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 <- "dd06877f1e7a4ca6cde3991db610af34372632f9"
```

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
census_key <- 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$value
  )
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()
response$status_code
```

```
[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)
```

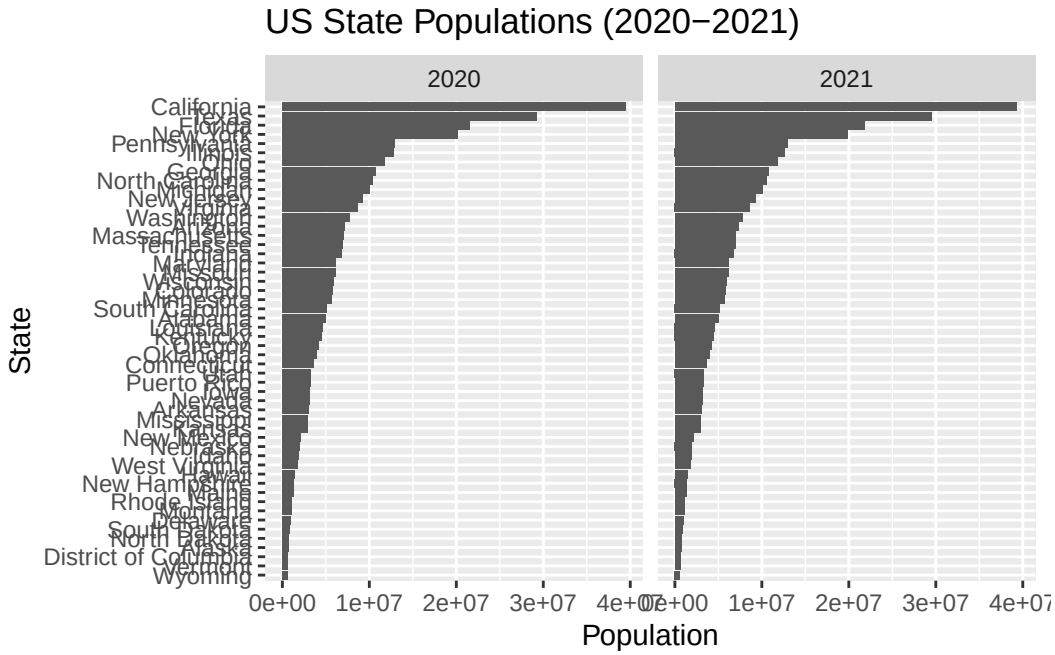
6. Examine the `population` matrix you just created. Notice that 1) it is not tidy, 2) the column types are not what we want, and 3) the first row is a header. Convert `population` to a tidy dataset. Remove the state ID column and change the name of the column with state names to `state_name`. Add a 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 |>
  as_tibble() |>
  row_to_names(row_number = 1) |> # janitor function to col names as
  select(-state) |>
  rename(state_name = NAME) |>
  pivot_longer(cols = starts_with("POP"),
               names_to = "year",
               values_to = "population") |>
  mutate(
    year = str_remove(year, "POP_"),
    population = as.numeric(population),
    state = case_when(
      state_name == "District of Columbia" ~ "DC",
      state_name == "Puerto Rico" ~ "PR",
      TRUE ~ state.abb[match(state_name, state.name)]
    )
  )
```

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

population |>
```

```
  ggplot(aes(x = reorder(state_name, population), y = population)) +
  geom_col() +
  coord_flip() +
  facet_wrap(~year) +
  labs(x = "State", y = "Population",
       title = "US State Populations (2020-2021)")
```



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 <- url |>
  fromJSON() |>
  mutate(region_name = ifelse(region_name == "New York and New Jersey,
   ↪  Puerto Rico, Virgin Islands", "(NYC, NJ, PR, VI)", region_name ))|>
  unnest(c(states, region))|>
  rename(state_name = states) |>
  select(state_name, region, region_name) |>
  as.data.frame()

# regions
```

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

```
# population <-

# merge regions data frame with population so that each state in population
 ↪  gets its corresponding region and region_name
# both data frames have a state_name column, you I use a left join
population <- population |>
  left_join(regions, by = "state_name")


# I added the region (numeric) and region_name (character) columns to your
 ↪  population data frame
head(population)
```

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

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 <-

cases_raw <- request(api) |> req_perform() |> resp_body_string() |>
 ↪  fromJSON()


# Check the structure of the data
# str(cases_raw)
# head(cases_raw)
# summary(cases_raw)
#
#
# # Check if there is any NA in the entire data frame
# any(is.na(cases_raw)) # No missing values

# We see exactly 1,000 rows. We should be seeing over $52 \times 3$ rows per
 ↪  state.
```

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.

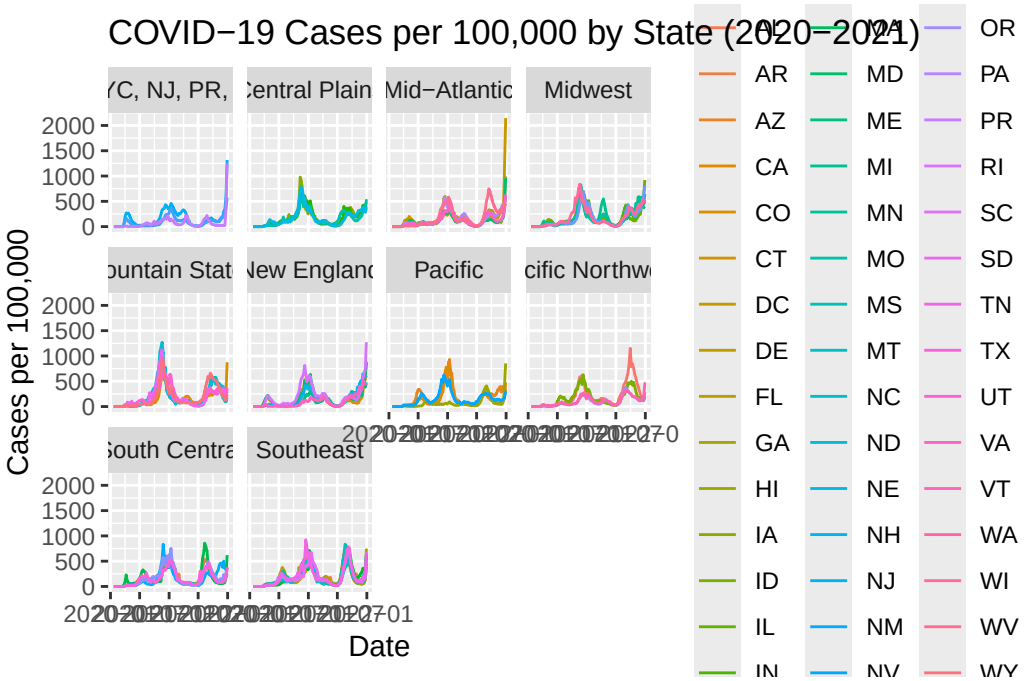
```
api <- "https://data.cdc.gov/resource/pwn4-m3yp.json"

cases_raw <-  request(api) |>
  req_url_query(`$limit` = 10000000000) |>  # remove default limit
  req_perform() |>
  resp_body_string() |>
  fromJSON() |>
# str(cases_raw) they were all chr, need to wrangle
  mutate(
    date = as.Date(end_date),  # convert end_date to Date
    cases = as.numeric(new_cases)          # convert tot_cases to numeric
  ) |>
```

```
  filter(!is.na(state),!is.na(date),!is.na(cases)) |>
  select(state, date, cases)               # keep only relevant columns

#head(cases_raw)
#str(cases_raw)

# date is end date in df
# cases is new cases in df
```

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_raw |>
  filter(year(date) == 2020 | year(date) == 2021) |>
  left_join(population, by = "state") |>
  filter(!is.na(state),!is.na(date),!is.na(cases),!is.na(population),!is.na(
   ↪  region_name))
   ↪  |>
  ggplot(aes(x = date, y = 10^5*(cases / population) , col = state)) +
  geom_line() +
  facet_wrap(~region_name) +
  labs(
    title = "COVID-19 Cases per 100,000 by State (2020-2021)",
    x = "Date",
    y = "Cases per 100,000"
  )
```

```
Warning in left_join(filter(cases_raw, year(date) == 2020 | year(date) == : Detected an unexp
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.
```



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)
library(lubridate)

cases_raw |>
  filter(year(date) == 2020 | year(date) == 2021) |>
  mutate(month = month(date, label = TRUE), day = day(date), year =
   ↪  year(date)) |>
  group_by(year, month) |>
  summarise(
    total_cases = sum(cases)
  ) |>
  arrange(year, month) |>
  kable()
```

```
`summarise()` has grouped output by 'year'. You can override using the
`.groups` argument.
```

year	month	total_cases
2020	Jan	11
2020	Feb	68
2020	Mar	68245
2020	Apr	974032
2020	May	650943
2020	Jun	654904
2020	Jul	1989512
2020	Aug	1461283
2020	Sep	1415438
2020	Oct	1628598
2020	Nov	3932646
2020	Dec	7027128
2021	Jan	5808063
2021	Feb	2667511
2021	Mar	2068441
2021	Apr	1773591
2021	May	972915
2021	Jun	493635
2021	Jul	1137440
2021	Aug	3572562
2021	Sep	5027537
2021	Oct	2356302
2021	Nov	2322814
2021	Dec	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.

```
# Your code here

death <- request(deaths_url) |>
  req_url_query(`$limit` = 10000000000) |>
  req_perform() |>
  resp_body_string() |>
```

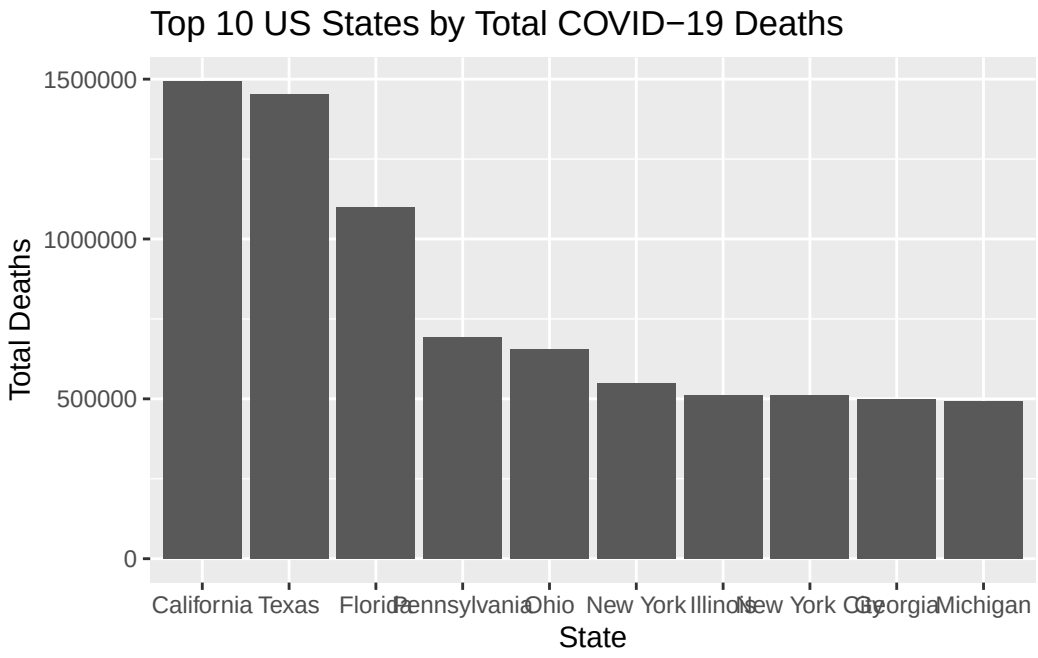
```
  fromJSON() |>
  drop_na(covid_19_deaths, end_date) |>
  summarize(state = state, death = as.numeric(covid_19_deaths), date =
   ↪  as.Date(end_date)) |>
  select(state, date, death) |>
  filter(!(state == "United States")) #|>
```

```
Warning: Returning more (or less) than 1 row per `summarise()` group was deprecated in
dplyr 1.1.0.
i Please use `reframe()` instead.
i When switching from `summarise()` to `reframe()`, remember that `reframe()`
  always returns an ungrouped data frame and adjust accordingly.
```

```
 # print()
```

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.

```
# Your code here
death |>
  group_by(state) |>
  summarize(total_deaths = sum(death), .groups = "drop") |>
  slice_max(total_deaths, n = 10) |>
  ggplot(aes(x = reorder(state, total_deaths, decreasing = TRUE) , y =
   ↪  total_deaths)) +
  geom_col() +
  labs(
    title = "Top 10 US States by Total COVID-19 Deaths",
    x = "State",
    y = "Total Deaths"
  )
```


Top 10 US States by Total COVID−19 Deaths
Total Deaths
1500000
1000000
500000
0
California
Texas
Florida
Pennsylvania
Ohio
New York
Illinois
New York City
Georgia
Michigan
State