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`.

```
## Your code here
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:

```
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.

```
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.

```
response <- request |> req_perform()
resp_status(response)
```

```
[1] 200
```

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

```
# Your code here
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.

```
population <- resp_body_json(response, simplifyVector = TRUE)
```

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.

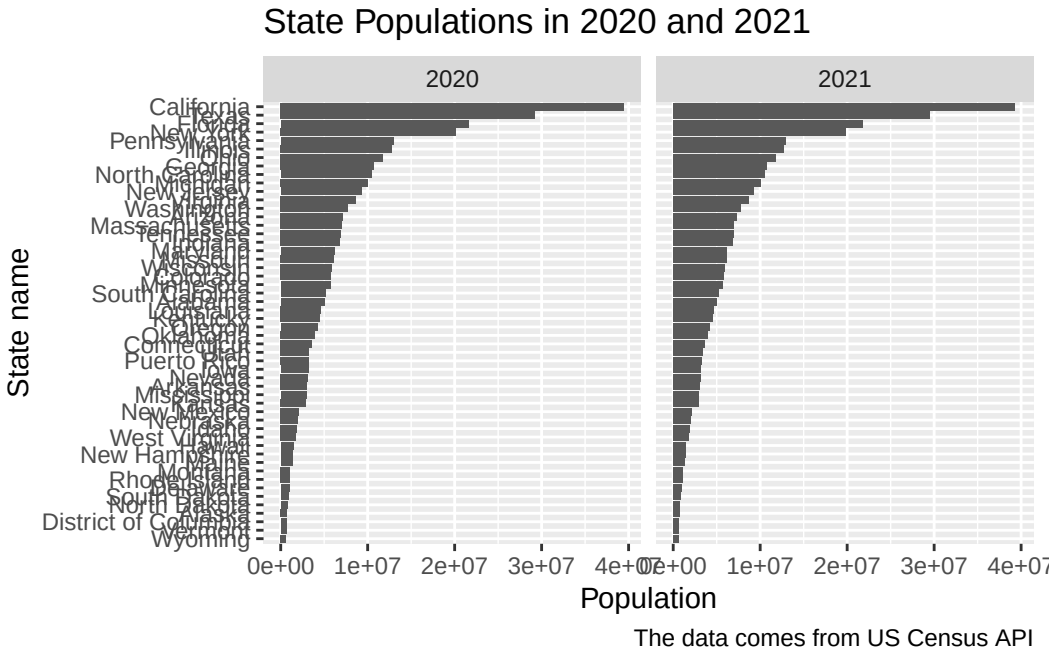
```r
library(tidyverse)
library(janitor)
#population <- population |> ## Use janitor row to names function
  # convert to tibble
  # remove stat column
  # rename state column to state_name
  # use pivot_longer to tidy
  # remove POP_ from year
  # parese all relevant colunns to numeric
  # add state abbreviations using state.abb variable
  # use case_when to add abbreviations for DC and PR

population <- population |> row_to_names(row_number = 1) |>
  as_tibble() |>
  select(-state) |>
  rename(state_name = NAME, `2020` = "POP_2020", `2021` = "POP_2021") |>
  pivot_longer(cols = starts_with("202"), names_to = "year", values_to =
   ↪  "population") |>
  mutate(
    year = as.numeric(year),
    population = as.numeric(population),
    state = case_when(
      state_name %in% state.name ~ state.abb[match(state_name, state.name)],
      state_name == "District of Columbia" ~ "DC",
      state_name == "Puerto Rico" ~ "PR"
    )
  ) |>
  select(state_name, state, year, population)

# Just for inspection here, formal table should be formulated using kable()
 ↪  (the library is not included here)
population
```

```
# A tibble: 104 x 4
   state_name   state  year population
   <chr>        <chr> <dbl>      <dbl>
 1 Oklahoma     OK     2020    3962031
 2 Oklahoma     OK     2021    3986639
 3 Nebraska     NE     2020    1961455
 4 Nebraska     NE     2021    1963692
 5 Hawaii       HI     2020    1451911
 6 Hawaii       HI     2021    1441553
```

```
 7 South Dakota SD     2020     887099
 8 South Dakota SD     2021     895376
 9 Tennessee    TN     2020    6920119
10 Tennessee    TN     2021    6975218
# i 94 more rows
```

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`.

```
# population |>
  # reorder state
  # assign aesthetic mapping
  # use geom_col to plot barplot
  # flip coordinates
  # facet by year

population |> group_by(state_name) |>
  mutate(max_pop = max(population, na.rm = TRUE)) |>  # largest population
   ↪  across both years
  ungroup() |>
  mutate(state_name = reorder(state_name, max_pop)) |>
  ggplot(aes(x = population, y = state_name)) +
  geom_col() +
  facet_wrap(~ year) +
  labs(x = "Population", y = "State name", title = "State Populations in 2020
   ↪  and 2021",
    caption = "The data comes from US Census API")
```



8. The following URL:

```
url <- "https://github.com/datasciencelabs/2025/raw/refs/heads/main/data/reg
 ↪  ions.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)
url <- "https://github.com/datasciencelabs/2025/raw/refs/heads/main/data/reg
 ↪  ions.json"
# regions <- use jsonlit JSON parser
# regions <- convert list to data frame. You can use map_df in purrr package

regions <- fromJSON(url) |>
  mutate(region_name = case_when(
      region_name == "New York and New Jersey, Puerto Rico, Virgin Islands" ~
 ↪  "NY/NJ + PR", TRUE ~ region_name),
      region = as.numeric(region)) |>
```

```
  unnest_longer(states, values_to = "state_name")  |>
  filter(state_name %in% c(state.name, "District of Columbia", "Puerto
   ↪  Rico"))

# Just for inspection here, formal table should be formulated using kable()
 ↪  (the library is not included here)
regions
```

```
# A tibble: 52 x 3
   region region_name  state_name
    <dbl> <chr>        <chr>
 1      1 New England  Connecticut
 2      1 New England  Maine
 3      1 New England  Massachusetts
 4      1 New England  New Hampshire
 5      1 New England  Rhode Island
 6      1 New England  Vermont
 7      2 NY/NJ + PR   New Jersey
 8      2 NY/NJ + PR   New York
 9      2 NY/NJ + PR   Puerto Rico
10      3 Mid-Atlantic Delaware
# i 42 more rows
```

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

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

# Just for inspection here, formal table should be formulated using kable()
 ↪  (the library is not included here)
population
```

```
# A tibble: 104 x 6
   state_name   state  year population region region_name
   <chr>        <chr> <dbl>      <dbl>  <dbl> <chr>
 1 Oklahoma     OK     2020    3962031      6 South Central
 2 Oklahoma     OK     2021    3986639      6 South Central
 3 Nebraska     NE     2020    1961455      7 Central Plains
 4 Nebraska     NE     2021    1963692      7 Central Plains
 5 Hawaii       HI     2020    1451911      9 Pacific
 6 Hawaii       HI     2021    1441553      9 Pacific
```

```
 7 South Dakota SD     2020     887099      8 Mountain States
 8 South Dakota SD     2021     895376      8 Mountain States
 9 Tennessee    TN     2020    6920119      4 Southeast
10 Tennessee    TN     2021    6975218      4 Southeast
# i 94 more rows
```

10. From reading https://data.cdc.gov/ we learn the endpoint `https://data.cdc.gov/resource/pwn4-m3yp.` 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 <-

response <- request(api) |> req_url_query() |> req_perform()

print(resp_status(response))
```

```
[1] 200
```

```
cases_raw <- resp_body_json(response, simplifyVector = TRUE) |> as_tibble()

nrow(cases_raw)
```

```
[1] 1000
```

```
# This is not all the data. There are only 1000 rows in the data frame, and
 ↪  this is because the CDC API limit how much we can download, and the
 ↪  default is 1000. This suggests that we need to change the limit later.
```

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 <-

response <- request(api) |> req_url_query(`$limit` = 10000000000) |>
 ↪  req_perform()

print(resp_status(response))
```

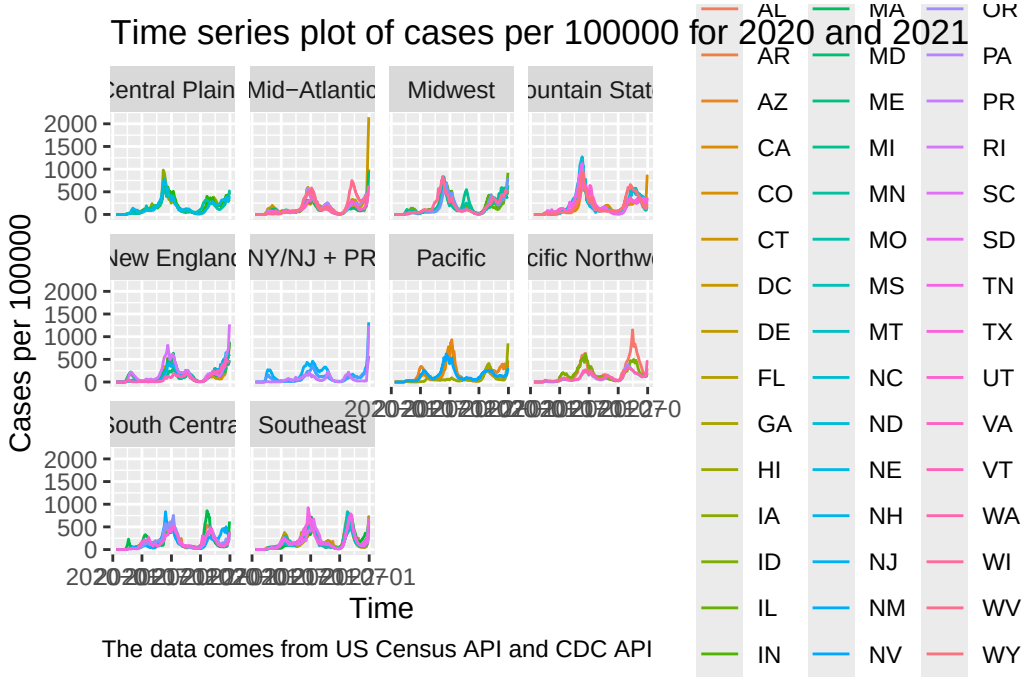
```
[1] 200
```

```
cases_raw <- resp_body_json(response, simplifyVector = TRUE) |> as_tibble()

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

# Just for inspection here, formal table should be formulated using kable()
 ↪  (the library is not included here)
cases_raw
```

```
# A tibble: 8,996 x 3
   state date       cases
   <chr> <date>     <dbl>
 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
 7 LA    2023-04-26  1884
 8 NV    2023-03-15  1233
 9 FL    2023-05-10  6937
10 KS    2022-09-28  2593
# i 8,986 more rows
```

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.

```
#cases |>

cases_raw |>
  filter(as.numeric(format(date, "%Y")) %in% c(2020, 2021)) |>
  inner_join(population,  by = "state", relationship = "many-to-many") |>
  ggplot(aes(x = date, y = 100000 * cases / population, colour = state)) +
  geom_line() +
  facet_wrap(~ region_name) +
  labs(x = "Time", y = "Cases per 100000", title = "Time series plot of cases
   ↪  per 100000 for 2020 and 2021",
    caption = "The data comes from US Census API and CDC API")
```



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.

```
#cases |>

library(lubridate)
```

```r
library(knitr)

# Date are already in Date type, as processed in Q11
cases_raw |>
  filter(year(date) %in% c(2020, 2021)) |>
  group_by(
    year  = year(date),
    month = month(date, label = TRUE, abbr = FALSE, locale = "C")
  ) |>
  summarise(total_cases = sum(cases, na.rm = TRUE), .groups = "drop") |>
  arrange(year, month) |>
  kable(caption = "Total new COVID-19 cases by month from 2020 to 2021")
```

Table 1: Total new COVID-19 cases by month from 2020 to 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
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.

```
# Your code here

response <- request(deaths_url) |> req_url_query(`$limit` = 10000000000) |>
 ↪  req_perform()

print(resp_status(response))
```

```
[1] 200
```

```
deaths <- resp_body_json(response, simplifyVector = TRUE) |> as_tibble()

deaths <- deaths |>
  filter(sex == "All Sexes", age_group == "All Ages", state %in%
   ↪  c(state.name, "District of Columbia", "Puerto Rico")) |>
  select(state, date = end_date, deaths = covid_19_deaths) |>
  mutate(deaths = as.numeric(deaths), date = as.Date(date)) |>
  filter(!is.na(state), !is.na(date), !is.na(deaths))

# Just for inspection here, formal table should be formulated using kable()
deaths
```

```
# A tibble: 2,500 x 3
   state                date       deaths
   <chr>                <date>      <dbl>
 1 Alabama              2023-09-23  21520
 2 Alaska               2023-09-23   1492
 3 Arizona              2023-09-23  30307
 4 Arkansas             2023-09-23  12663
 5 California           2023-09-23 109248
 6 Colorado             2023-09-23  15378
 7 Connecticut          2023-09-23  12571
 8 Delaware             2023-09-23   3445
 9 District of Columbia 2023-09-23   2231
```

```
10 Florida              2023-09-23  81894
# i 2,490 more rows
```

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
# Your code here

deaths |> group_by(state) |>
  summarise(total_deaths = sum(deaths, na.rm = TRUE), .groups = "drop") |>
  arrange(desc(total_deaths)) |>
  slice_head(n = 10) |>
  ggplot(aes(x = reorder(state, -total_deaths), y = total_deaths)) +
  geom_col() +
  labs(x = "State", y = "Total deaths", title = "Top 10 states by total
   ↪  COVID-19 deaths",
    caption = "The data comes from CDC")
```