Problem set 4

2025-10-05

In the next problem set, we plan to explore the relationship between COVID-19 death rates and vaccination rates across US states by visually examining their correlation. This analysis will involve gathering COVID-19 related data from the CDC's API and then extensively processing it to merge the various datasets. Since the population sizes of states vary significantly, we will focus on comparing rates rather than absolute numbers. To facilitate this, we will also source population data from the US Census to accurately calculate these rates.

In this problem set we will learn how to extract and wrangle data from the data US Census and CDC APIs.

1. Get an API key from the US Census at https://api.census.gov/data/key_signup.html. You can't share this public key. But your code has to run on a TFs computer. Assume the TF will have a file in their working directory named `census-key.R` with the following one line of code:

```
census_key <- "A_CENSUS_KEY_THAT_WORKS"
```

Write a first line of code for your problem set that defines `census_key` by running the code in the file `census-key.R`.

```r
source("census-key.R")
```

2. The US Census API User Guide provides details on how to leverage this valuable resource. We are interested in vintage population estimates for years 2021 and 2022. From the documentation we find that the *endpoint* is:

```r
url <- "https://api.census.gov/data/2021/pep/population"
```

Use the **httr2** package to construct the following GET request.

```
https://api.census.gov/data/2021/pep/population?get=POP_2020,POP_2021,NAME&for=state:*&key=Y
```

Create an object called `request` of class `httr2_request` with this URL as an endpoint. Hint: Print out `request` to check that the URL matches what we want.

```r
library(httr2)
request <- request(url) |>
  req_url_query(
    get = "POP_2020,POP_2021,NAME",
    `for` = "state:*",
    key = census_key
  )
```

3. Make a request to the US Census API using the `request` object. Save the response to and object named `response`. Check the response status of your request and make sure it was successful. You can learn about *status codes* here.

```r
response <- req_perform(request)
resp_status(response)
```

```
[1] 200
```

4. Use a function from the **httr2** package to determine the content type of your response.

```r
resp_content_type(response)
```

```
[1] "application/json"
```

5. Use just one line of code and one function to extract the data into a matrix. Hints: 1) Use the `resp_body_json` function. 2) The first row of the matrix will be the variable names and this OK as we will fix in the next exercise.

```r
population <- resp_body_json(response) |> do.call(rbind, args = _)
```

6. Examine the `population` matrix you just created. Notice that 1) it is not tidy, 2) the column types are not what we want, and 3) the first row is a header. Convert `population` to a tidy dataset. Remove the state ID column and change the name of the column with state names to `state_name`. Add a column with state abbreviations called `state`. Make sure you assign the abbreviations for DC and PR correctly. Hint: Use the **janitor** package to make the first row the header.

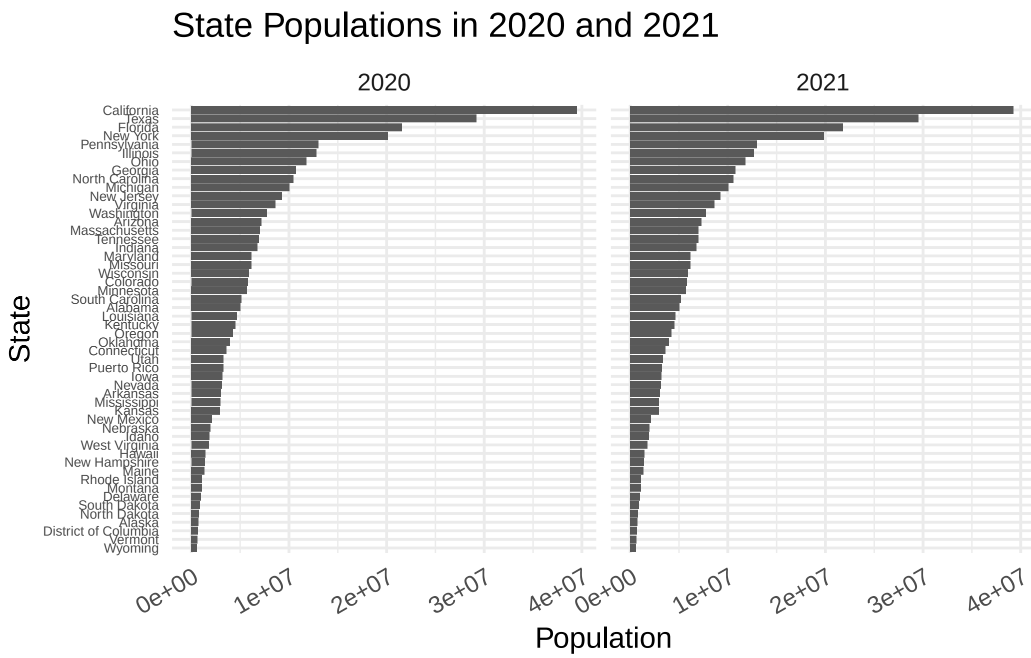
```r
library(tidyverse)
library(janitor)

population <- population |>
  as_tibble() |>              # convert to tibble
  row_to_names(row_number = 1) |> # turns entries in first row into column
   ↪ names
  select(-state) |>                   # remove state column
  rename(state_name = NAME) |>  # rename state column to state_name
  pivot_longer(
    cols = starts_with("POP_"), # select POP_ columns
    names_to = "year", # make new column "year" which includes POP_ entries
    values_to = "population" # corresponding population will be listed in
     ↪ column
  ) |>
  mutate(state_name = as.character(state_name),
    population = as.numeric(population),
    year = str_remove(year, "POP_"),        # remove POP_ from year
    state = state.abb[match(state_name, state.name)], # add state
     ↪ abbreviations using state.abb variable, specifying how to abbreviate
     ↪ PR and DC
    state = case_when(
    state_name == "District of Columbia" ~ "DC",
    state_name == "Puerto Rico" ~ "PR",
    TRUE ~ state)
    )
```

7. As a check, make a barplot of states' 2021 and 2022 populations. Show the state names in the y-axis ordered by population size. Hint: You will need to use `reorder` and use `facet_wrap`.

```r
library(ggplot2)
population |>
  filter(year %in% c("2020", "2021")) |>
  mutate(order_state = reorder(state_name, population)) |>
  ggplot(aes(x = population, y = order_state,  # reorder by population within
   ↪ year
  )) + geom_col() +
  labs(title = "State Populations in 2020 and 2021", x = "Population",
    y = "State") +
  facet_wrap(~ year) +  # each year in its own panel
  theme_minimal() +
```

```
  theme(axis.text.y = element_text(size=5),
        axis.text.x = element_text(angle = 30, hjust = 1))
```



8. The following URL:

```
url <- "https://github.com/datasciencelabs/2025/raw/refs/heads/main/data/regions.json"
```

points to a JSON file that lists the states in the 10 Public Health Service (PHS) defined by CDC. We want to add these regions to the `population` dataset. To facilitate this create a data frame called `regions` that has two columns `state_name`, `region`, `region_name`. One of the regions has a long name. Change it to something shorter.

```
library(jsonlite)
library(purrr)
regions <-fromJSON(url)  |> unnest(states) |>
  rename(state_name = states) |>
  mutate(region = as.integer(region), state_name = as.character(state_name),
   region_name = ifelse(region_name == "New York and New Jersey, Puerto Rico, Virgin Islands", "NY, NJ, PE, VI", region_name))  |>
  filter(state_name %in% c(state.name, "District of Columbia", "Puerto Rico"))  |> select(state_name, region, region_name)
```

9. Add a region and region name columns to the `population` data frame.

```
population <- population |> left_join(regions, by = "state_name")
```

10. From reading https://data.cdc.gov/ we learn the endpoint `https://data.cdc.gov/resource/pwn4-m3yp.json` provides state level data from SARS-COV2 cases. Use the **httr2** tools you have learned to download this into a data frame. Is all the data there? If not, comment on why.

```
api <- "https://data.cdc.gov/resource/pwn4-m3yp.json"
cases_raw <- request(api) |> req_perform() |>
↪ resp_body_json(simplifyDataFrame = TRUE)
```

We see exactly 1,000 rows. We should be seeing over 52×3 rows per state.

11. The reason you see exactly 1,000 rows is because CDC has a default limit. You can change this limit by adding `$limit=10000000000` to the request. Rewrite the previous request to ensure that you receive all the data. Then wrangle the resulting data frame to produce a data frame with columns `state`, `date` (should be the end date) and `cases`. Make sure the cases are numeric and the dates are in `Date` ISO-8601 format.

```
api <- "https://data.cdc.gov/resource/pwn4-m3yp.json"
cases_raw <- request(api) |>  req_url_query(`$limit` =10000000000) |>
↪ req_perform() |> resp_body_json(simplifyDataFrame = TRUE)

cases <- cases_raw |> mutate(state = state, date = as.Date(end_date), cases =
↪ as.numeric(new_cases)) |> select(state, date, cases) |> filter(state %in%
↪ c(state.abb, "DC", "PR"), !is.na(cases), !is.na(date))
head(cases)
```

```
  state       date cases
1    AZ 2023-02-22  3716
2    LA 2022-12-21  4041
3    GA 2023-02-22  5298
4    LA 2023-03-29  2203
5    LA 2023-02-01  5725
6    LA 2023-03-22  1961
```

12. For 2020 and 2021, make a time series plot of cases per 100,000 versus time for each state. Stratify the plot by region name. Make sure to label you graph appropriately.

```r
library(lubridate)
```

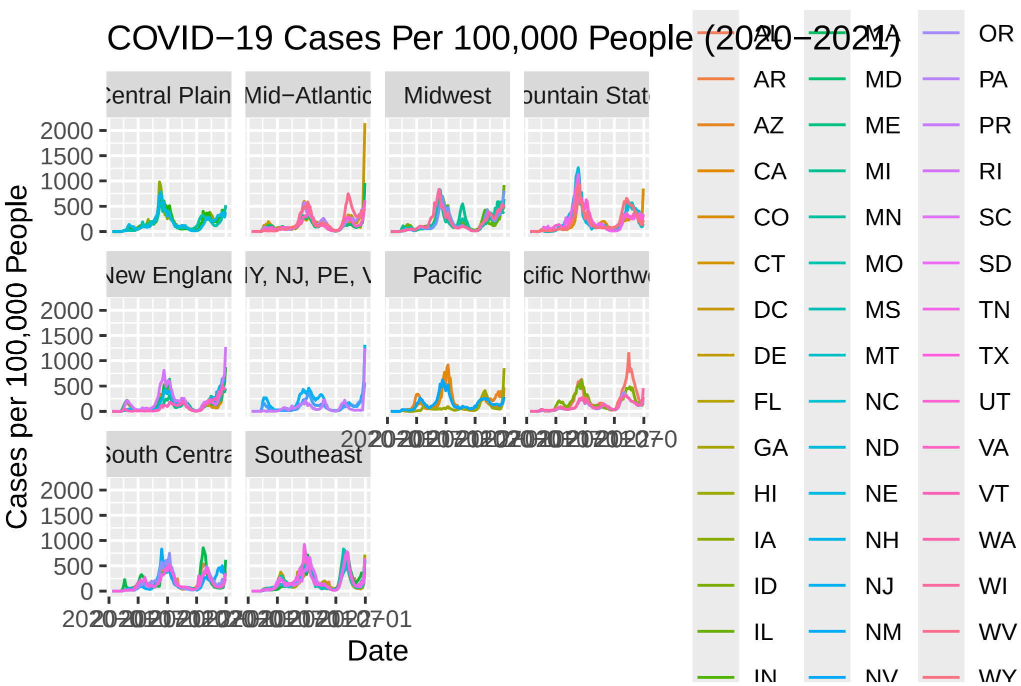
```

Attaching package: 'lubridate'

The following objects are masked from 'package:base':

    date, intersect, setdiff, union
```

```r
cases |> mutate(year = as.character(year(date))) |> # add a year column
  filter(year %in% c(2020, 2021)) |>
  left_join(population, by = c("state", "year")) |>
  filter(!is.na(region_name)) |> mutate(cases100k = (cases / population)
  ↪  *100000) |>
  ggplot(aes(x = date, y = cases100k, col = state)) + geom_line() +
  ↪  facet_wrap(~ region_name) +
  labs(title = "COVID-19 Cases Per 100,000 People (2020-2021)", x = "Date", y
  ↪  = "Cases per 100,000 People")
```



13. The dates in the `cases` dataset are stored as character strings. Use the **lubridate** package to properly parse the `date` column, then create a summary table showing the

total COVID-19 cases by month and year for 2020 and 2021. The table should have columns for year, month (as month name), and total cases across all states. Order by year and month. Use the **knitr** package and `kable()` function to display the results as a formatted table.

```
library(knitr)
cases |> filter(year(date) %in% c(2020, 2021)) |>
  group_by(year = year(date), month = month(date, label = TRUE, abbr =
  ↪ FALSE)) |>
  summarise(total_cases = sum(cases, na.rm = TRUE), .groups = "drop") |>
  arrange(year, month) |>
  kable(caption = "Total COVID-19 Cases by Month and Year (2020-2021)",
      col.names = c("Year", "Month", "Total Cases"))
```

```
Warning: 'xfun::attr()' is deprecated.
Use 'xfun::attr2()' instead.
See help("Deprecated")
Warning: 'xfun::attr()' is deprecated.
Use 'xfun::attr2()' instead.
See help("Deprecated")
```

Table 1: Total COVID-19 Cases by Month and Year (2020–2021)

Year	Month	Total Cases
2020	January	11
2020	February	68
2020	March	50335
2020	April	822648
2020	May	616691
2020	June	642552
2020	July	1977016
2020	August	1452393
2020	September	1401917
2020	October	1608932
2020	November	3887222
2020	December	6907540
2021	January	5649115
2021	February	2543964
2021	March	1928749
2021	April	1694189
2021	May	948953

Year	Month	Total Cases
2021	June	484817
2021	July	1120939
2021	August	3519407
2021	September	4960807
2021	October	2317854
2021	November	2289118
2021	December	5293391

14. The following URL provides additional COVID-19 data from the CDC in JSON format:

```
deaths_url <- "https://data.cdc.gov/resource/9bhg-hcku.json"
```

Use **httr2** to download COVID-19 death data from this endpoint. Make sure to remove the default limit to get all available data. Create a clean dataset called `deaths` with columns `state`, `date`, and `deaths` (renamed from the original column name). Ensure dates are in proper Date format and deaths are numeric.

```
deaths_raw <- request(deaths_url) |>  req_url_query(`$limit` =10000000000) |>
↪ req_perform() |> resp_body_json(simplifyDataFrame = TRUE)

deaths <- as_tibble(deaths_raw) |> mutate(state = state, date =
↪ as.Date(end_date),  deaths = as.numeric(covid_19_deaths)) |>
↪ select(state, date, deaths) |> filter(state %in% population$state_name,
↪ !is.na(deaths), !is.na(date))
head(deaths)
```

```
# A tibble: 6 x 3
  state   date       deaths
  <chr>   <date>      <dbl>
1 Alabama 2023-09-23  21520
2 Alabama 2023-09-23     19
3 Alabama 2023-09-23     46
4 Alabama 2023-09-23    142
5 Alabama 2023-09-23    267
6 Alabama 2023-09-23    416
```

15. Using the `deaths` dataset you created, make a bar plot showing the total COVID-19 deaths by state. Show only the top 10 states with the highest death counts. Order the bars from highest to lowest and use appropriate labels and title.

```r
deaths |> group_by(state) |> summarise(total_deaths = sum(deaths, na.rm =
↪ TRUE)) |>
  arrange(desc(total_deaths)) |> slice(1:10) |> ggplot(aes(x = reorder(state,
  ↪ total_deaths), y = total_deaths)) + coord_flip() + geom_col(fill =
  ↪ "lightblue") + geom_text(aes(label = scales::comma(total_deaths)),
  ↪ hjust = 1.1) + labs(
    title = "Top 10 States by Total COVID-19 Deaths", x = "State", y = "Total
    ↪ Deaths")
```