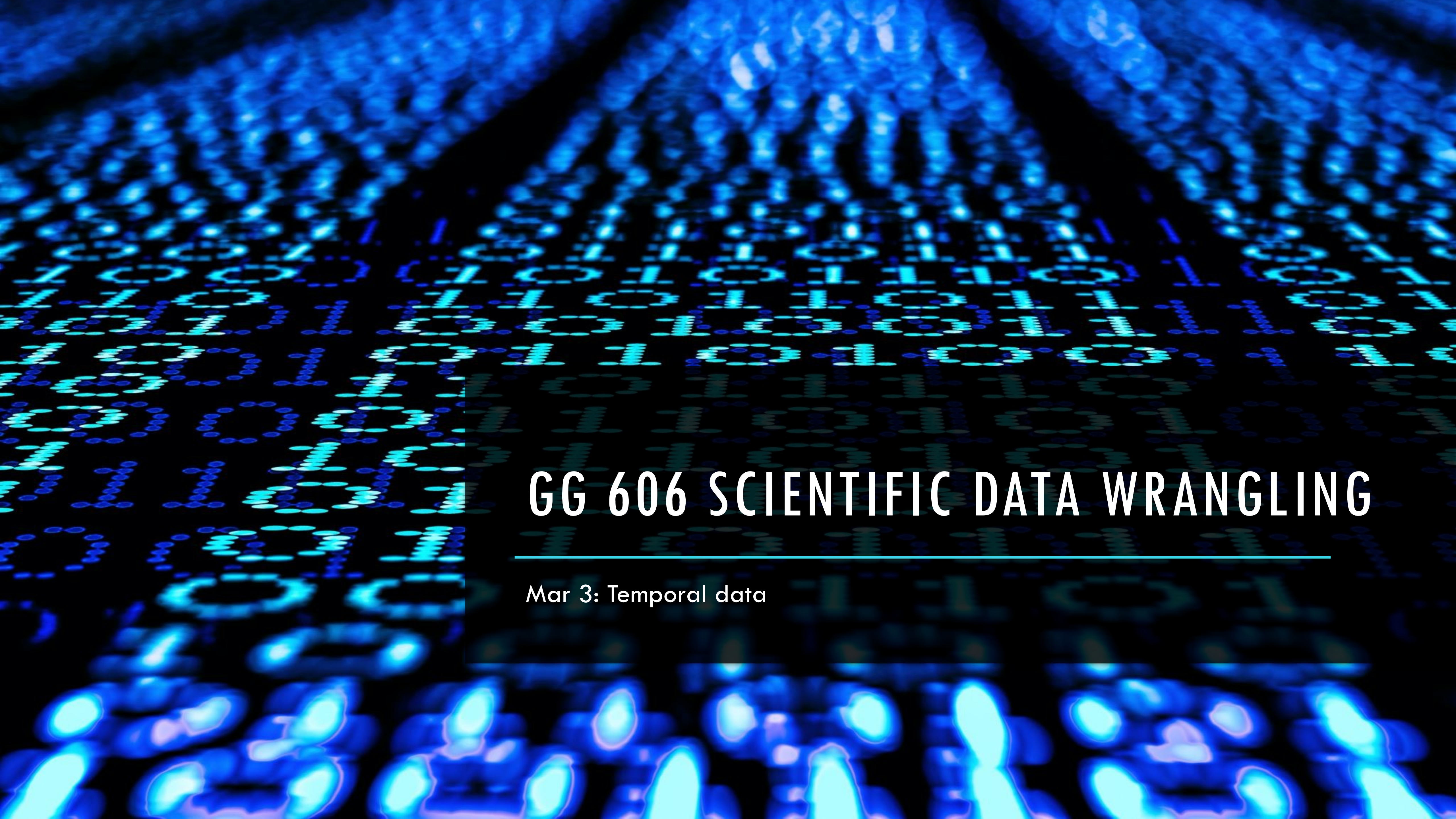


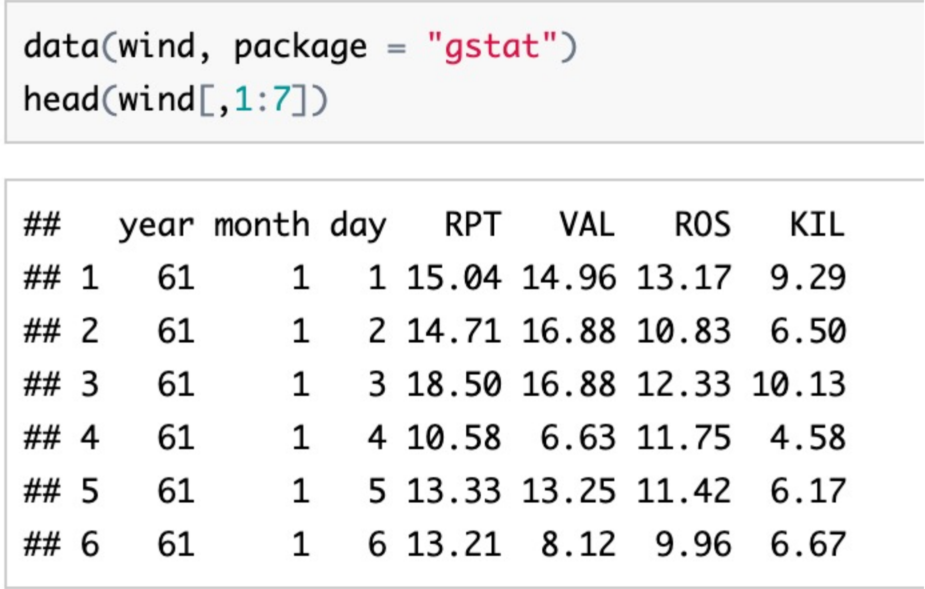
GG 606 SCIENTIFIC DATA WRANGLING

Mar 3: Temporal data

TEMPORAL DATA WRANGLING

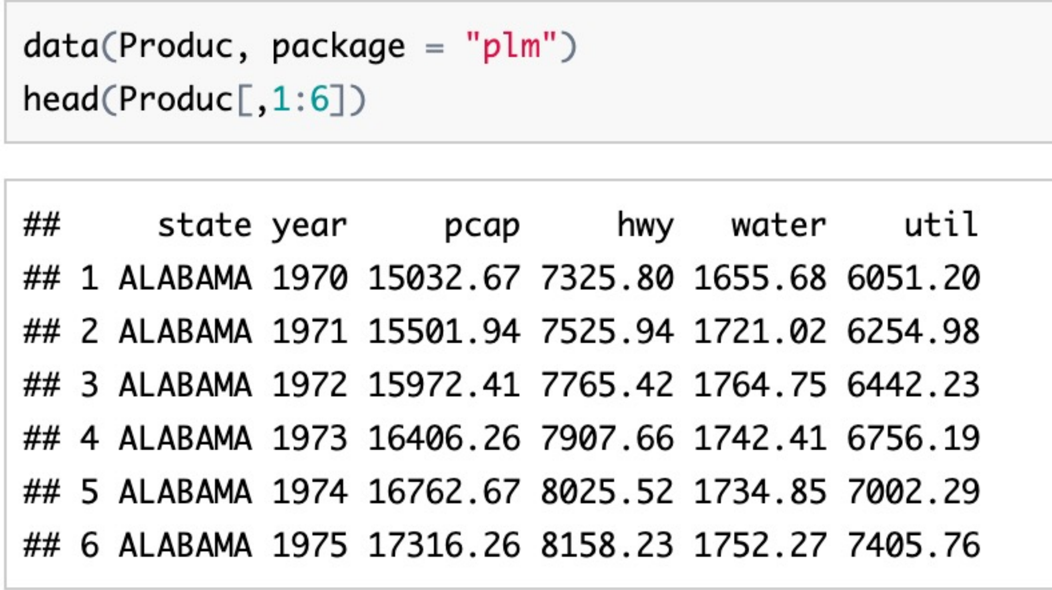
- Data frequently has a temporal component in addition to a spatial
- dates / times are frequently challenging to work with and need careful consideration when processing
    - if you can, ignore date/time and use integer / arbitrary time units
    - often you cant…

DATE/TIME DATA IN R??



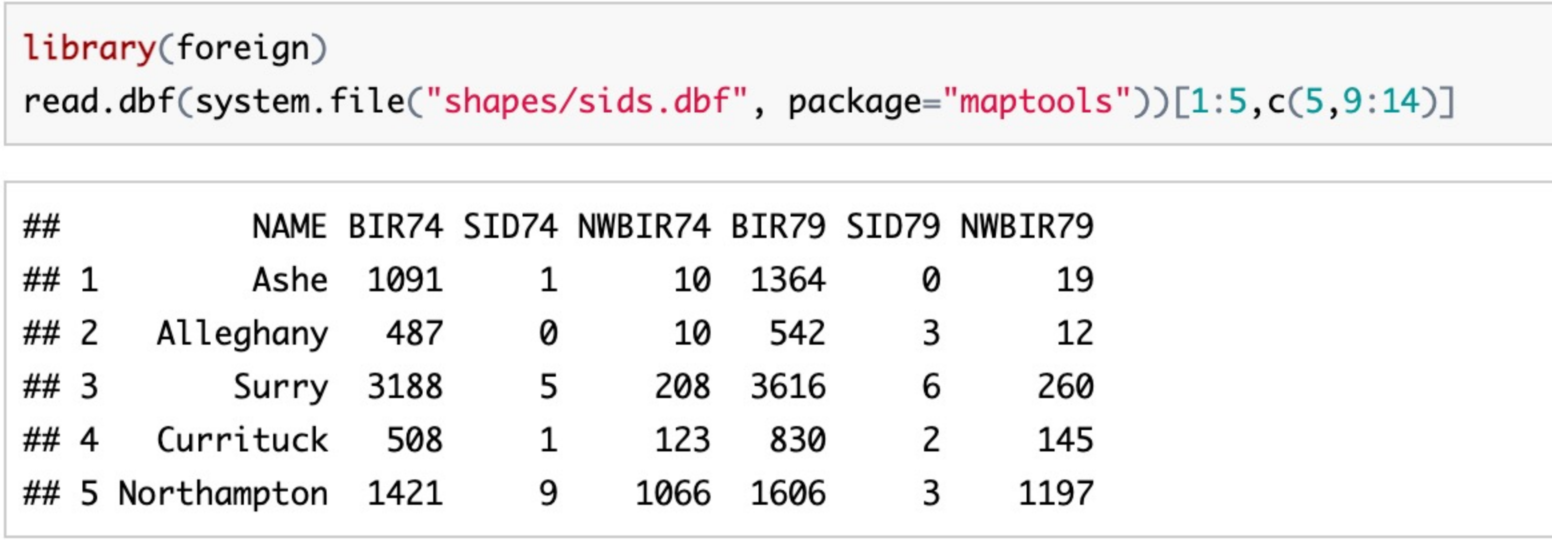
```r
data(wind, package = "gstat")
head(wind[,1:7])
```

```
##   year month day   RPT   VAL   ROS   KIL
## 1   61     1   1 15.04 14.96 13.17  9.29
## 2   61     1   2 14.71 16.88 10.83  6.50
## 3   61     1   3 18.50 16.88 12.33 10.13
## 4   61     1   4 10.58  6.63 11.75  4.58
## 5   61     1   5 13.33 13.25 11.42  6.17
## 6   61     1   6 13.21  8.12  9.96  6.67
```



```r
data(Produc, package = "plm")
head(Produc[,1:6])
```

```
##     state year     pcap     hwy   water    util
## 1 ALABAMA 1970 15032.67 7325.80 1655.68 6051.20
## 2 ALABAMA 1971 15501.94 7525.94 1721.02 6254.98
## 3 ALABAMA 1972 15972.41 7765.42 1764.75 6442.23
## 4 ALABAMA 1973 16406.26 7907.66 1742.41 6756.19
## 5 ALABAMA 1974 16762.67 8025.52 1734.85 7002.29
## 6 ALABAMA 1975 17316.26 8158.23 1752.27 7405.76
```



```r
library(foreign)
read.dbf(system.file("shapes/sids.dbf", package="maptools"))[1:5,c(5,9:14)]
```

```
##          NAME BIR74 SID74 NWBIR74 BIR79 SID79 NWBIR79
## 1        Ashe  1091     1      10  1364     0      19
## 2   Alleghany   487     0      10   542     3      12
## 3       Surry  3188     5     208  3616     6     260
## 4   Currituck   508     1     123   830     2     145
## 5 Northampton  1421     9    1066  1606     3    1197
```

lubridate basics

- There are three types of date/time data that refer to an instant in time:
  - A **date**. Tibbles print this as <date>.
  - A **time** within a day. Tibbles print this as <time>.
    - no need to handle this generally, can use a double of integer generally
  - A **date-time** is a date plus a time: it uniquely identifies an instant in time (typically to the nearest second). Tibbles print this as <dttm>.
    - base R uses a class called POSIXct

lubridate basics

- Sometimes we need to create dates based on the current execution time in r
- usually however we are reading in date and/or date-time data as character data in data files
- if we are connected to a database or have a database file with actual data types (e.g., dbf files) we can read directly into date type classes in r
- date data type is actually a `double`

```
> Sys.Date()
[1] "2022-03-02"
> class(Sys.Date())
[1] "Date"
> library(lubridate)
> today()
[1] "2022-03-02"
> class(today())
[1] "Date"
> now()
[1] "2022-03-02 12:36:06 PST"
> class(now())
[1] "POSIXct" "POSIXt"
> typeof(today())
[1] "double"
> typeof(Sys.Date())
[1] "double"
```

lubridate - input

- Dates from character data
- identify the order in which year, month, and day appear in your dates, then arrange “y”, “m”, and “d” in the same order
  - gives you the name of the lubridate function that will parse your date

lubridate - dates

```
ymd("2017-01-31")
#> [1] "2017-01-31"

mdy("January 31st, 2017")
#> [1] "2017-01-31"
dmy("31-Jan-2017")
#> [1] "2017-01-31"
```

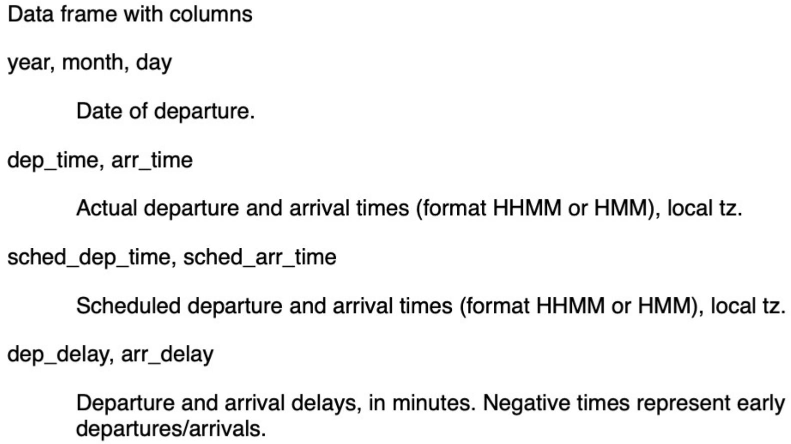
lubridate – dates + times

```
ymd_hms("2017-01-31 20:11:59")
#> [1] "2017-01-31 20:11:59 UTC"
mdy_hm("01/31/2017 08:01")
#> [1] "2017-01-31 08:01:00 UTC"
```

base r

```
as.Date("2017-01-31", "%Y-%m-%d")
[1] "2017-01-31"
> as.Date("January 31, 2017", "%B %d, %Y")
[1] "2017-01-31"
as.Date("31-Jan-2017", "%d-%b-%Y")
[1] "2017-01-31"
```

`lubridate` - input


Data frame with columns

year, month, day

Date of departure.

dep_time, arr_time

Actual departure and arrival times (format HHMM or HMM), local tz.

sched_dep_time, sched_arr_time

Scheduled departure and arrival times (format HHMM or HMM), local tz.

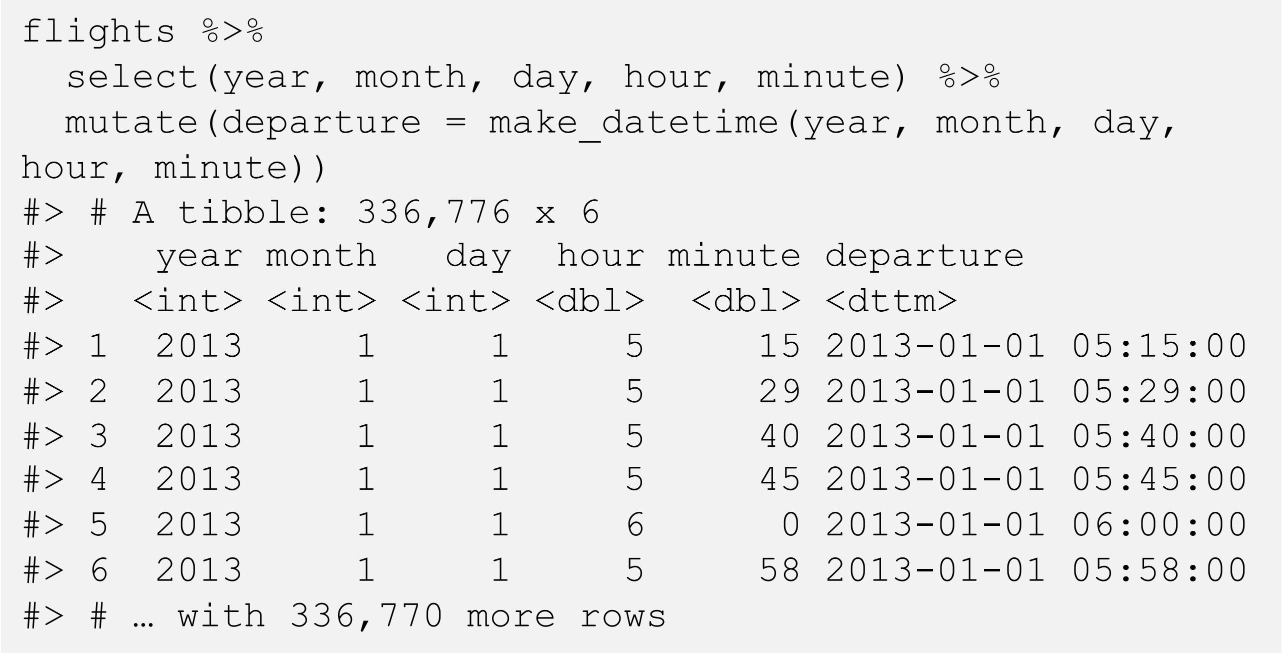
dep_delay, arr_delay

Departure and arrival delays, in minutes. Negative times represent early departures/arrivals.

- Equally common – because dates are such as pain – is that date data are split across columns, making it easy to query and filter using integer data types

- to filter by month or by year we can use standard filtering techniques and not have to use date classes at all

- to explore arrival and departure times we must create date-time data using `make_datetime`
    - assemble from existing columns



```
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
#> # … with 336,770 more rows
```

lubridate – extracting parts

- pull out individual parts of the date with the accessor functions
  - year(), month()
  - mday() (day of the month)
  - yday() (day of the year)
  - wday() (day of the week)
  - hour(), minute(), and second()

```
datetime <- ymd_hms("2016-07-08 12:34:56")

year(datetime)
#> [1] 2016
month(datetime)
#> [1] 7
month(datetime, label = TRUE)
#> [1] Jul
mday(datetime)
#> [1] 8
yday(datetime)
#> [1] 190
wday(datetime)
#> [1] 6
wday(datetime, label = TRUE, abbr = FALSE)
#> [1] Friday
```

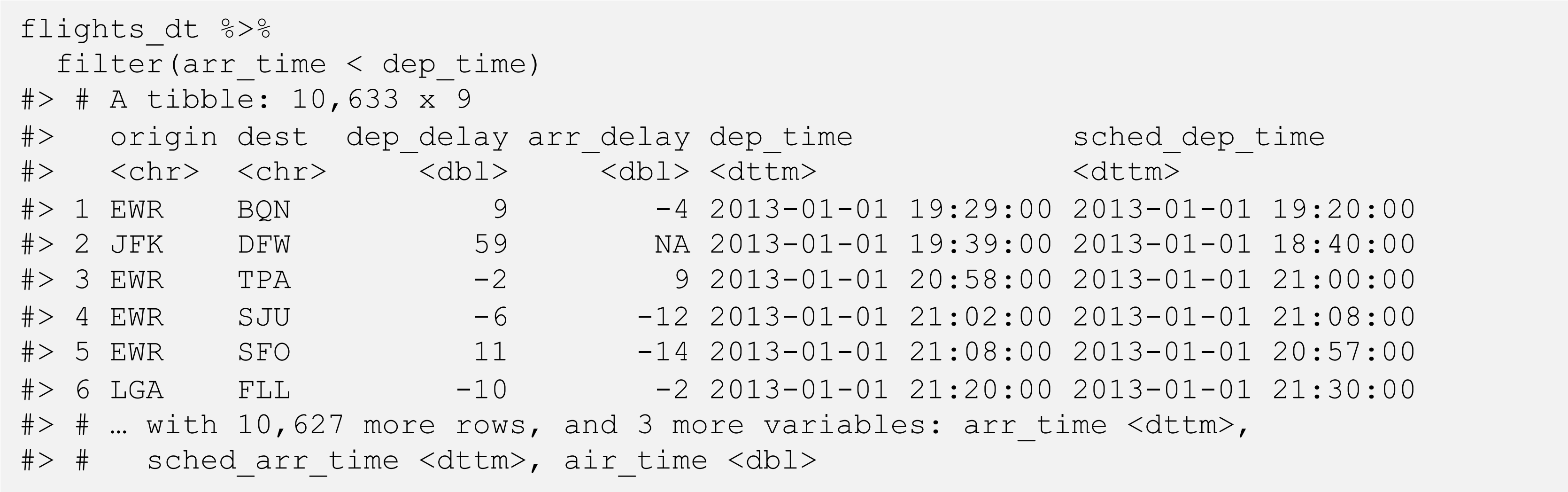
`lubridate` – manipulating date data

- Often you need to do basic arithmetic with date data:
  - e.g., calculate the duration between a start date end date or start time and end time
- difftime class object records a time span of seconds, minutes, hours, days, or weeks.
- `as.duration` provides a response always in seconds
- constructors such as ddays and dyears can be used for arithmetic operations

```
> Sys.Date() - 4
[1] "2022-02-26"
> class(Sys.Date() - 4)
[1] "Date"
> Sys.Date() - (Sys.Date() - 4)
Time difference of 4 days
> class(Sys.Date() - (Sys.Date() - 4))
[1] "difftime”
> as.duration(Sys.Date() - (Sys.Date() - 4))
[1] "345600s (~4 days)"
```

```
tomorrow <- today() + ddays(1)
last_year <- today() - dyears(1)
```

lubridate – manipulating date data

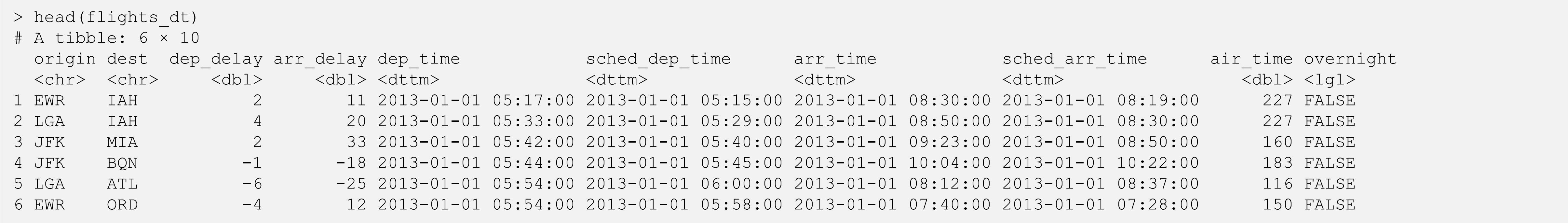
- Some planes appear to have arrived at their destination *before* they departed from New York City.
- 

```
flights_dt %>%
  filter(arr_time < dep_time)
#> # A tibble: 10,633 x 9
#>   origin dest  dep_delay arr_delay dep_time            sched_dep_time
#>   <chr>  <chr>     <dbl>     <dbl> <dttm>              <dttm>
#> 1 EWR    BQN           9        -4 2013-01-01 19:29:00 2013-01-01 19:20:00
#> 2 JFK    DFW          59        NA 2013-01-01 19:39:00 2013-01-01 18:40:00
#> 3 EWR    TPA          -2         9 2013-01-01 20:58:00 2013-01-01 21:00:00
#> 4 EWR    SJU          -6       -12 2013-01-01 21:02:00 2013-01-01 21:08:00
#> 5 EWR    SFO          11       -14 2013-01-01 21:08:00 2013-01-01 20:57:00
#> 6 LGA    FLL         -10        -2 2013-01-01 21:20:00 2013-01-01 21:30:00
#> # … with 10,627 more rows, and 3 more variables: arr_time <dttm>,
#> #   sched_arr_time <dttm>, air_time <dbl>
```

lubridate – manipulating date data

- We can fix this by adding days(1) to the arrival time of each overnight flight

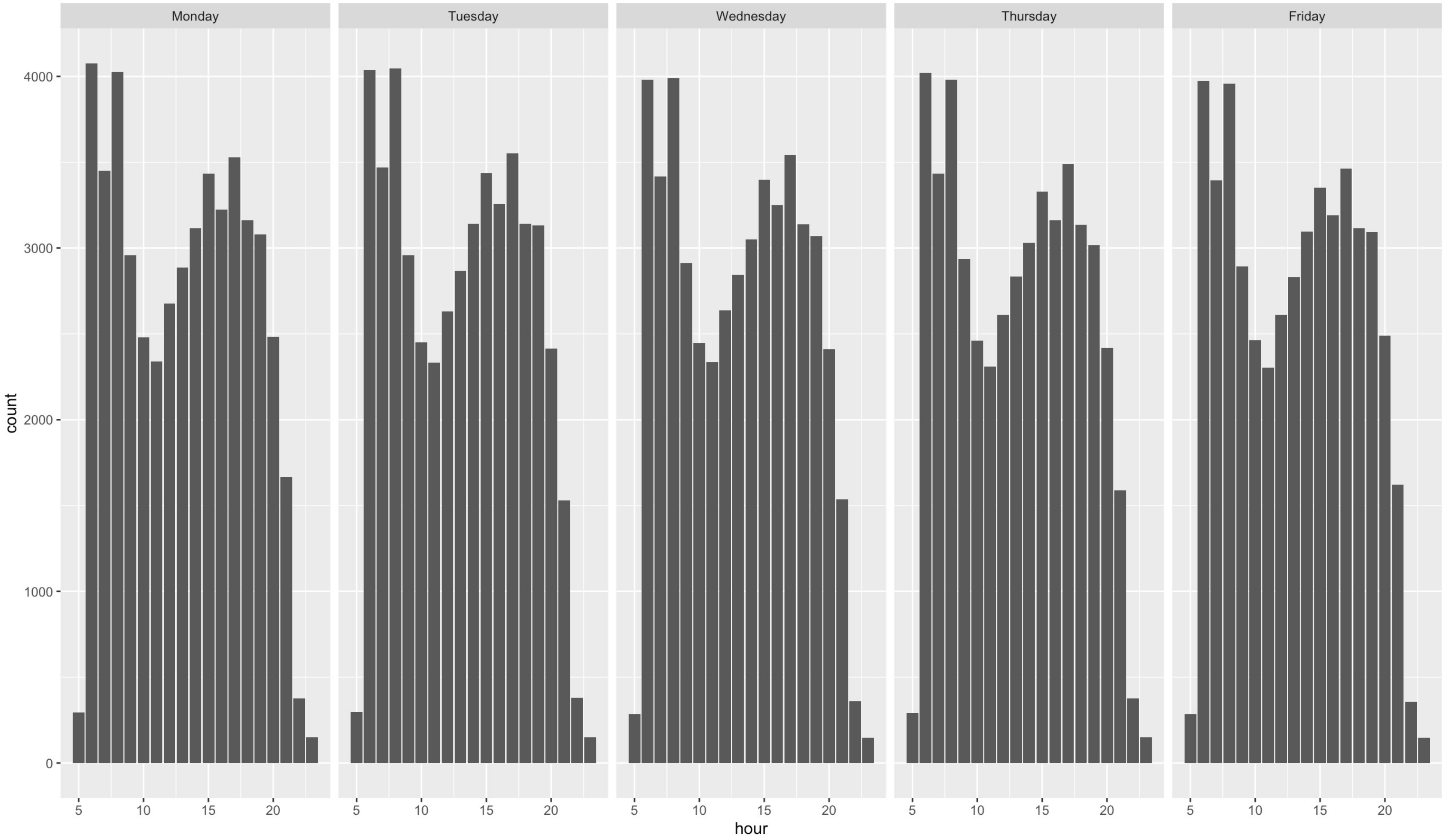
```
flights_dt <- flights_dt %>%
  mutate(
    overnight = arr_time < dep_time,
    arr_time = arr_time + days(overnight * 1),
    sched_arr_time = sched_arr_time + days(overnight * 1)
  )
```



```
> head(flights_dt)
# A tibble: 6 × 10
  origin dest  dep_delay arr_delay dep_time            sched_dep_time      arr_time            sched_arr_time      air_time overnight
  <chr>  <chr>     <dbl>     <dbl> <dttm>              <dttm>              <dttm>              <dttm>                 <dbl> <lgl>
1 EWR    IAH           2        11 2013-01-01 05:17:00 2013-01-01 05:15:00 2013-01-01 08:30:00 2013-01-01 08:19:00      227 FALSE
2 LGA    IAH           4        20 2013-01-01 05:33:00 2013-01-01 05:29:00 2013-01-01 08:50:00 2013-01-01 08:30:00      227 FALSE
3 JFK    MIA           2        33 2013-01-01 05:42:00 2013-01-01 05:40:00 2013-01-01 09:23:00 2013-01-01 08:50:00      160 FALSE
4 JFK    BQN          -1       -18 2013-01-01 05:44:00 2013-01-01 05:45:00 2013-01-01 10:04:00 2013-01-01 10:22:00      183 FALSE
5 LGA    ATL          -6       -25 2013-01-01 05:54:00 2013-01-01 06:00:00 2013-01-01 08:12:00 2013-01-01 08:37:00      116 FALSE
6 EWR    ORD          -4        12 2013-01-01 05:54:00 2013-01-01 05:58:00 2013-01-01 07:40:00 2013-01-01 07:28:00      150 FALSE
```

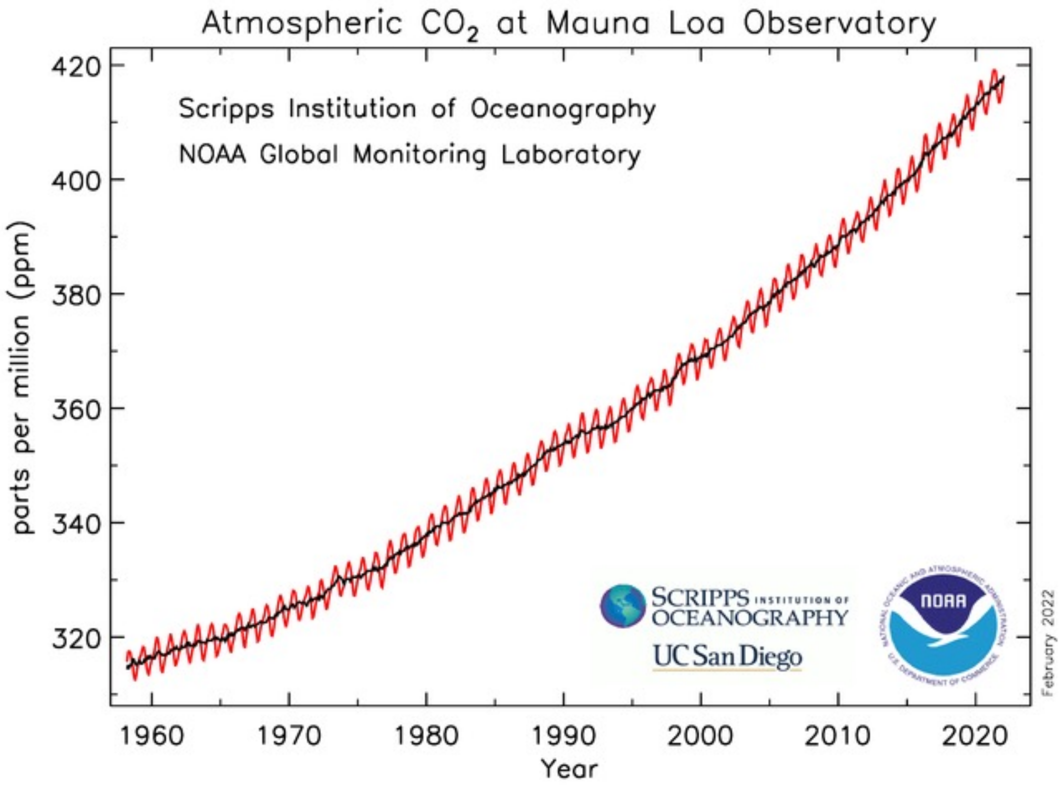
LUBRIDATE – MANIPULATING DATE DATA

```
# days of the week:
dow = c("Sunday", "Monday", "Tuesday",
"Wednesday", "Thursday", "Friday", "Saturday")


flights_dow <- flights_dt %>%
  mutate(hour = lubridate::hour(sched_dep_time))
%>%
  mutate(day = factor(weekdays(sched_dep_time),
levels = dow))


flights_dow %>%
  filter(!is.na(hour) & !day %in% c("Saturday",
"Sunday")) %>%
  ggplot(aes(hour)) +
  geom_bar() +
  facet_wrap(~day, nrow = 1)
```



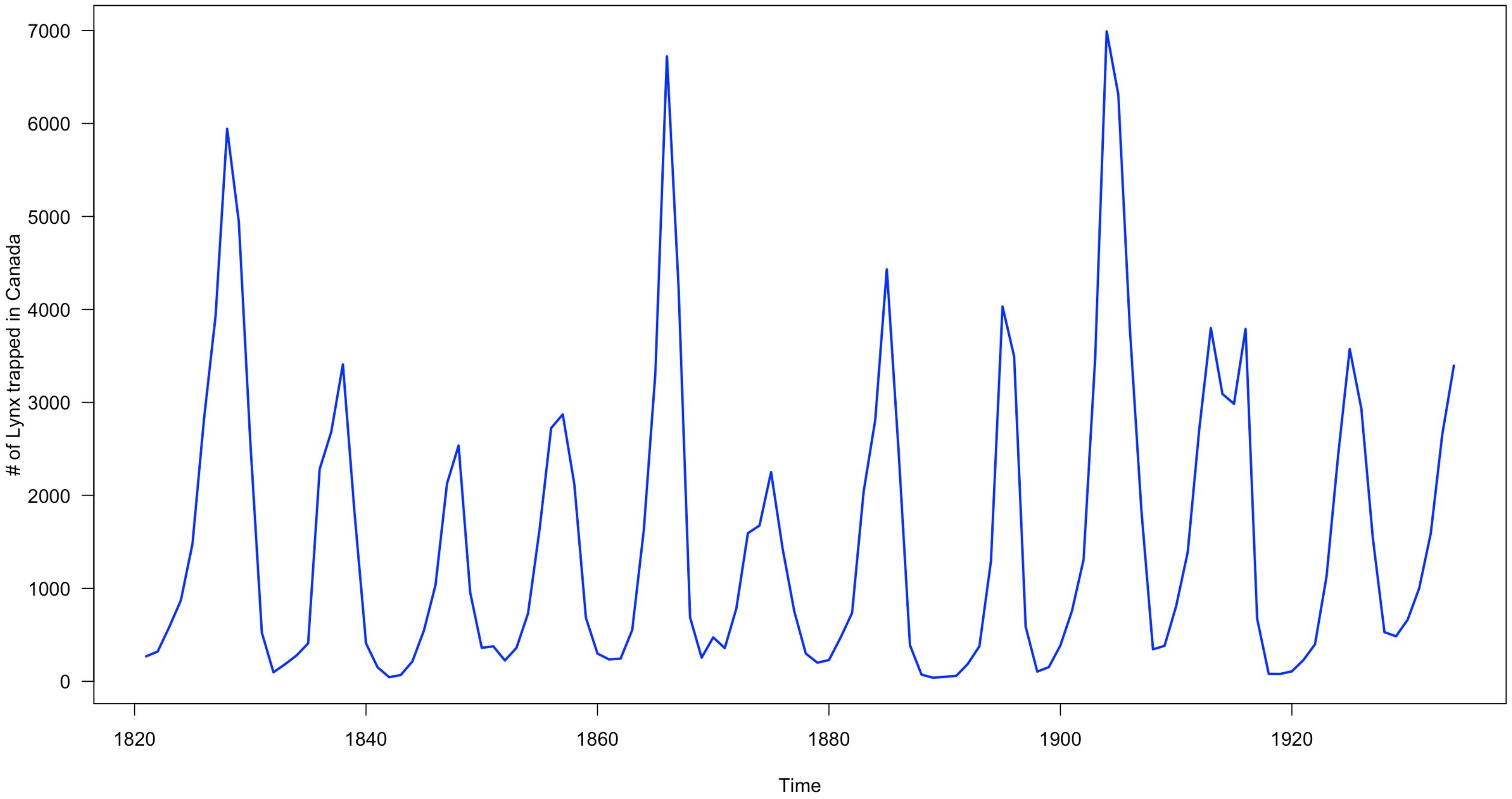
ANALYSIS OF TEMPORAL DATA

- Time series analysis – lots of options available in
- a time series is a set of observations taken sequentially in time
- We often want to plot time series
  - with time increasing almost always on the x-axis
- For time series we typically want arbitrary time units
  - i.e., data stored as integers



ANALYSIS OF TEMPORAL DATA

```
data(lynx, package = "datasets")
plot.ts(lynx, ylab = "# of Lynx trapped in Canada", las = 1, col = "blue", lwd = 2)
```



ANALYSIS OF TEMPORAL DATA

- Common forms of time series analysis available in R
  - time series modelling
  - time series decomposition
    - trend, seasonal, error components
  - Box Jenkins modelling
    - ARMA – autoregressive moving average (forecast package)
    - ARIMA – autoregressive integrated moving average (forecast package)
  - Random walks, correlated random walks, Brownian movement, Levy models
  - Change point analysis
    - significance shifts in mean, variance, etc.

https://cran.uni-muenster.de/web/views/TimeSeries.html

`xts` AND `zoo` PACKAGES

- For time series analysis while maintining date/time data classes

Introducing xts and zoo objects

MANIPULATING TIME SERIES DATA WITH XTS AND ZOO IN R

spacetime

SUMMARY

- There are many different tools available for handling temporal data in r
  - ts objects
  - date classes and lubridate
  - zoo and xts packages
- In general, select the simplest packages/classes you can for your needs and that integrate with your overall workflow
- Build on your knowledge of spatial and temporal classes to explore space/time data

GG 606 SCIENTIFIC DATA WRANGLING

1. Jan 27: databases