# Problem set 4

2024-10-04

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 2020 and 2021. 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(url) |> req_url_query(get = I("POP_2020,POP_2021,NAME"),
                                         `for` = I("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.

```r
response <- request |> req_perform()
```

4. Use a function from the **httr2** package to determine the content type of your response.

```r
library('httr2')
request |> req_perform() |> resp_content_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
library(dplyr)
```

```


Attaching package: 'dplyr'


The following objects are masked from 'package:stats':

    filter, lag


The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(janitor)
```

```
Attaching package: 'janitor'


The following objects are masked from 'package:stats':

    chisq.test, fisher.test
```

```r
library(tidyr)
library(stringr)
population <- response |> resp_body_json(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 |>
  row_to_names(1)|>
  as_tibble()|>
  select(-state)|>
  rename(state_name=NAME) |>
  pivot_longer(-state_name, names_to = 'year',values_to ='population')|>
  mutate(year = str_remove(year,"POP_"))|>
  mutate(across(-state_name,as.numeric))|>
  mutate(state= state.abb[match(state_name,state.name)])  |>
  mutate(state = case_when (state_name == "Puerto Rico" ~ "PR",
                            state_name == "District of Columbia" ~ "DC",
                            .default = state))
head(population)
```

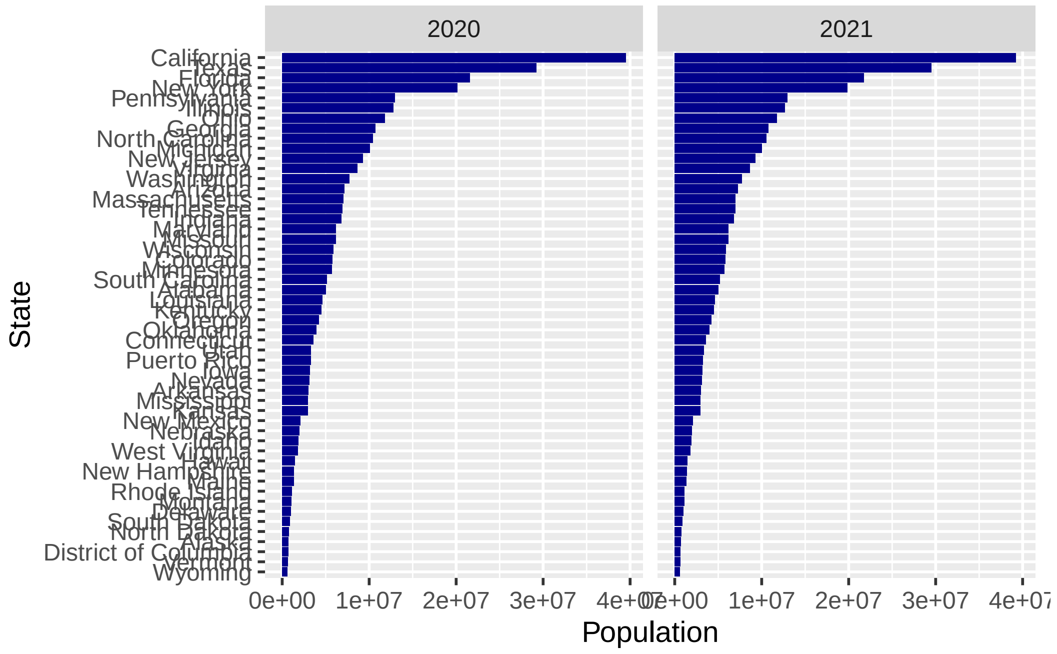
```
# A tibble: 6 x 4
  state_name  year population state
  <chr>      <dbl>      <dbl> <chr>
```

```
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
```

```
#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
```

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 |>
  ggplot(aes(x = reorder(state_name, population), y = population))+
  geom_col(fill='darkblue') +
  coord_flip()+
  facet_wrap(~year)+
  labs(x = "State", y = "Population")
```



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

8. The following URL:

```r
url <- "https://github.com/datasciencelabs/2024/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)
regions <- fromJSON(url,simplifyDataFrame = FALSE)
regions <- map_df(regions, function(x)
  data.frame(region = x$region, region_name = x$region_name, state_name = x$states)) |>
  mutate(region_name = ifelse(region_name == "New York and New Jersey, Puerto Rico, Virgin Is
head(regions)
```

```
  region region_name    state_name
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
```

```
# regions <- use jsonlit JSON parser
# regions <- convert list to data frame. You can use map_df in purrr package
```

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

```
population <- left_join(population, 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.

```
api <- "https://data.cdc.gov/resource/pwn4-m3yp.json"
cases_raw <- request(api) |> req_perform() |> resp_body_json(simplifyVector = TRUE)
head(cases_raw)
```

```
             date_updated state              start_date                end_date
1 2023-02-23T00:00:00.000    AZ 2023-02-16T00:00:00.000 2023-02-22T00:00:00.000
2 2022-12-22T00:00:00.000    LA 2022-12-15T00:00:00.000 2022-12-21T00:00:00.000
3 2023-02-23T00:00:00.000    GA 2023-02-16T00:00:00.000 2023-02-22T00:00:00.000
4 2023-03-30T00:00:00.000    LA 2023-03-23T00:00:00.000 2023-03-29T00:00:00.000
5 2023-02-02T00:00:00.000    LA 2023-01-26T00:00:00.000 2023-02-01T00:00:00.000
6 2023-03-23T00:00:00.000    LA 2023-03-16T00:00:00.000 2023-03-22T00:00:00.000
```

```
  tot_cases new_cases tot_deaths new_deaths new_historic_cases
1 2434631.0    3716.0    33042.0       39.0              23150
2 1507707.0    4041.0    18345.0       21.0              21397
3 3061141.0    5298.0    42324.0       88.0               6800
4 1588259.0    2203.0    18858.0       23.0               5347
5 1548508.0    5725.0    18572.0       47.0               4507
6 1580709.0    1961.0    18835.0       35.0               2239
  new_historic_deaths
1                   0
2                   0
3                   0
4                   0
5                   0
6                   0
```

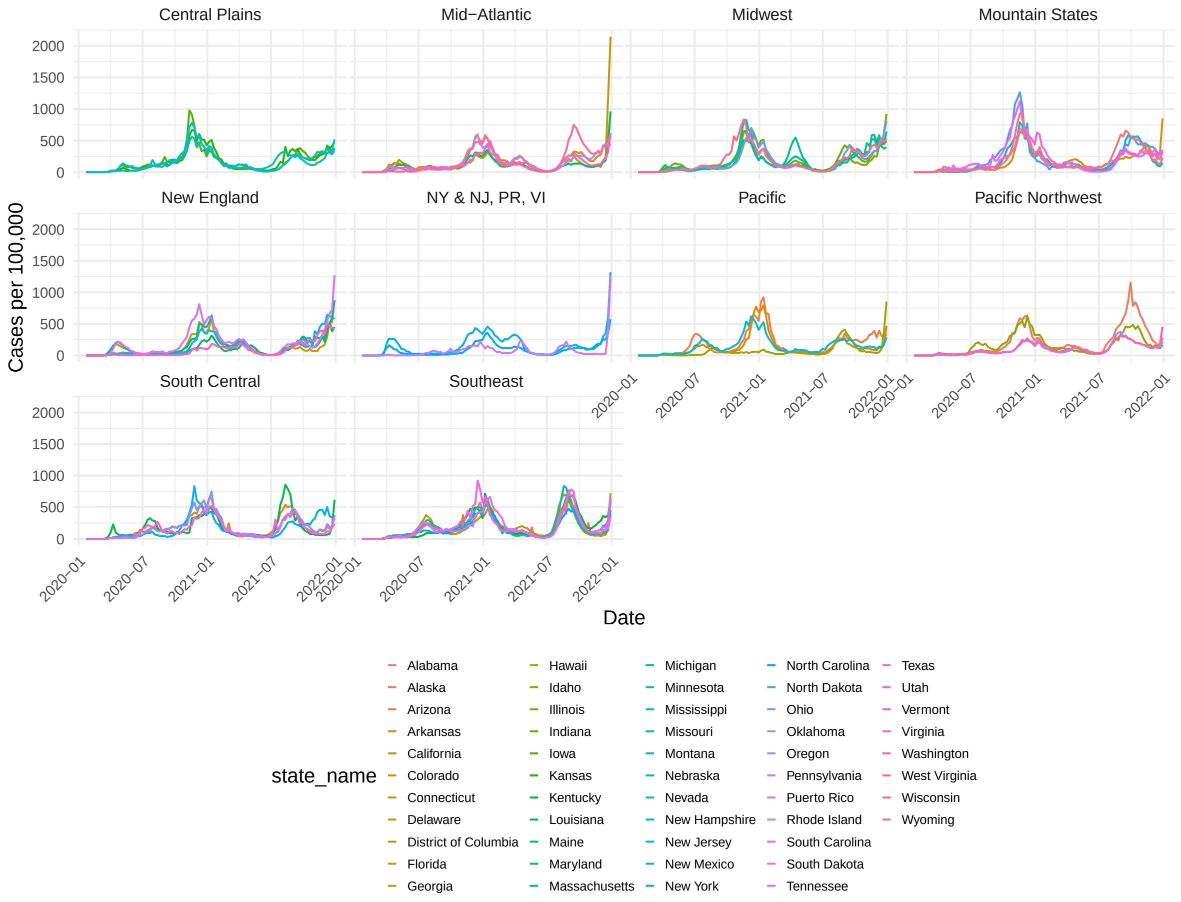
No, data is not all there. We see exactly 1,000 rows. We should be seeing over 52 × 3 rows per state.

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?$limit=10000000000"
cases_raw <- request(api) |> req_perform() |> resp_body_json(simplifyVector = TRUE)
cases <- cases_raw|>
  mutate(date = as.Date(end_date),cases = as.numeric(new_cases))|>
  select(state, date, 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|> filter(year(date) %in% c(2020, 2021))|>
  mutate(year = year(date))|>
  left_join(population, by =c("state",'year')) |>
  mutate(cases_per_100k = (cases / population) * 100000) |>
  filter(!is.na(cases_per_100k))|>
  ggplot(aes(x = date, y = cases_per_100k, color = state_name))+
  geom_line()+
  facet_wrap(~region_name) +
  theme(axis.text.y = element_text(angle = 0, hjust = 3, vjust = 2)) +
  labs(x = "Date",
       y = "Cases per 100,000") +  # Labels
  theme_minimal()  +  # Use a minimal theme
  theme(
    panel.spacing = unit(0.1, "lines"),
    legend.position = "bottom",
    axis.text.x = element_text(size = 8, angle = 45, hjust = 1),
    axis.text.y = element_text( size = 8),
    legend.key.size = unit(0.2, "cm"),
    legend.text = element_text(size = 7)
  )
```


Central Plains
Mid−Atlantic
Midwest
Mountain States
New England
NY & NJ, PR, VI
Pacific
Pacific Northwest
South Central
Southeast
Cases per 100,000
Date
state_name
Alabama
Alaska
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
District of Columbia
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky
Louisiana
Maine
Maryland
Massachusetts
Michigan
Minnesota
Mississippi
Missouri
Montana
Nebraska
Nevada
New Hampshire
New Jersey
New Mexico
New York
North Carolina
North Dakota
Ohio
Oklahoma
Oregon
Pennsylvania
Puerto Rico
Rhode Island
South Carolina
South Dakota
Tennessee
Texas
Utah
Vermont
Virginia
Washington
West Virginia
Wisconsin
Wyoming