Problem set 4

Qianyu Fan

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)
head(population)
```

```
     [,1]       [,2]       [,3]           [,4]
[1,] "POP_2020" "POP_2021" "NAME"         "state"
[2,] "3962031"  "3986639"  "Oklahoma"     "40"
[3,] "1961455"  "1963692"  "Nebraska"     "31"
[4,] "1451911"  "1441553"  "Hawaii"       "15"
[5,] "887099"   "895376"   "South Dakota" "46"
[6,] "6920119"  "6975218"  "Tennessee"    "47"
```

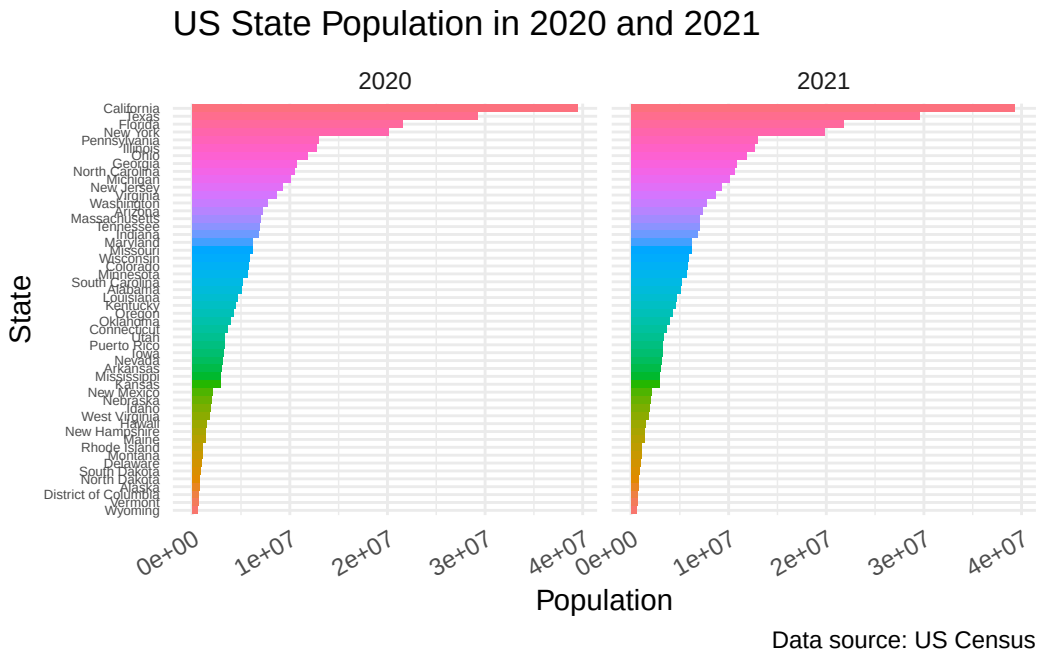
6. Examine the `population` matrix you just created. Notice that 1) it is not tidy, 2) the column types are not what we want, and 3) the first row is a header. Convert `population` to a tidy dataset. Remove the state ID column and change the name of the column with state names to `state_name`. Add a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

```
# A tibble: 104 x 4
   state_name    year population state
   <chr>        <dbl>      <dbl> <chr>
 1 Oklahoma      2020    3962031 OK
 2 Oklahoma      2021    3986639 OK
 3 Nebraska      2020    1961455 NE
 4 Nebraska      2021    1963692 NE
 5 Hawaii        2020    1451911 HI
 6 Hawaii        2021    1441553 HI
 7 South Dakota  2020     887099 SD
 8 South Dakota  2021     895376 SD
 9 Tennessee     2020    6920119 TN
10 Tennessee     2021    6975218 TN
# 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
  mutate(state_name = reorder(state_name, population)) |>
  # assign aesthetic mapping
  ggplot(aes(population, state_name, fill=state_name)) +
  # use geom_col to plot barplot
  geom_col(show.legend = FALSE) +
  # label the axes and add a title
  labs(x = "Population", y = "State",
       title = "US State Population in 2020 and 2021",
       caption = "Data source: US Census") +
  # facet by year
  facet_wrap(~year, ncol = 3) +
  theme_minimal() +
  theme(axis.text.y = element_text(size=5),
        axis.text.x = element_text(angle = 30, hjust = 1))
```



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 three 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"
# use jsonlit JSON parser
regions <- fromJSON(url) |>
  as_tibble() |>
  # Expand list-columns into rows
  unnest_longer(states) |>
  # Rename states as state_names
  rename(state_name = states) |>
  # Change long region name to short
```

```
  mutate(region = as.numeric(region),
         region_name = str_replace(region_name,
           "New York and New Jersey, Puerto Rico, Virgin Islands",
           "NY, NJ, PR, VI")) |>
  # Filter state
  filter(state_name %in% c(state.name, "District of Columbia", "Puerto
   ↪  Rico"))
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, VI New Jersey
 8      2 NY, NJ, PR, VI New York
 9      2 NY, NJ, PR, VI 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 <- left_join(population, regions, by="state_name")
population
```

```
# A tibble: 104 x 6
   state_name    year population state region region_name
   <chr>        <dbl>      <dbl> <chr>  <dbl> <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
 7 South Dakota  2020     887099 SD         8 Mountain States
 8 South Dakota  2021     895376 SD         8 Mountain States
 9 Tennessee     2020    6920119 TN         4 Southeast
```

```
10 Tennessee     2021    6975218 TN         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 <- request(api) |>
  req_perform() |>
  resp_body_json(simplifyDataFrame = TRUE) |>
  as_tibble()
cases_raw
```

```
# A tibble: 1,000 x 10
   date_updated         state start_date end_date tot_cases new_cases tot_deaths
   <chr>                <chr> <chr>      <chr>    <chr>     <chr>     <chr>
 1 2023-02-23T00:00:00~ AZ    2023-02-1~ 2023-02~ 2434631.0 3716.0    33042.0
 2 2022-12-22T00:00:00~ LA    2022-12-1~ 2022-12~ 1507707.0 4041.0    18345.0
 3 2023-02-23T00:00:00~ GA    2023-02-1~ 2023-02~ 3061141.0 5298.0    42324.0
 4 2023-03-30T00:00:00~ LA    2023-03-2~ 2023-03~ 1588259.0 2203.0    18858.0
 5 2023-02-02T00:00:00~ LA    2023-01-2~ 2023-02~ 1548508.0 5725.0    18572.0
 6 2023-03-23T00:00:00~ LA    2023-03-1~ 2023-03~ 1580709.0 1961.0    18835.0
 7 2023-04-27T00:00:00~ LA    2023-04-2~ 2023-04~ 1597070.0 1884.0    18937.0
 8 2023-03-16T00:00:00~ NV    2023-03-0~ 2023-03~ 891702.0  1233.0    11937.0
 9 2023-05-11T00:00:00~ FL    2023-05-0~ 2023-05~ 7572282.0 6937.0    88248.0
10 2022-10-27T00:00:00~ NYC   2022-10-2~ 2022-10~ 2928439.0 14590.0   42863.0
# i 990 more rows
# i 3 more variables: new_deaths <chr>, new_historic_cases <chr>,
#   new_historic_deaths <chr>
```

We see exactly 1,000 rows. We should be seeing over 52×3 rows per state.

Not all data there because the CDC API returns only 1,000 rows by default. We need to use the $limit parameter adjustment to retrieve the complete data.

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.

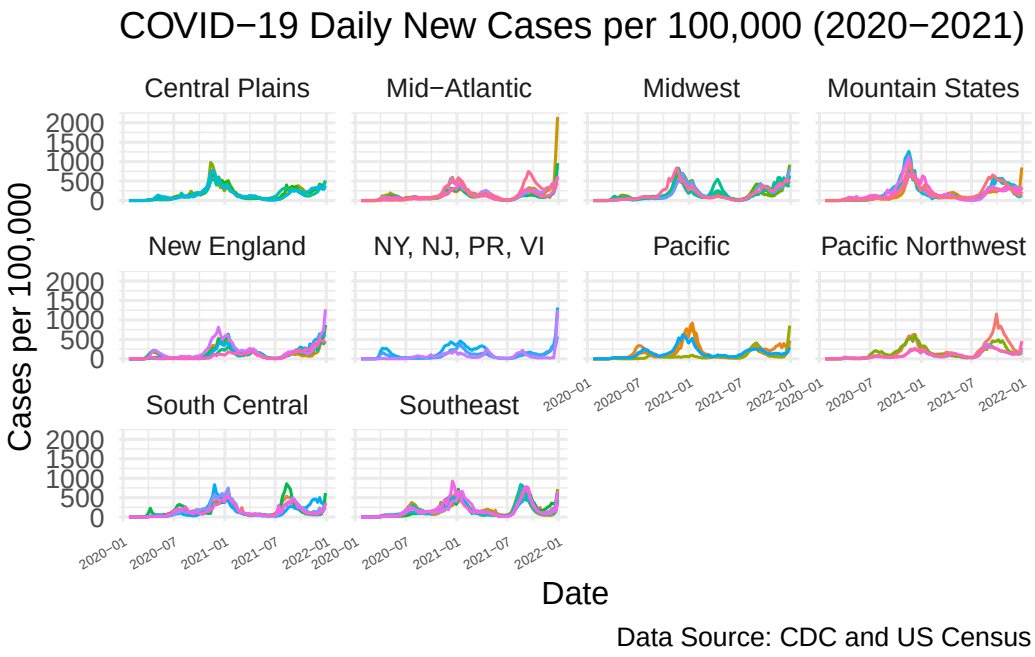
```r
api <- "https://data.cdc.gov/resource/pwn4-m3yp.json"
cases_raw <- request(api) |>
  req_url_query(`$limit` = 10000000000) |>
  req_perform() |>
  resp_body_json(simplifyDataFrame = TRUE) |>
  as_tibble()

cases <- cases_raw |>
  # Selects state, end_date, and new_cases, renaming them correctly
  select(state = state, date = end_date, cases = new_cases) |>
  # Convert cases to numeric and date to Date format
  mutate(cases = as.numeric(cases), date = as.Date(date)) |>
  # Filter only has states, DC, and PR as well as NA
  filter(state %in% c(state.abb, "DC", "PR"), !is.na(date), !is.na(cases))
cases
```

```
# 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.

```r
cases |>
  # Add year column to cases to enable join with population data
  mutate(year = year(date)) |>
  # Filter 2020 and 2021
  filter(year %in% c(2020, 2021)) |>
  # Join with population data
  left_join(population, by = c("state", "year")) |>
  # Filter NA
```

```r
filter(!is.na(state), !is.na(date), !is.na(cases), !is.na(region_name)) |>
# Calculate cases per 100,000
mutate(cases_per100k = (cases / population) * 100000) |>
# Create time series plot
ggplot(aes(date, cases_per100k, colour = state)) +
# Plot line
geom_line() +
# Stratify plot by region name
facet_wrap(~region_name) +
# Label
labs(x = "Date", y="Cases per 100,000",
     title = "COVID-19 Daily New Cases per 100,000 (2020-2021)",
     caption = "Data Source: CDC and US Census",
     color = "State") +
theme_minimal(base_size = 12) +
theme(axis.text.x = element_text(angle = 30, hjust = 1, size = 5),
      legend.position = "None")
```



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 |>
  # Parse date column and create year and month column
  mutate(date = ymd(date),
         year = year(date),
         month = month(date, label = TRUE, abbr = FALSE)) |>
  # Filter 2020 and 2021
  filter(year %in% c(2020, 2021)) |>
  # Group the data by year and month
  group_by(year, month) |>
  # Calculate the total cases across all state
  summarize(total_cases = sum(cases, na.rm = TRUE), .groups = "drop") |>
  # Order by year and month
  arrange(year, month) |>
  # Display Table
  kable(caption = "Monthly Total COVID-19 Cases (2020-2021)",
        col.names = c("Year", "Month", "Total Cases"))
```

Table 1: Monthly Total COVID-19 Cases (2020–2021)

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

Year	Month	Total Cases
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.

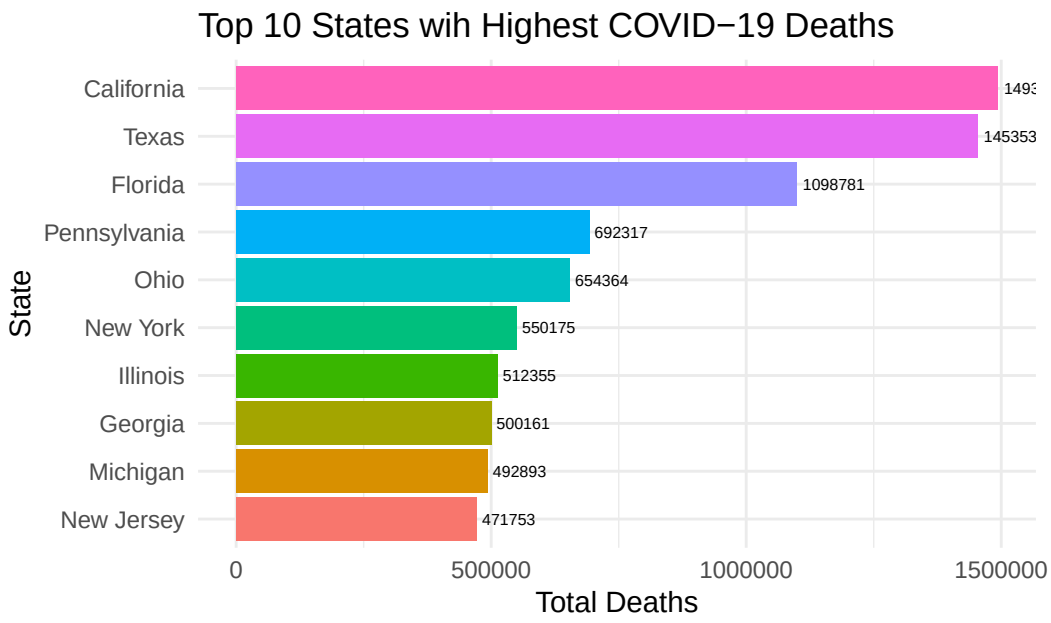
```
# Your code here
deaths_raw <- request(deaths_url) |>
  req_url_query(`$limit` = 10000000000) |>
  req_perform() |>
  resp_body_json(simplifyDataFrame = TRUE) |>
  as_tibble()

deaths <- deaths_raw |>
  # Select state, date, and deaths
  select(state, date = end_date, deaths = covid_19_deaths) |>
  # Ensure dates in proper Date formate and deaths are numeric
  mutate(deaths = as.numeric(deaths), date = as.Date(date)) |>
  # Filter state and NA
  filter(state %in% c(state.name, "District of Columbia", "Puerto Rico"),
         !is.na(deaths), !is.na(date))
deaths
```

```
# A tibble: 93,933 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 93,923 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 data by state
  group_by(state) |>
  # Calculate total deaths by state
  summarize(total_deaths = sum(deaths, na.rm = TRUE), .groups = "drop") |>
  # Order deaths counts from highest to lowest
  arrange(desc(total_deaths)) |>
  # Select Top 10
  slice(1:10) |>
  # Order states from highest to lowest
  mutate(state = reorder(state, total_deaths)) |>
  # assign aesthetic mapping
  ggplot(aes(total_deaths, state, fill=state)) +
  # use geom_col to plot barplot
  geom_col(show.legend = FALSE) +
  # Add number label
  geom_text(aes(label = total_deaths), hjust = -0.1, size = 2) +
  # Add label
  labs(x = "Total Deaths", y = "State",
       title = "Top 10 States wih Highest COVID-19 Deaths",
       caption = "Data Source: CDC") +
  theme_minimal()
```



Data Source: CDC