

# 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](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:",
    key = census_key
  )
# request
```

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

```
response <- req_perform(request)
# response
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)

population <- do.call(rbind, population) #convert to matrix

# view(population)
```

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
  janitor::row_to_names(row_number = 1) |>## Use janitor row to names
  ↪ function |>
  select(-state) |> # remove stat column
  rename(state_name = NAME) |> # rename state column to state_name
  pivot_longer(cols = c(POP_2020, POP_2021),
    names_to = "year",
    values_to = "pop",
    names_prefix = "POP_") |> # use pivot_longer to tidy # remove
    ↪ POP_ from year
  mutate(across(c(year, pop), as.numeric)) |> # parse all relevant columns
    ↪ to numeric
  mutate(state = state.abb[match(state_name, state.name)]) |> # add state
    ↪ abbreviations using state.abb variable
  mutate(
    state = case_when(
      state_name == "District of Columbia" ~ "DC",
      state_name == "Puerto Rico" ~ "PR",
      .default = state
    )
  )# use case_when to add abbreviations for DC and PR

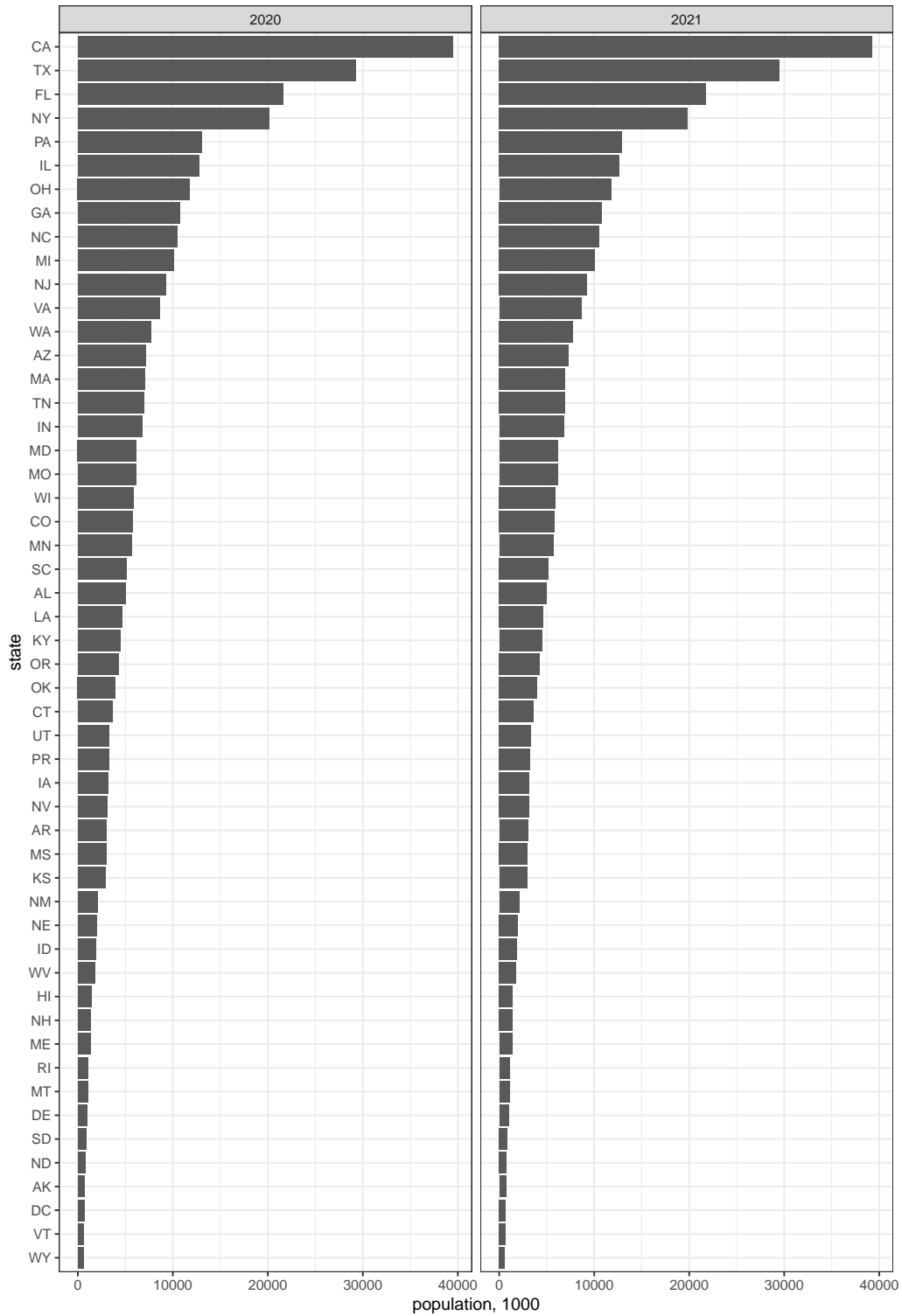
# view(population)
```

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 = reorder(state, pop, mean)) |> # reorder state # by both year
  ggplot(aes(x = state, y = pop/1000)) + # assign aesthetic mapping
  geom_col() + # use geom_col to plot barplot
```

```
coord_flip() + # flip coordinates
facet_wrap(~ year) + # facet by year
labs(y = "population, 1000", # divide by 1000 for clarity
      title = "State population by year") +
theme_bw()
```

State population 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, flatten = TRUE) # use jsonlit JSON parser

regions <- regions |>
  unnest() |> #unnest the lists within the dataframe
  mutate(
    region_name = case_when(
      region_name == "New York and New Jersey, Puerto Rico, Virgin Islands" ~
        "NY, NJ, PR, VI",
      .default = region_name #avoid non-matching values become NA
    )
  ) |>
  rename(state_name = states)

# view(regions)
```

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

```
population$state_name <- as.character(population$state_name) # convert to
  character from list, previously the right join returned an error
population <- right_join(regions, population, by = "state_name")
view(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_string() |>
  fromJSON()

# str(cases_raw)
# 1000 obs. of 10 variables
# There is only 1000 obs because API often limit how much you can download.
↪ We can use the req_url_query to add a limit to the syntax
```

We see exactly 1,000 rows. We should be seeing over  $52 \times 3$  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_string() |>
  fromJSON()

# str(cases_raw) #10380 obs. of 10 variables:
# head(cases_raw)

cases <- cases_raw |>
  as_tibble() |>
  select(state, end_date, new_cases) |>
  rename(date = end_date, cases = new_cases) |>
  mutate(across(c(cases), parse_number)) |>
  mutate(across(c(date), as.Date)) # seconds were all 00, drop the
  ↪ information

# head(cases)
```

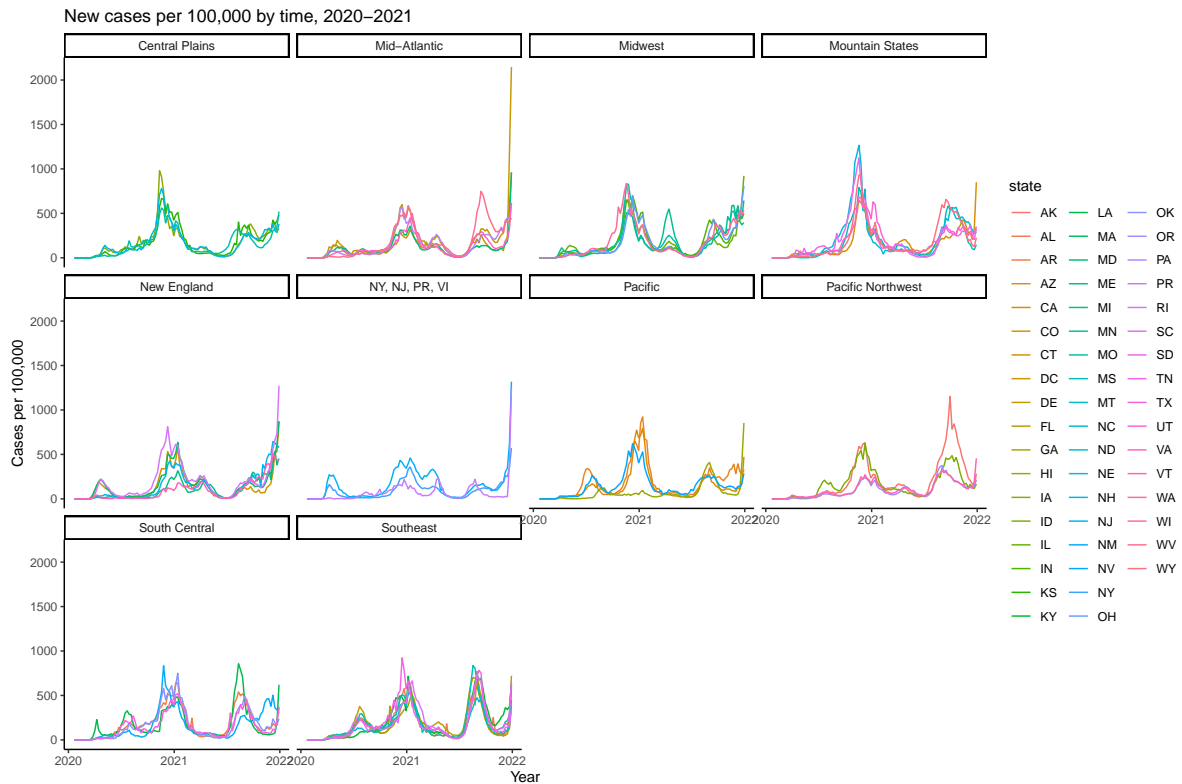
12. For 2020 and 2021, make a time series plot of cases per 100,000 versus time for each state. Stratify the plot by region name. Make sure to label you graph appropriately.

```

cases |>
  mutate(year = year(date)) |>
  inner_join(population, by = c("state", "year")) |># merge with population
  ↪ dataset by abbreviation
  select(state, date, cases, region_name, pop, year) |>
  filter(date >= "2020-01-01" & date < "2022-01-01") |>
  ggplot(aes(x = date, y = cases/pop*100000, group = state, color = state)) +
#   geom_point() +
  geom_line() +
  facet_wrap(~ region_name) +
  scale_x_date(
    date_breaks = "1 year",      # tick every 1 year
    date_labels = "%Y"          # show just the year
  ) +
  labs(
    title = "New cases per 100,000 by time, 2020-2021",
    y = "Cases per 100,000",
    x = "Year"
  ) +
  theme_classic()

```





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)
```

Warning: package 'knitr' was built under R version 4.4.3

```
#| message: false
#| warning: false
#in Q11, dates were already in date format, not a string
cases |>
  mutate(
    year = year(date),
```

```

    month = month(date, label = TRUE)
  ) |>
left_join(population, by = c("state", "year")) |># merge with population
  ↪ dataset by abbreviation
select(state, cases, year, month) |>
filter(year == 2020 | year == 2021) |>
group_by(month, year) |>
summarise(
  totalcases = sum(cases),
  .groups = "drop"
) |>
arrange(year, month) |>
kable()

```

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

```
deaths_raw <- request(deaths_url) |>
  req_url_query(`$limit` = 10000000000) |>
  req_perform() |>
  resp_body_string() |>
  fromJSON()
# head(deaths_raw)

death <- deaths_raw |>
  select(state, end_date, covid_19_deaths) |>
  rename(date = end_date) |>
  rename(deaths = covid_19_deaths) |>
  mutate(across(c(date), as.Date)) |> # str(death) all variables are chr
  mutate(across(c(deaths), as.numeric)) |>
  filter(state != "United States")
# str(death)
# head(death)
```

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.

```
death |>
  group_by(state) |>
  summarise(
    totalcoviddeath = sum(deaths, na.rm = TRUE),
    .groups = "drop"
  ) |>
  arrange(desc(totalcoviddeath)) |>
  slice(1:10) |>
  mutate(state = reorder(state, -totalcoviddeath)) |>
  ggplot(aes(x = state, y = totalcoviddeath)) +
  geom_col() +
  labs(x = "state",
       y = "total COVID-19 deaths",
```

```
title = "Total COVID-19 deaths by state") +
theme_minimal()
```

