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`.

```
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:*", #'' because for is a preserved keyword in base R
    key = census_key
  )

#request
```

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.

```
response |> resp_content_type()
```

```
[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)
 ↪  #simplifyVector for matrix form
```

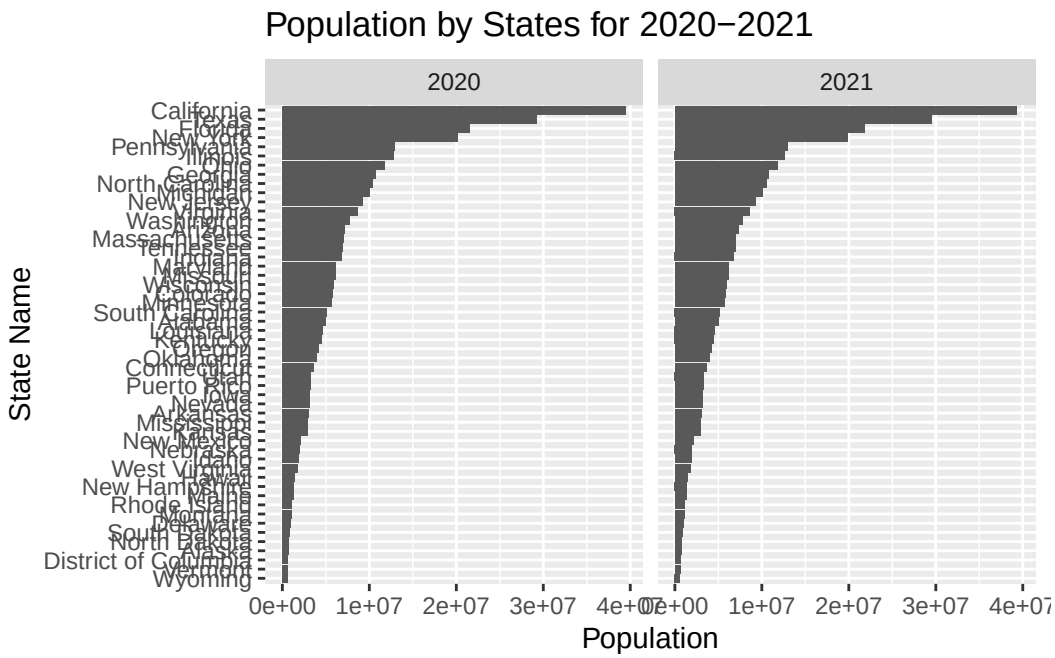
6. Examine the `population` matrix you just created. Notice that 1) it is not tidy, 2) the column types are not what we want, and 3) the first row is a header. Convert `population` to a tidy dataset. Remove the state ID column and change the name of the column with state names to `state_name`. Add a 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.

```
library(tidyverse)
library(janitor)

population <- population |>
  as_tibble() |> # convert to tibble
  row_to_names(row_number = 1) |> # Use janitor row to names function
  rename(state_name = NAME) |> # rename state column to state_name
  select(-state) |> # remove stat column
  pivot_longer(cols = starts_with("POP_"),   # use pivot_longer to tidy
               names_to = "year",
               values_to = "population") |>
  mutate(
    year = str_remove(year,"POP_"), # remove POP_ from year
    year = as.integer(year),
    population = as.numeric(population),   # parese all relevant colunns to
     ↪  numeric
    state = case_when(   # use case_when to add abbreviations for DC and PR
      state_name == "District of Columbia" ~ "DC",
      state_name == "Puerto Rico" ~ "PR",
      TRUE ~ state.abb[match(state_name, state.name)] # add state
       ↪  abbreviations using state.abb variable
    )
  )
```

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 |>
  mutate(state_name = reorder(state_name, population)) |> # reorder state
  ggplot(aes(x = population, y = state_name)) + # assign aesthetic mapping
    geom_col() +# use geom_col to plot barplot
    #coord_flip() + # flip coordinates
    facet_wrap(~year) + # facet by year
    labs(title = "Population by States for 2020-2021", x = "Population", y =
     ↪  "State Name")
```



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 <- fromJSON(url)

#str(regions) # shows that the data is nested
regions <- regions |>
  as_tibble() |>
  unnest(states) |>       # expands list of states into rows
  unnest(region)
```

```
regions <- regions |>
  filter(states %in% c(state.name, "District of Columbia", "Puerto Rico")) |>
  mutate(region_name = ifelse(
    region_name == "New York and New Jersey, Puerto Rico, Virgin Islands",
 ↪  "Region 2", region_name
  ))
#class(regions)
#glimpse(regions)
#regions
```

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

```
# Renaming states to states_name in regions dataframe to avoid naming
 ↪  mismatch
regions <- regions |>
  rename(state_name = states)

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

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()

#response <- request(api) |> req_perform()
#resp_status(response)
#check response = 200

length(cases_raw)
```

```
[1] 1000
```

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(simplifyVector = TRUE)

#response <- request(api) |> req_url_query('$limit' = 10000000000) |>
 ↪  req_perform()
#resp_status(response)
#check response = 200

cases_raw <- cases_raw |>
  select(state = state, date = end_date, cases = new_cases) |>
  mutate(
    cases = as.numeric(cases),
    date = as.Date(date)
  )
```

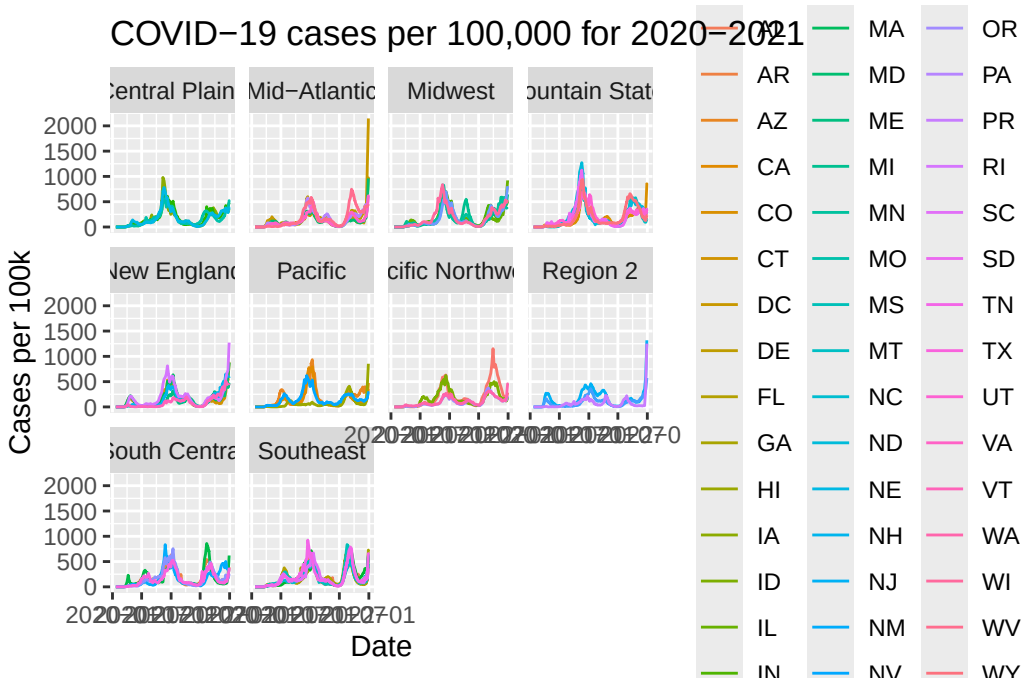
12. For 2020 and 2021, make a time series plot of cases per 100,000 versus time for each state. Stratify the plot by region name. Make sure to label you graph appropriately.

```
cases <- cases_raw |>
  filter(year(date) %in% c(2020, 2021)) |>
  left_join(population, by = "state")
```

```
Warning in left_join(filter(cases_raw, year(date) %in% c(2020, 2021)), population, : Detecte
to-many relationship between `x` and `y`.
i Row 1 of `x` matches multiple rows in `y`.
i Row 103 of `y` matches multiple rows in `x`.
i If a many-to-many relationship is expected, set `relationship =
  "many-to-many"` to silence this warning.
```

```
cases |>
  filter(!is.na(state), !is.na(date), !is.na(cases), !is.na(region_name)) |>
  ggplot(aes(x = date, y = (cases / population)*100000, color = state)) +
```

```
    geom_line() +
    facet_wrap(~region_name) +
    labs(title = "COVID-19 cases per 100,000 for 2020-2021", x = "Date", y =
     ↪  "Cases per 100k")
```



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(lubridate)
library(knitr)
#cases
cases |>
  mutate(date = lubridate::ymd(date)) |>
  filter(lubridate::year(date) %in% c(2020, 2021)) |>
  group_by(year = lubridate::year(date), month = lubridate::month(date,
   ↪  label = TRUE, abbr = FALSE)) |>
  summarise(total_cases = sum(cases, na.rm = TRUE), .groups = "drop") |>
```

```
  arrange(year, month) |>
  knitr::kable(caption = "Total cases in 2020-2021", align = "c")
```

Table 1: Total cases in 2020-2021

year	month	total_cases
2020	January	22
2020	February	136
2020	March	118580
2020	April	1796680
2020	May	1267634
2020	June	1297456
2020	July	3966528
2020	August	2913676
2020	September	2817355
2020	October	3237530
2020	November	7819868
2020	December	13934668
2021	January	11457178
2021	February	5211475
2021	March	3997190
2021	April	3467780
2021	May	1921868
2021	June	978452
2021	July	2258379
2021	August	7091969
2021	September	9988344
2021	October	4674156
2021	November	4611932
2021	December	10909035

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.

```r
# response <- request(deaths_url) |>
#   req_url_query(`$limit` = 10000000000) |>
#   req_perform()
# resp_status(response)
#check response = 200

deaths <- request(deaths_url) |>
  req_url_query(`$limit` = 10000000000) |>
  req_perform() |>
  resp_body_json(simplifyVector = TRUE) |>
  as_tibble() |>
  select(state, date = end_date, deaths = covid_19_deaths) |>
  mutate(
    date = as.Date(date),
    deaths = as.numeric(deaths)
  ) |>
  filter(!is.na(deaths), !(state %in% c("United States", "New York City",
   ↪  "District of Columbia", "Puerto Rico"))) #filter out non-state entities
deaths
```

```
# A tibble: 90,626 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
 7 Alabama 2023-09-23    670
 8 Alabama 2023-09-23   1053
 9 Alabama 2023-09-23   1628
10 Alabama 2023-09-23   4563
# i 90,616 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
deaths |>
  group_by(state) |>
```

```
  summarise(total_deaths = sum(deaths, na.rm = TRUE)) |>
  arrange(desc(total_deaths)) |>
  head(n = 10) |>
  ggplot(aes(x = total_deaths, y = reorder(state, total_deaths))) +
  geom_col(fill = "skyblue") +
  labs(title = "Top 10 States by COVID-19 Deaths", x = "Total deaths", y =
   ↪  "State")
```