Dates and Times I

The next few questions reference the moment that the ball fully dropped in Times Square in New York City at the first moment of the new year in 2000. The corresponding UNIX time was 946684800. Please form a vector to captures this point in time in several different ways. Remember you may need to look at function help files to learn more about how the functions taught in class work.

1. Using baseR and the Unix timepoint.

2. Using baseR and a string that expresses the timepoint in a format similar to `"2025-10-10 13:00:00"`.

3. Using baseR and a string that expresses the timepoint in a format similar to `"Oct 10, 2025 1:00 pm"`.

4. Using lubridate and a string that expresses the timepoint in a format similar to `"2025-10-10 13:00:00"`.

5. Using lubridate and a string that expresses the timepoint in a format similar to `"Oct 10, 2025 1:00 pm"`.

6. What is the type of the objects created by the different commands used above?

Each of the following questions should be answered with an R command.

7. What day of the week were you born on?

8. The first day of classes this semester fell on which week of the current year?

9. In which year will Unix time cross the 2 trillion second mark?

10. What is the precise date and time at which that milestone will occur?

Dates and Times II

Using the notions of durations, periods, and timezones from `lubridate`, provide the code and output to answer the following questions.

11. How old are you? Answer as a duration.

12. How old will you be in one month? Answer both using durations and periods.

13. What time is it now? Answer using the `now()` function in lubridate and answer in HH:MM:SS and provide the timezone.

14. What is the current time in UTC (that is to say, in London)?

15. Now imagine you call your friend right now in Delhi, India and ask them to do the same on their phone. What will they say? (in HH:MM:SS with timezone).

These exercises will be easiest done iteratively in R. You can copy your answers back here or produce your problem set document using Quarto directly. The data set at hand is called `flights` inside the `nycflights` package.

```r
library(dplyr)
library(lubridate)    # this is also in library(tidyverse)
library(nycflights13) # install this if you don't have it
flights               # take a glimpse at the data frame
```

```
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# ℹ 336,766 more rows
# ℹ 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

16. Start by reading the documention for the data frame. What is the unit of observation? What time span and geography is covered by this data? Which time zones are any time columns recorded in?

17. Using `year`, `month`, `day`, and the integer clock times `sched_dep_time` and `dep_time` (HHMM)[1], add two more variables to your data frame:

i. `sched_dep_ts`: the scheduled departure timestamp.
ii. `dep_ts`: the actual wheels-off timestamp

Both must be POSIXct in the "America/New_York" time zone.

18. Calculate the actual wheels-off timestamp in two additional ways and store them as additional variables in your data frame:

i. `dep_ts_dur`: add to `sched_dep_ts` the recorded time span of the departure delay as a *duration.*
ii. `dep_ts_per`: add to `sched_dep_ts` the recorded time span of the departure delay as a *period.*

In general do the different approaches align? How about if you look at the precise edge of a switch to/from Daylight Savings Time?

[1] 1

19. *Challenge*: Construct scheduled and realized arrival timestamps using only columns in flights. Build `sched_arr_ts` from `year`, `month`, `day`, and `sched_arr_time` (HHMM) in “America/New_York”.

For red-eyes that arrive after midnight local time, `sched_arr_time` may be smaller than `sched_dep_time`. Correct the date by adding a day to `sched_arr_ts` when needed.

Do the analogous construction for actual `arr_ts` from `arr_time`.

Then compute:

- `sched_block = as.duration(sched_arr_ts - sched_dep_ts)`
- `actual_block = as.duration(arr_ts - dep_ts)`

Report the median difference `actual_block` - `sched_block` in minutes.

Spatial data 1

There are multiple R packages that can be used to load geographic data stored in the *simple features* format. `rnaturalearth` stores data at the global scale, particular countries. `tigris` stores data from the US Census at a much finer grain of detail; it has counties, census blocks, zip codes, etc, and is updated yearly.

20. Using the `rnaturalearth` download map data of a country of your choosing (not US or Mexico) in `sf` format. How many simple features are in the data set? What datum was used to record it (see "Geodetic CRS")?

21. Using `sf` and `ggplot2` packages, make a map of your country[2].

22. Using the `tigris` package, download a map of California that shows county outlines as of 2024. How many simple features are in the data set? What datum was used to record it?

```r
library(tigris)
ca_counties <- counties(state = "CA", cb = TRUE, year = 2024)
```

23. Make a map of the counties of California two ways: using the default datum and also by using the Albers Equal Area projection[3]. How do they compare?

[2]If you are unsatisfied with the resolution of your map, you can tinker with the `scale` argument of the `ne_countries()` function.

[3]Each projection has a four digit code associated with it. Albers Equal Area is 3310. That can be passed as a string to the `crs` argument of the `st_transform()` function in `sf`.

24. UC Berkeley is host to the Northern California Earthquake Data Center (http://ncedc.org). Data from all of the earthquakes in September and October of 2025 were downloaded as a csv file and can be read in with the following.

```r
quakes <- read_csv("https://tinyurl.com/ca-quakes-25")
```

Add the locations of these earthquakes to both of your CA maps using `geom_point()`.

25. As you will have noticed, the earthquakes do not appear on the Albers projection because the points in the polygons of the counties were projected while the earthquake locations remain in WGS84. Remake your Albers map after converting the earthquake data frame to an sf object in WGS84, then transform it to the Albers projection (id 3310)[4].

```r
quakes_wgs84 <- st_as_sf(quakes, coords = c("Longitude","Latitude"), crs = 4326,
remove = FALSE)
quakes_albers <- st_transform(quakes_wgs84, 3310)
```

[4]Hint: since `quakes_albers` is an sf object instead of a tibble, `geom_point()` will no longer work. You can add it in a second `geom_sf()` layer (the first will be for the basemap).