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.

```r
library(httr2)
request <- request(paste0(
  "https://api.census.gov/data/2021/pep/population?",
  "get=POP_2020,POP_2021,NAME&for=state:*&key=",
  census_key
))
# print(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.

```r
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.

```r
type <- resp_content_type(response)
print(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.

```r
population <- resp_body_json(response,
                             # automatically convert JSON arrays into R vectors
                             simplifyVector = T) |> as.matrix()
# head(population)
class(population)
```

```
[1] "matrix" "array"
```

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.

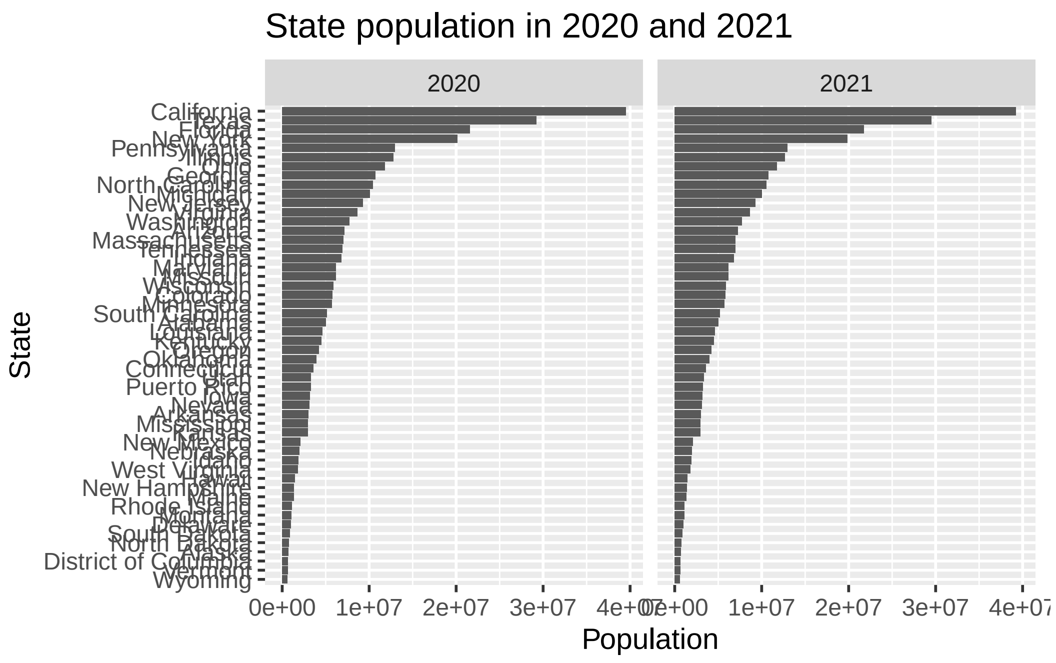
```r
library(tidyverse)
library(janitor)

population <- population |>
  # Use janitor row to names function
  row_to_names(row_number = 1) |>
  # convert to tibble
  as_tibble() |>
  # remove stat column
  select(-state) |>
  # rename state column to state_name
  rename(state_name = NAME) |>
  # use pivot_longer to tidy (POP_2020 and POP_2021 to two columns)
  pivot_longer(
    cols = starts_with("POP_"),
    names_to = "year",
    values_to = "population"
  ) |>
  # remove POP_ from year
  mutate(year = str_remove(year, "POP_")) |>
  # parese all relevant colunns to numeric
  mutate(population = as.numeric(population),
         year = as.numeric(year)) |>
  # add state abbreviations using state.abb variable
  # use case_when to add abbreviations for DC and PR
  mutate(state = case_when(
    state_name == "District of Columbia" ~ "DC",
    state_name == "Puerto Rico" ~ "PR",
    TRUE ~ state.abb[match(state_name, state.name)]
  ))
head(population)
```

```
# A tibble: 6 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. 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 by population sizes (automatically from large to small)
  ggplot(aes(x = reorder(state_name, population),
             y = population)) +
  # assign aesthetic mapping
  labs(
    x = "State",
    y = "Population",
    title = "State population in 2020 and 2021"
  ) +
  # use geom_col to plot barplot
  geom_col() +
  # flip coordinates so state names are on y and population sizes are on x
  coord_flip() +
  # facet by year
  facet_wrap(~ year)
```



8. The following URL:

```r
url <- "https://github.com/datasciencelabs/2025/raw/refs/heads/main/data/regions.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.

```r
library(jsonlite)
library(purrr)
url <- "https://github.com/datasciencelabs/2025/raw/refs/heads/main/data/regions.json"
# regions <- use jsonlit JSON parser
regions_raw <- fromJSON(url)
regions <-
  # regions <- convert list to data frame. You can use map_df in purrr package
  map_df(seq_len(nrow(regions_raw)), function(i) {
  tibble(
    state_name = regions_raw$states[[i]],
    region = regions_raw$region[[i]],
    region_name = regions_raw$region_name[[i]]
  )
}) |>
  # change long name
  mutate(region_name = ifelse(
    region_name == "New York and New Jersey, Puerto Rico, Virgin Islands",
    "Other",
    region_name
  )
) |>
  select(state_name, region, region_name)
head(regions)
```

```
# A tibble: 6 x 3
  state_name    region region_name
  <chr>          <int> <chr>
1 Connecticut        1 New England
2 Maine              1 New England
3 Massachusetts      1 New England
4 New Hampshire      1 New England
5 Rhode Island       1 New England
6 Vermont            1 New England
```

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

```
library(dplyr)
population <- population |>
  left_join(regions, by = "state_name")
head(population)
```

```
# A tibble: 6 x 6
  state_name  year population state region region_name
  <chr>      <dbl>      <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.` 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.

```r
api <- "https://data.cdc.gov/resource/pwn4-m3yp.json"
# create request object
req <- request(api)
head(req)
```

```
$url
[1] "https://data.cdc.gov/resource/pwn4-m3yp.json"

$method
NULL

$headers
list()

$body
NULL

$fields
list()

$options
list()
```

```r
cases_raw <- req |>
  # perform the request (return a response with status code, headers, body)
  req_perform() |>
  # convert JSON arrays into a list of vectors -> data frame
  resp_body_json(simplifyVector = T) |>
  as_tibble()
head(cases_raw)
```

```
# A tibble: 6 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
# i 3 more variables: new_deaths <chr>, new_historic_cases <chr>,
#   new_historic_deaths <chr>
```

```
print("Comments: The cases_raw data contains 1000 rows instead of all rows. This is because t
```

```
[1] "Comments: The cases_raw data contains 1000 rows instead of all rows. This is because the
```

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 = T) |>
  as_tibble()
summary(cases_raw)
```

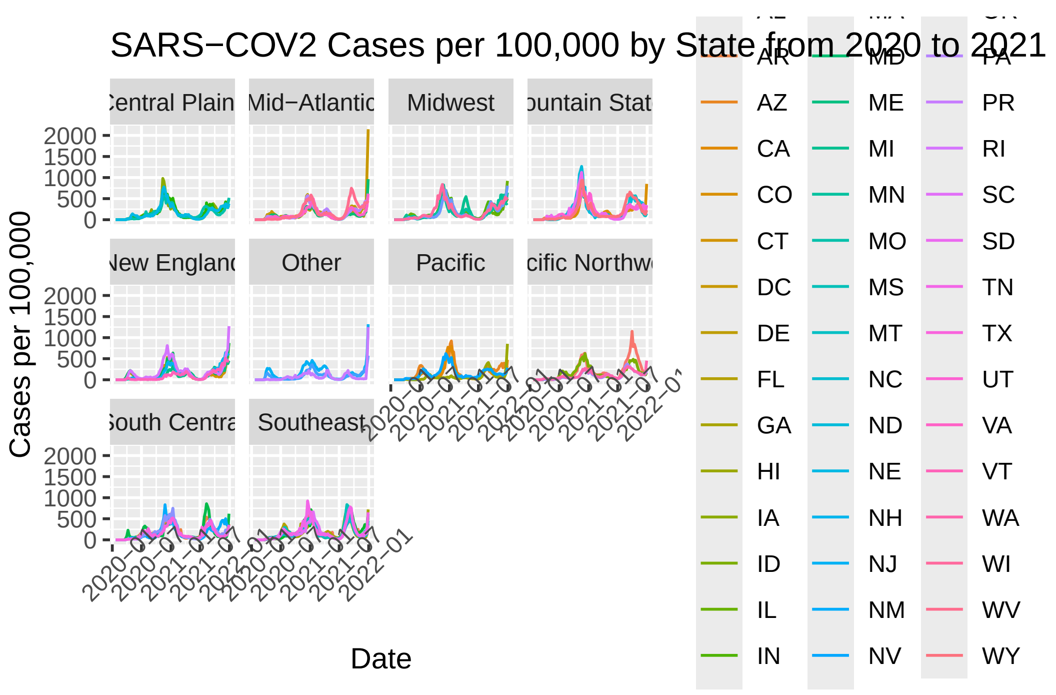
```
 date_updated          state             start_date          end_date        
 Length:10380       Length:10380       Length:10380       Length:10380      
 Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character  
  tot_cases          new_cases          tot_deaths         new_deaths       
 Length:10380       Length:10380       Length:10380       Length:10380      
 Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character  
 new_historic_cases new_historic_deaths
 Length:10380       Length:10380       
 Class :character   Class :character   
 Mode  :character   Mode  :character   
```

```
cases <- cases_raw |>
  mutate(
    # ISO-8601 datetime with a time not just simple date, so ymd can't parse
    date = as.Date(end_date),
    cases = as.numeric(new_cases)
  ) |>
  filter(!is.na(state), !is.na(date), !is.na(cases)) |>
  select(state, date, cases)
head(cases)
```

```
# A tibble: 6 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
```

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)) |>
  filter(year %in% c(2020, 2021)) |>
  left_join(population, by = c("state" = "state", "year" = "year")) |>
  mutate(cases_per_100k = cases / population * 100000) |>
  filter(!is.na(cases_per_100k)) |>
  filter(!is.na(region_name)) |>
  ggplot(aes(x = date, y = cases_per_100k, color = state)) +
  geom_line() +
  facet_wrap(~ region_name) +
  labs(
    x = "Date",
    y = "Cases per 100,000",
    title = "SARS-COV2 Cases per 100,000 by State from 2020 to 2021"
  ) +
  theme(
    axis.text.x = element_text(angle = 45)
  )
```



```
#head(cases)
```

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 |>
  mutate(date = lubridate::ymd(date),
         year = lubridate::year(date),
         month = lubridate::month(date, label = T, abbr = F),
         day = lubridate::day(date)) |>
  filter(year %in% c(2020, 2021)) |>
  group_by(year, month) |>
  summarise(
    total_cases = sum(cases, na.rm = T),
    .groups = "drop"
  ) |>
```

```r
  arrange(year, month) |>
  kable()
```

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

```r
# cases
```

14. The following URL provides additional COVID-19 data from the CDC in JSON format:

```r
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
req2 <- request(deaths_url) |>
  req_url_query(`$limit`=10000000000)
head(req2)
```

```
$url
[1] "https://data.cdc.gov/resource/9bhg-hcku.json?%24limit=10000000000"

$method
NULL

$headers
list()

$body
NULL

$fields
list()

$options
list()
```

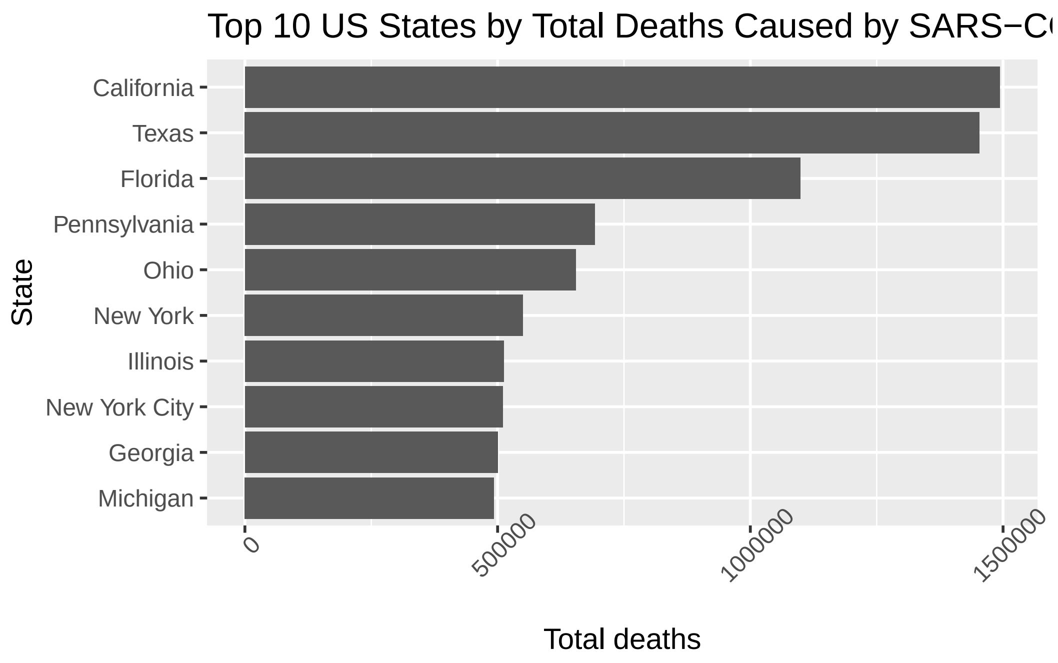
```r
deaths <- req2 |>
  req_perform()|>
  resp_body_json(simplifyVector = T) |>
  as_tibble() |>
  select(state, end_date, covid_19_deaths) |>
  rename(
    date = end_date,
    deaths = covid_19_deaths
  ) |>
  mutate(
    date = as.Date(date),
    deaths = as.numeric(deaths)
  ) |>
  filter(!is.na(state), !is.na(date), !is.na(deaths))
head(deaths)
```

```
# A tibble: 6 x 3
  state         date         deaths
  <chr>         <date>        <dbl>
```

```
1 United States 2023-09-23 1146774
2 United States 2023-09-23     519
3 United States 2023-09-23    1696
4 United States 2023-09-23     285
5 United States 2023-09-23     509
6 United States 2023-09-23    3021
```

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 |>
  filter(state != "United States") |>
  group_by(state) |>
  summarise(
    total = sum(deaths, na.rm = T),
    .groups = "drop"
  ) |>
  arrange(desc(total)) |>
  slice_head(n = 10) |>
  mutate(state = factor(state, levels = rev(state))) |>
  ggplot(aes(x = state, y = total)) +
  geom_col() +
  labs(
    x = "State",
    y = "Total deaths",
    title = "Top 10 US States by Total Deaths Caused by SARS-COV2 Cases"
  ) +
  coord_flip()+
  theme(
    axis.text.x = element_text(angle = 45)
  )
```


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