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")
census_key <- Sys.getenv("CENSUS_API_KEY")
```

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
  )

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.

```
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) |> (\(x) do.call(rbind, x))()
```

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.

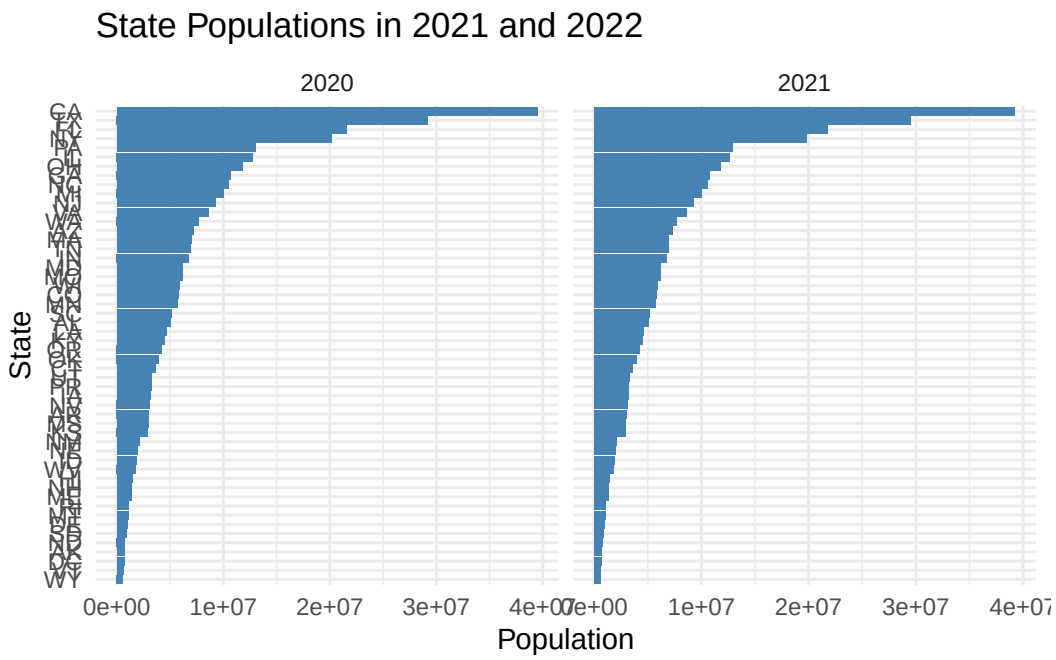
```
library(tidyverse)
library(janitor)
population <- as_tibble(population) |>  # convert to tibble
  row_to_names(row_number = 1) |>          # Use janitor row to names
   ↪ function
  select(-state) |>                         # remove state id (numbers)
   ↪ column
  rename(state_name = NAME) |>             # rename state column to
   ↪ state_name
  pivot_longer(                             # use pivot_longer to tidy
    cols = starts_with("POP_"),
    names_to = "year",
    values_to = "population"
  ) |>
  mutate(
    year = parse_number(year),             # remove "POP_" and keep numeric
     ↪ year
    population = as.numeric(population),
    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`.

```
str(population$population)
```

```
 num [1:104] 3962031 3986639 1961455 1963692 1451911 ...
```

```r
population |>
  ggplot(aes(x = reorder(state, population), y = population)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  facet_wrap(~year, scales = "free_x") +
  labs(
    x = "State",
    y = "Population",
    title = "State Populations in 2021 and 2022"
  ) +
  theme_minimal()
```




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

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)
regions <- map_df(1:nrow(regions), function(i) {
  tibble(
    state_name = regions$states[[i]],       # extract the i-th list element
    region = unlist(regions$region[[i]]),   # convert the list element to
     ↪  scalar
    region_name = regions$region_name[i]    # character vector, no unlist
     ↪  needed
  )
})

regions <- regions |>
  mutate(
    region_name = case_when(
      region_name == "New York and New Jersey, Puerto Rico, Virgin Islands" ~
        "NY/NJ, PR, VI",
      TRUE ~ region_name
    )
  )
```

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

```
population <- population |>
  mutate(state_name = as.character(state_name)) |>
  left_join(regions, by = "state_name")
```

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"
response <- request(api) |> req_perform()
cases_raw <- resp_body_json(response, simplifyVector = FALSE) |> (\(x)
 ↪  do.call(rbind, x))()
cases_raw <- as_tibble(cases_raw, .name_repair = "minimal")
```

- Not all the data is there, because the default of CDC data assess is limited to the 1000 most recent data entries. We did not specify any limits, so the amount of data scrapped is the defaulted amount.

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

cases_raw <- resp_body_json(response, simplifyVector = TRUE)
#str(cases_raw)
cases_df <- as_tibble(cases_raw) |>
  transmute(
    state = state,
    date = as_date(end_date),
    cases = as.numeric(new_cases)
  )

str(cases_df)
```

```
tibble [10,380 x 3] (S3: tbl_df/tbl/data.frame)
 $ state: chr [1:10380] "AZ" "LA" "GA" "LA" ...
 $ date : Date[1:10380], format: "2023-02-22" "2022-12-21" ...
 $ cases: num [1:10380] 3716 4041 5298 2203 5725 ...
```

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
regions <- regions |>
  mutate(state_abbr = case_when(
    state_name == "District of Columbia" ~ "DC",
    state_name == "Puerto Rico" ~ "PR",
    state_name == "Virgin Islands" ~ "VI",
    state_name == "American Samoa" ~ "AS",
    state_name == "Commonwealth of the Northern Mariana Islands" ~ "MP",
    state_name == "Federated States of Micronesia" ~ "FSM",
    state_name == "Guam" ~ "GU",
    state_name == "Marshall Islands" ~ "RMI",
    state_name == "Republic of Palau" ~ "PW",
    TRUE ~ state.abb[match(state_name, state.name)]
  ))

cases_per_capita <- cases_df |>
  mutate(year = as.numeric(format(date, "%Y")))|>
  left_join(
    population |> filter(year %in% c(2020, 2021)) |>
      select(state, population, year) |> distinct(),
    by = c("state", "year")
  ) |>
  mutate(
    cases_per_100k = (cases / population) * 100000
  ) |>
  filter(year %in% c(2020, 2021))
cases_per_capita <- cases_per_capita |>
  left_join(
    regions |> select(state_abbr, region_name),
    by = c("state" = "state_abbr")
  ) |>
  filter(!is.na(cases_per_100k))

ggplot(cases_per_capita, aes(x = date, y = cases_per_100k, color = state)) +
  geom_line(show.legend = FALSE) +
  facet_wrap(~region_name, scales = "free_y") +
  labs(
    title = "COVID-19 Cases per 100,000 by State (2020-2021)",
    x = "Date",
    y = "Cases per 100,000",
    caption = "Source: CDC COVID-19 State-Level Data"
  ) +
  theme_minimal() +
  theme(
```

```
    strip.text = element_text(face = "bold"),
    axis.text.x = element_text(angle = 45, hjust = 1)
  )
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on 'COVID-19 Cases per 100,000 by State (2020-2021)' in
'mbcsToSbcs': dot substituted for <e2>
```

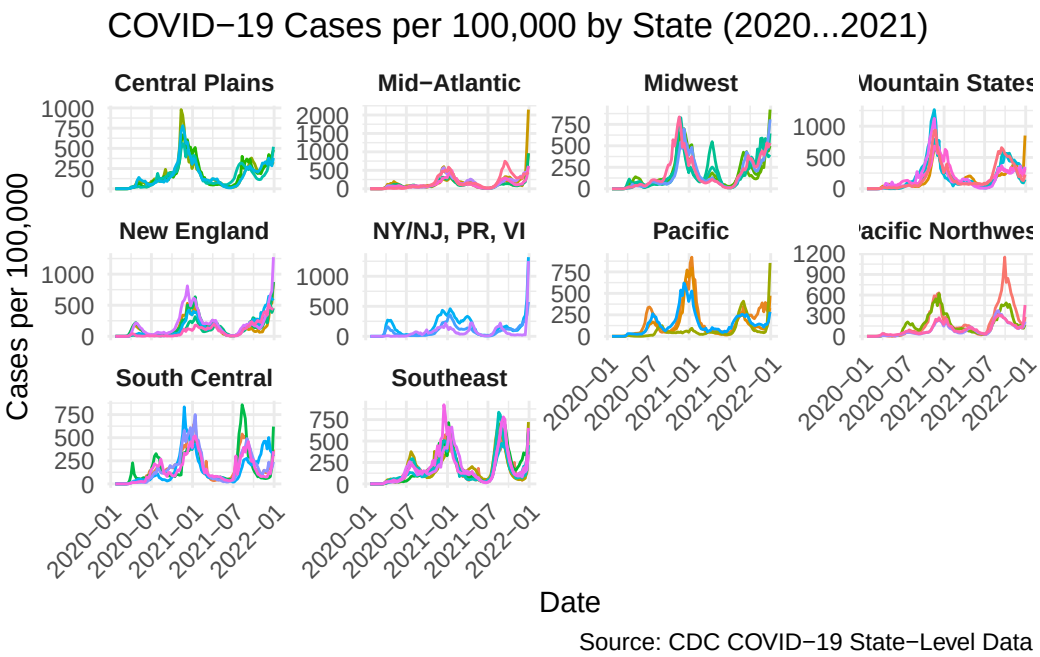
```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on 'COVID-19 Cases per 100,000 by State (2020-2021)' in
'mbcsToSbcs': dot substituted for <80>
```

```
Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on 'COVID-19 Cases per 100,000 by State (2020-2021)' in
'mbcsToSbcs': dot substituted for <93>
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on 'COVID-19 Cases per 100,000 by State (2020-2021)' in
'mbcsToSbcs': dot substituted for <e2>
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on 'COVID-19 Cases per 100,000 by State (2020-2021)' in
'mbcsToSbcs': dot substituted for <80>
```

```
Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
conversion failure on 'COVID-19 Cases per 100,000 by State (2020-2021)' in
'mbcsToSbcs': dot substituted for <93>
```



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(dplyr)
library(knitr)
cases_summary <- cases_df |>
  mutate(
    date = ymd(date),
    year = year(date),
    month = month(date, label = TRUE, abbr = FALSE)
  ) |>
  filter(year %in% c(2020, 2021)) |>
  group_by(year, month) |>
  summarise(total_cases = sum(as.numeric(cases), na.rm = TRUE), .groups =
   ↪  "drop") |>
  arrange(year, month)

kable(cases_summary, caption = "Total COVID-19 Cases by Month and Year
 ↪  (2020-2021)")
```

```
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	68245
2020	April	974032
2020	May	650943
2020	June	654904
2020	July	1989512
2020	August	1461283
2020	September	1415438
2020	October	1628598
2020	November	3932646
2020	December	7027128
2021	January	5808063
2021	February	2667511
2021	March	2068441
2021	April	1773591
2021	May	972915
2021	June	493635
2021	July	1137440
2021	August	3572562
2021	September	5027537
2021	October	2356302
2021	November	2322814
2021	December	5615644

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.

```
response <- request(deaths_url) |>
  req_url_query(`$limit` = 1000000000) |>
  req_perform()

deaths_raw <- resp_body_json(response, simplifyVector = TRUE)
deaths <- as_tibble(deaths_raw) |>
  transmute(
    state = state,
    date = as.Date(substr(end_date, 1, 10)),
    deaths = as.numeric(covid_19_deaths)
  )
```

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.

```
top10_deaths <- deaths |>
  filter(!state == 'United States', !state == 'New York City') |>
  group_by(state) |>
  summarise(total_deaths = sum(deaths, na.rm = TRUE), .groups = "drop") |>
  arrange(desc(total_deaths)) |>
  slice_head(n = 10)

ggplot(top10_deaths, aes(x = reorder(state, -total_deaths), y =
 ↪  total_deaths)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(
    title = "Top 10 States by Total COVID-19 Deaths",
    x = "State",
    y = "Total Deaths"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
```

```
    plot.title = element_text(face = "bold", hjust = 0.5)
  )
```