_datetime(year, month,
day, hour, minute))
```




```
> flights %>% select(year, month, day, hour, minute) %>%
mutate(departure = make_datetime(year, month, day, hour,
minute))
# A tibble: 336,776 x 6
    year month   day  hour minute           departure
   <int> <int> <int> <dbl>  <dbl>              <dttm>
 1  2013     1     1     5     15 2013-01-01 05:15:00
 2  2013     1     1     5     29 2013-01-01 05:29:00
 3  2013     1     1     5     40 2013-01-01 05:40:00
 4  2013     1     1     5     45 2013-01-01 05:45:00
 5  2013     1     1     6      0 2013-01-01 06:00:00
 6  2013     1     1     5     58 2013-01-01 05:58:00
 7  2013     1     1     6      0 2013-01-01 06:00:00
 8  2013     1     1     6      0 2013-01-01 06:00:00
 9  2013     1     1     6      0 2013-01-01 06:00:00
10  2013     1     1     6      0 2013-01-01 06:00:00
# ... with 336,766 more rows
```

Format All Time Data In The Flights Table

Let's automate the date and time formatting by creating a function to perform the work!

```
#Define a function with inputs: year, month, day and time



make_datetime_100 <- function(year, month, day, time) { make_datetime(year, month, day, time %/% 100, time %% 100) }
```

Format All Time Data In The Flights Table

```
# Keep the dep_time and arr_time variables
(rows) that exist (are not na)


flights_dt <- flights %>%
  filter(!is.na(dep_time), !
is.na(arr_time))
 %>%
# filter out the time components and pass
the time data to a mutate function (next).
```

Code: part 1 of 3

Format All Time Data In The Flights Table

```
# format the individual times of the column:
mutate(
dep_time = make_datetime_100(year, month,
day, dep_time),
arr_time = make_datetime_100(year, month,
day, arr_time),
sched_dep_time = make_datetime_100(year,
month, day, sched_dep_time),
sched_arr_time = make_datetime_100(year,
month, day, sched_arr_time)) %>%
```

Code: part 2 of 3

Format All Time Data In The Flights Table


```
# pull out the column headers (names) ending with “delay” or “time”

select(origin, dest, ends_with("delay"), ends_with("time"))
```

Code: part 3 of 3

All Formatting Code in One Block

```
make_datetime_100 <- function(year, month, day, time) {
  make_datetime(year, month, day, time %/% 100, time %% 100) }

flights_dt <- flights %>%
  filter(!is.na(dep_time), !is.na(arr_time)) %>%
  mutate(
    dep_time = make_datetime_100(year, month, day, dep_time),
    arr_time = make_datetime_100(year, month, day, arr_time),
    sched_dep_time = make_datetime_100(year, month, day,
sched_dep_time),
    sched_arr_time = make_datetime_100(year, month, day,
sched_arr_time)
  ) %>%
  select(origin, dest, ends_with("delay"), ends_with("time"))

flights_dt %>% ggplot(aes(dep_time)) + geom_freqpoly(binwidth =
86400)
```

All code together with plotter

The Formatted Times in Table

```
View(flights_dt)
```

Now our time data looks like this and is easier to manipulate

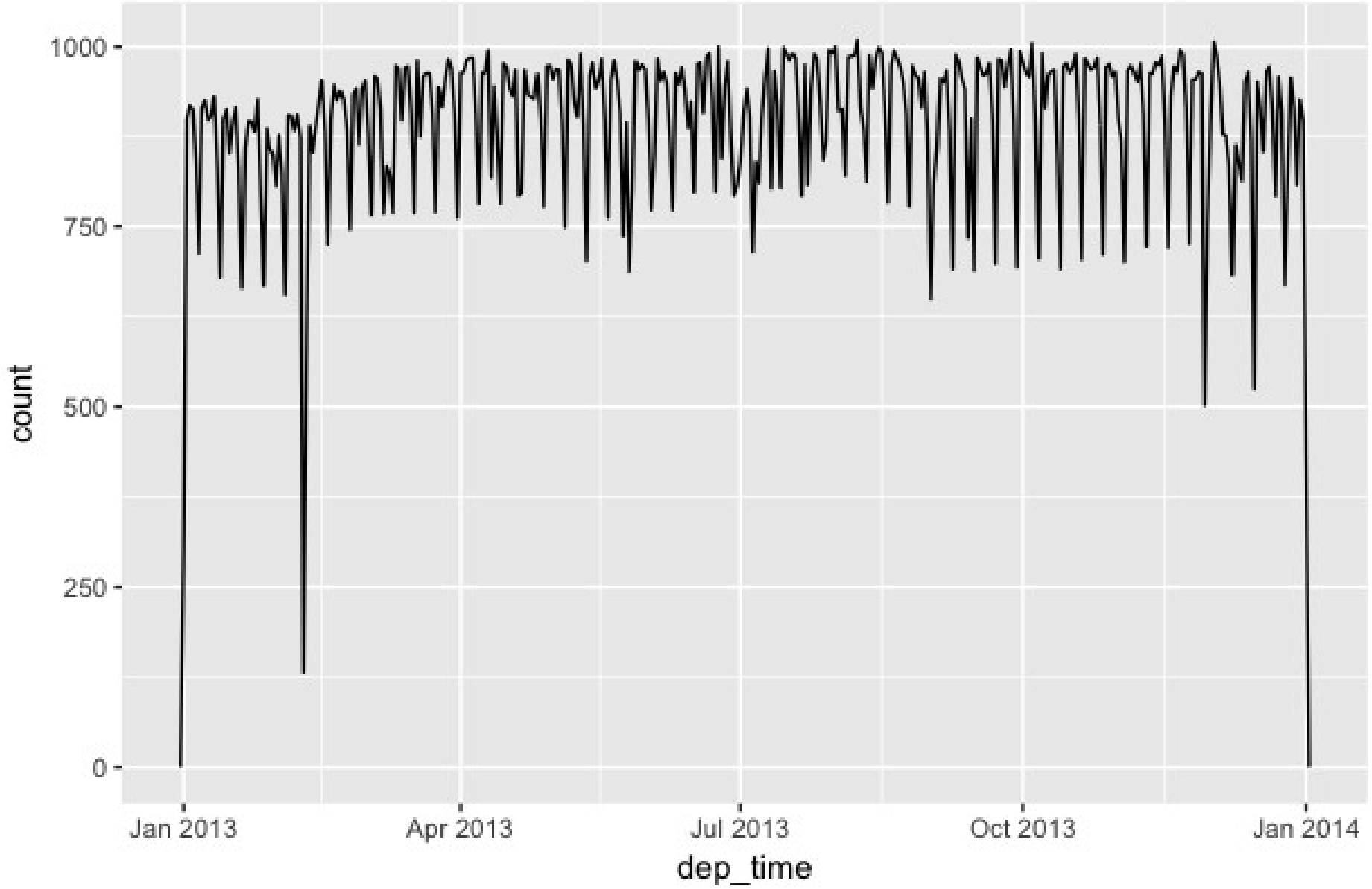
	origin	dest	dep_delay	arr_delay	dep_time	sched_dep_time
1	EWR	IAH	2	11	2013-01-01 05:17:00	2013-01-01 05:15:00
2	LGA	IAH	4	20	2013-01-01 05:33:00	2013-01-01 05:29:00
3	JFK	MIA	2	33	2013-01-01 05:42:00	2013-01-01 05:40:00
4	JFK	BQN	-1	-18	2013-01-01 05:44:00	2013-01-01 05:45:00
5	LGA	ATL	-6	-25	2013-01-01 05:54:00	2013-01-01 06:00:00
6	EWR	ORD	-4	12	2013-01-01 05:54:00	2013-01-01 05:58:00
7	EWR	FLL	-5	19	2013-01-01 05:55:00	2013-01-01 06:00:00
8	LGA	IAD	-3	-14	2013-01-01 05:57:00	2013-01-01 06:00:00
9	JFK	MCO	-3	-8	2013-01-01 05:57:00	2013-01-01 06:00:00

Quick Analysis:
What do the dep_times counts look like over the year, in days?

```
# We have formatted the time data and have made
it convenient to insert into other functions.
Let's plot it.

#We visualize the distribution of departure times
across the year in day-size binwidths.

# note: 86400 seconds = 1 day

flights_dt %>% ggplot(aes(dep_time))
+ geom_freqpoly(binwidth = 86400)
```

Binwidths determine the day-intervals during the year.

Data By Year: Visualization of *dep_time*

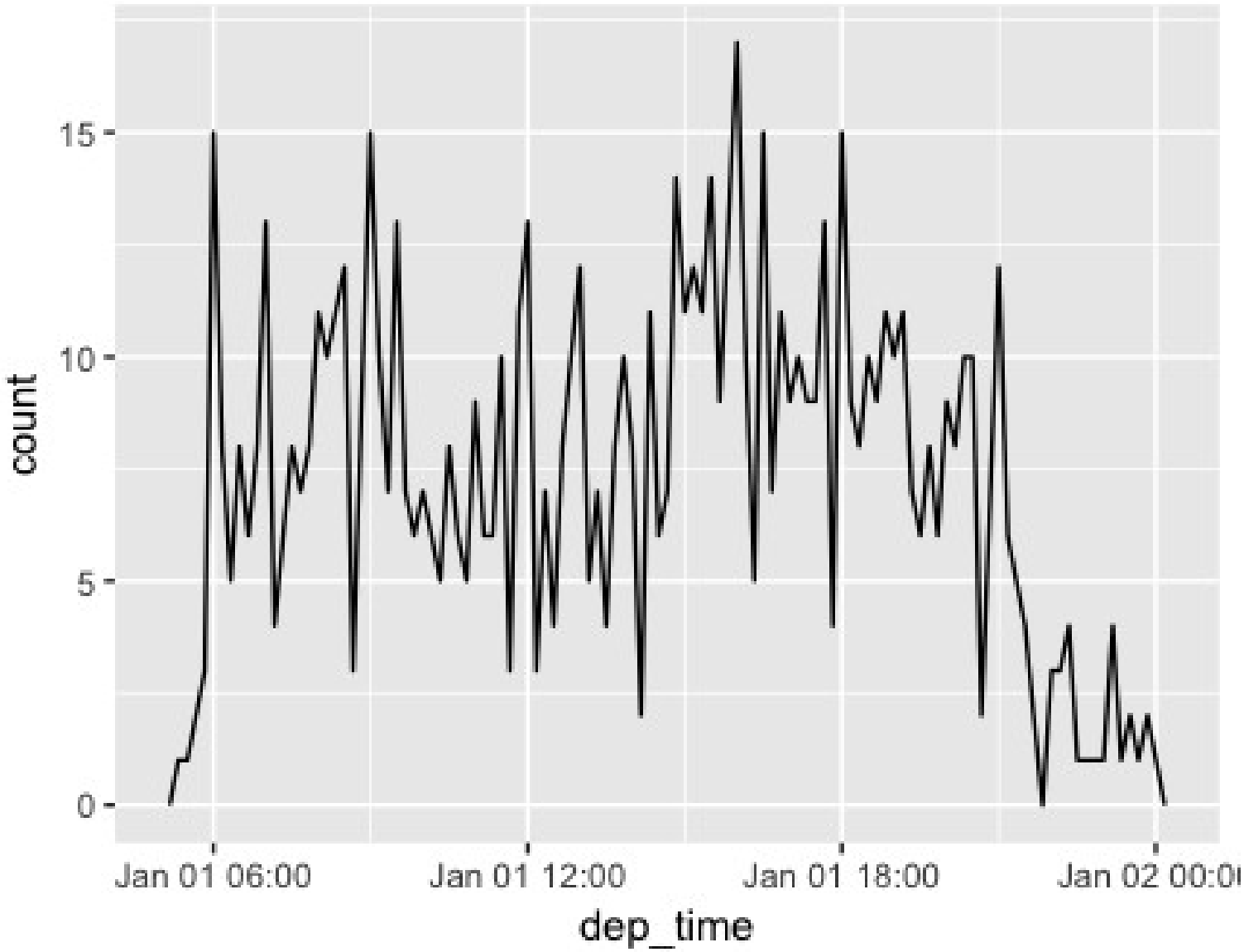


Quick Analysis:
What do the dep_times counts look like over day, in minutes?

```
# Now visualize this dep_time data by day.

flights_dt %>% filter(dep_time < ymd(20130102))
%>% ggplot(aes(dep_time)) +

geom_freqpoly(binwidth = 600)

# Note: 600 seconds = 10 minutes
```

Binwidths determine the minute-intervals during the day

Data By Day: Visualization of *dep_time*

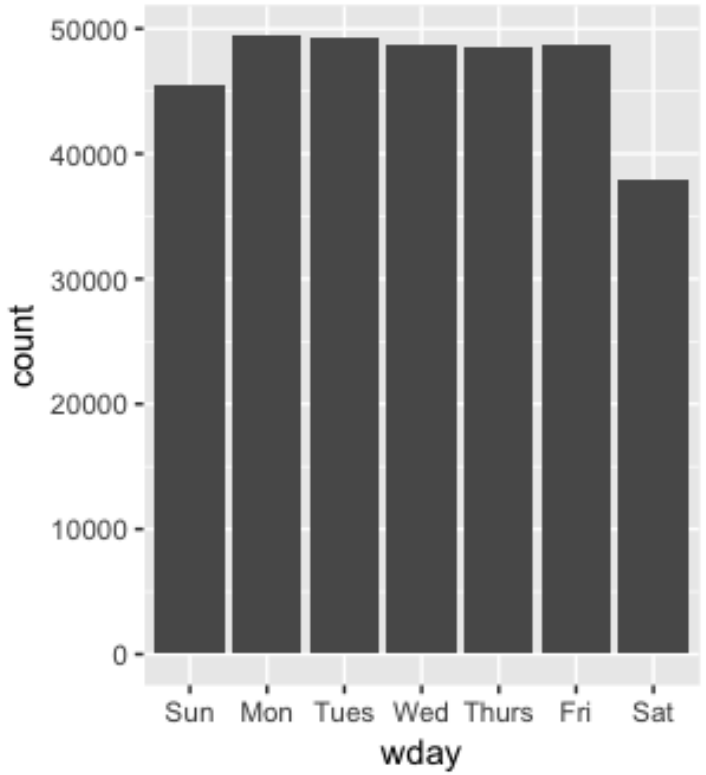


Quick Analysis!

When are most of the flights?

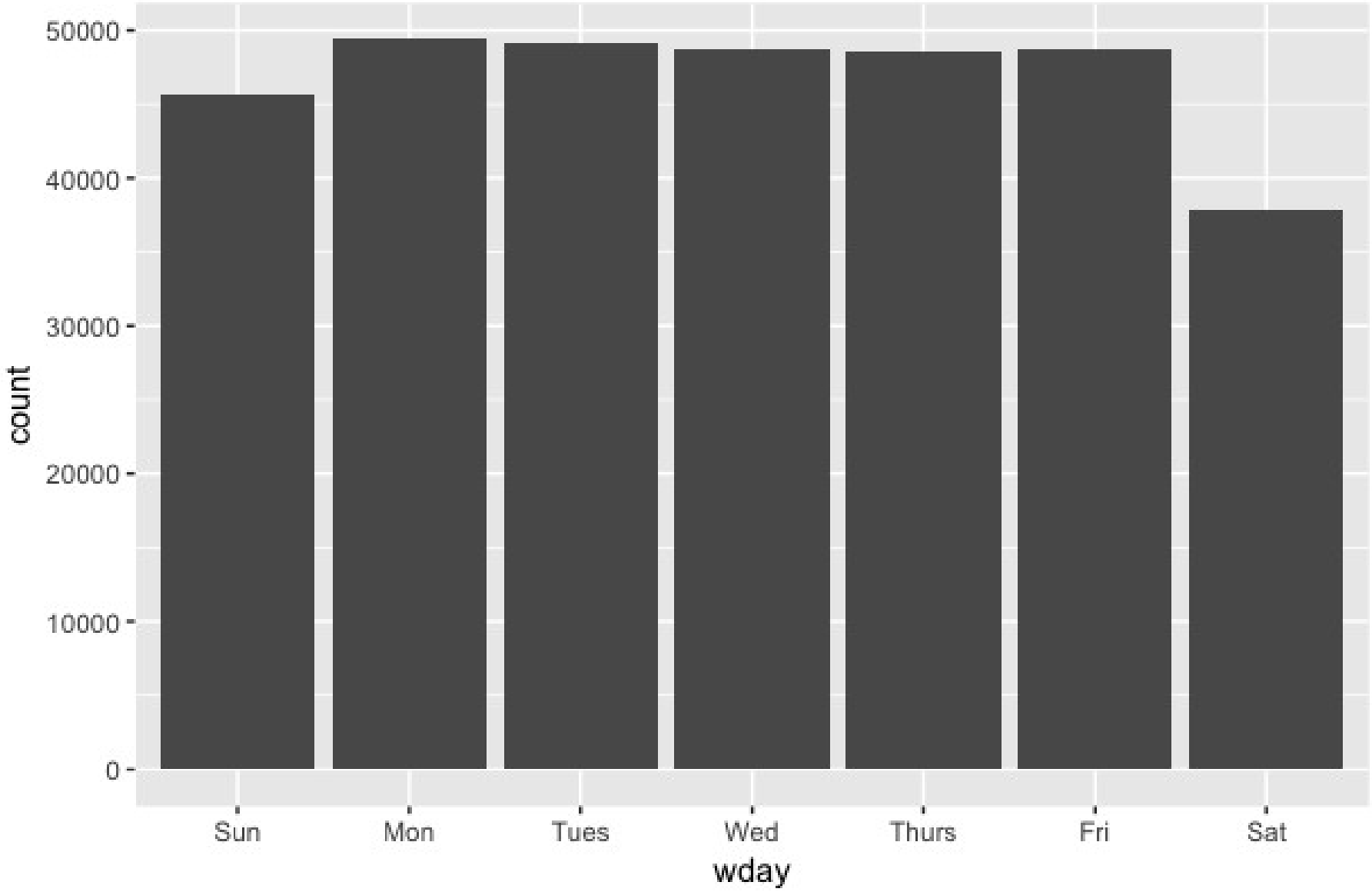
```
flights_dt %>%
  mutate(wday = wday(dep_time, label = TRUE)) %>%
  ggplot(aes(x = wday)) + geom_bar()
```

Aggregate by day of the week using wday()



More flights depart during the week days than on the weekend.

Average Departure Comparison

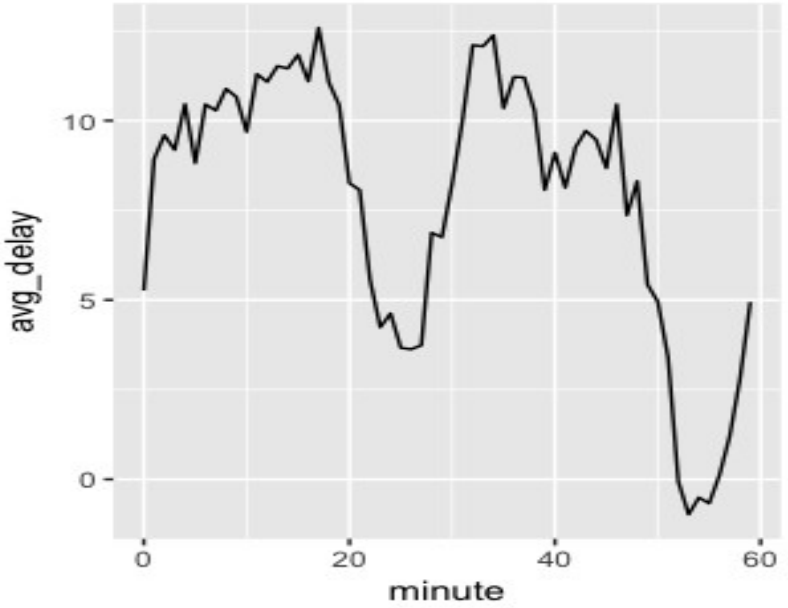


Quick Analysis!

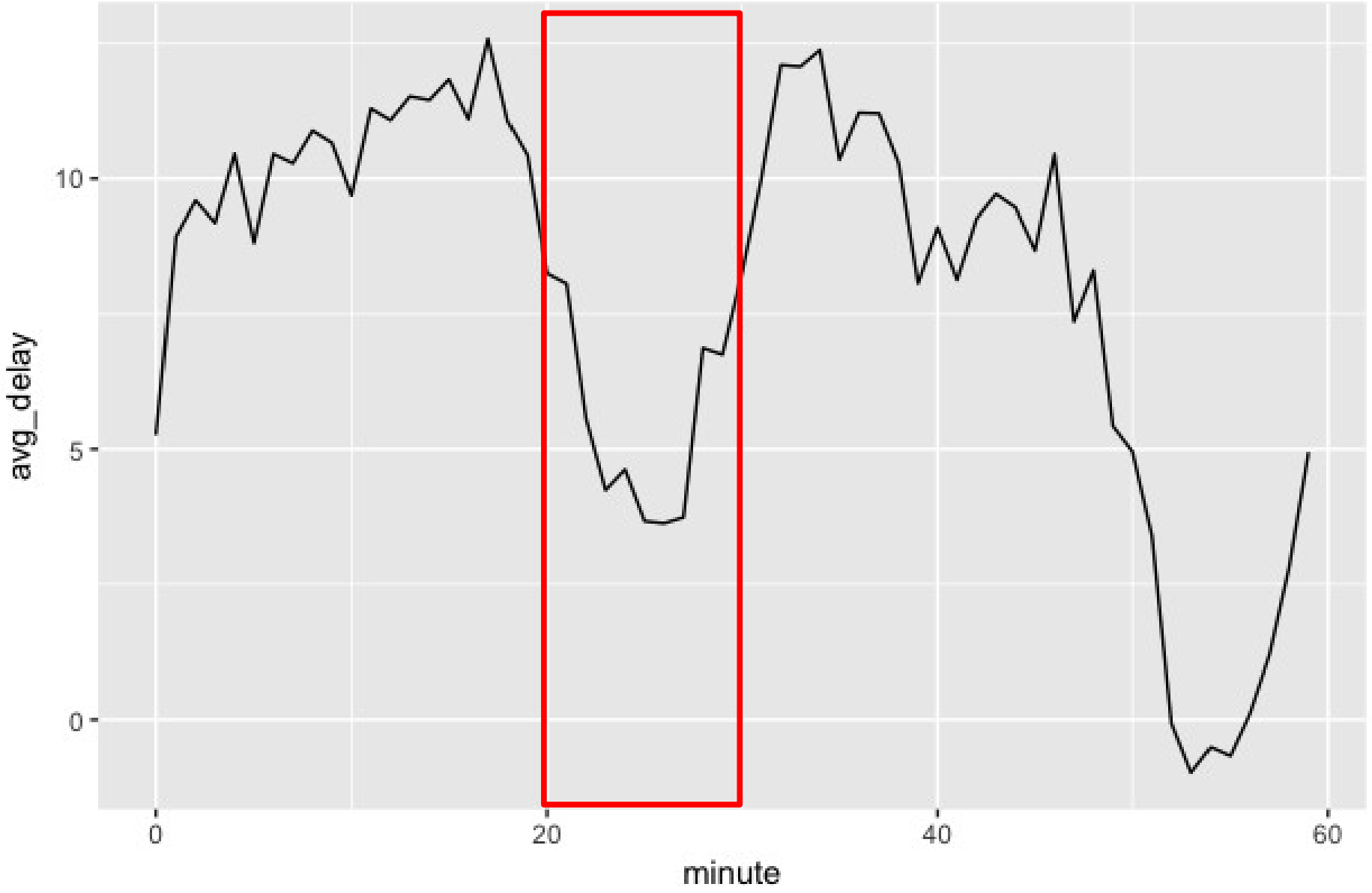
What is the average departure delay by minute, each hour?

```
flights_dt %>%
  mutate(minute = minute(dep_time)) %>% group_by(minute) %>%
  summarise(
    avg_delay = mean(arr_delay, na.rm = TRUE),
    n = n()) %>% ggplot(aes(minute, avg_delay)) + geom_line()
```

Flights leaving around minutes 20-30 And 50-60, have much lower delays than the rest



Trend: Flights leaving in minutes 20-30 and 50-60 have lower delays than the rest of the hour.



Code to Add the Vertical Lines

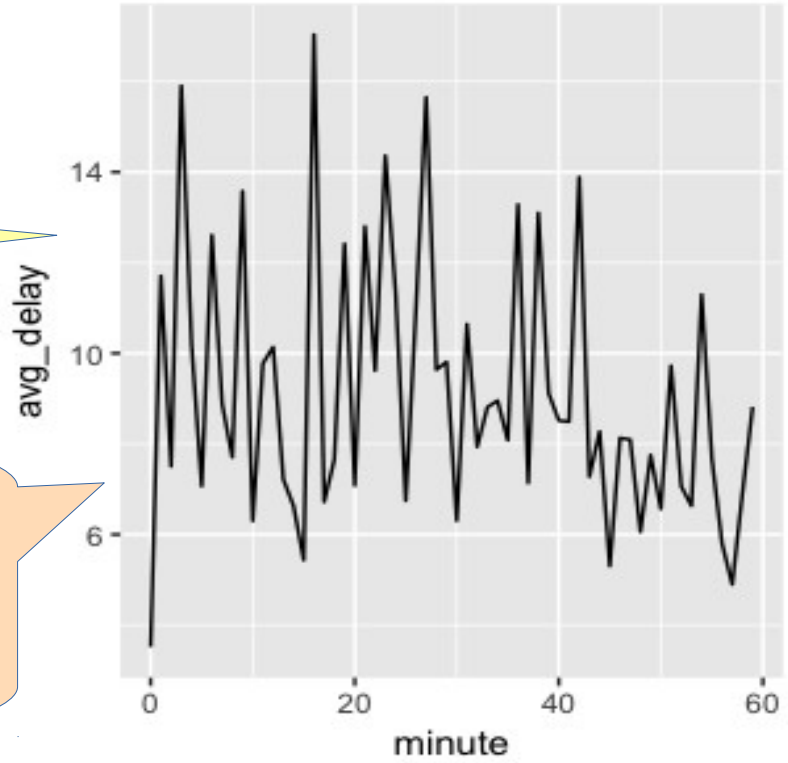
```
# The average departure time delay by minute within the
hour


    flights_dt %>%
    mutate(minute = minute(dep_time)) %>%
    group_by(minute) %>%
    summarise(
       avg_delay = mean(arr_delay, na.rm = TRUE),
       n = n()) %>%
    ggplot(aes(minute, avg_delay)) + geom_line() +
    geom_vline(xintercept = 20)
  + geom_vline(xintercept = 30)
```

Quick Analysis!

Is there a (similar) pattern according to scheduled departure times?

```
sched_dep <- flights_dt %>%  mutate(minute = minute(sched_dep_time)) %>%
group_by(minute) %>%
  summarise( avg_delay = mean(arr_delay, na.rm = TRUE), n = n())
ggplot(sched_dep, aes(minute, avg_delay)) + geom_line()
```

No (real) pattern exists.

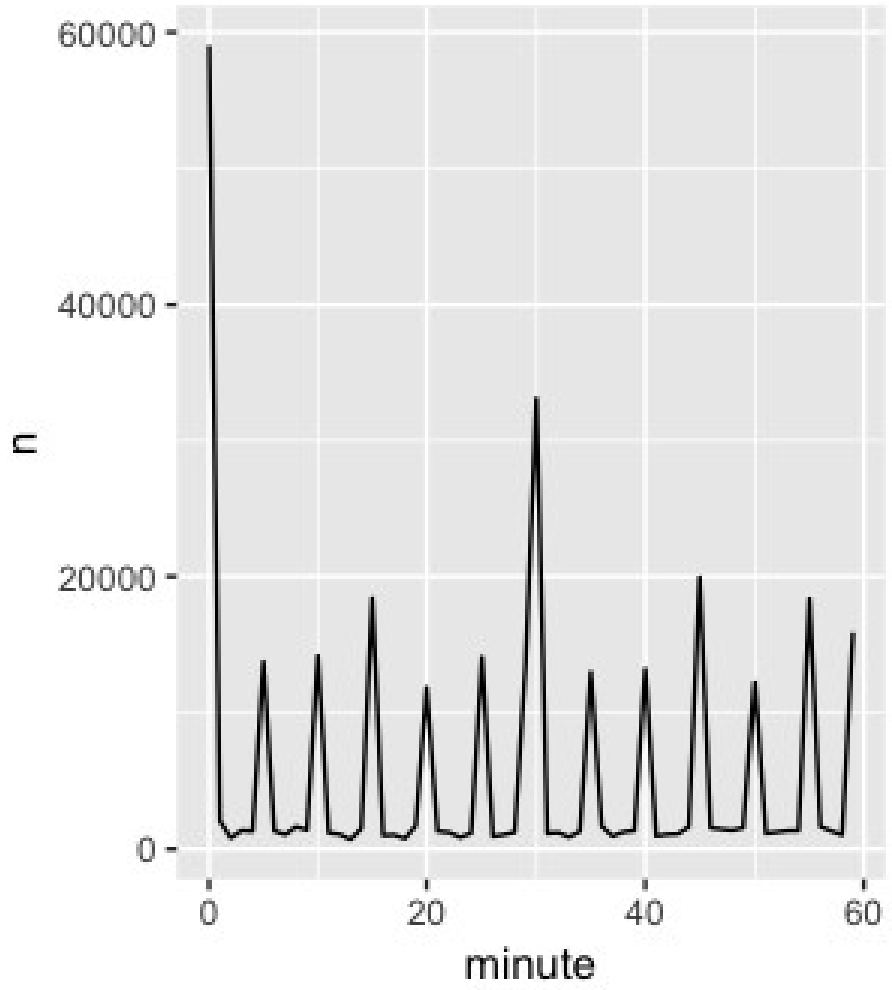


No pattern with the actual departure times?
Flights depart at times of convenience.
This is a bias that creeps into the
data since it is collected by people.

Quick Analysis!

What do *human-selected* departure times look like, by the hour?

```
ggplot(sched_dep, aes(minute, n)) + geom_line()
```



Predictable!