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 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.

```
library(httr2)
request_url <- paste0(url, "?get=POP_2020,POP_2021,NAME&for=state:*&key=",census_key)
request <- request(request_url)
```

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 <- req_perform(request)
print(paste("Response status is 200?", response$status_code == 200))
```

```
[1] "Response status is 200? TRUE"
```

```
print(response)
```

```
<httr2_response>

GET
https://api.census.gov/data/2021/pep/population?get=POP_2020,POP_2021,NAME&for=state:*&key=0

Status: 200 OK

Content-Type: application/json

Body: In memory (2112 bytes)
```

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.

```r
population <- resp_body_json(response, simplifyVector = TRUE)
```

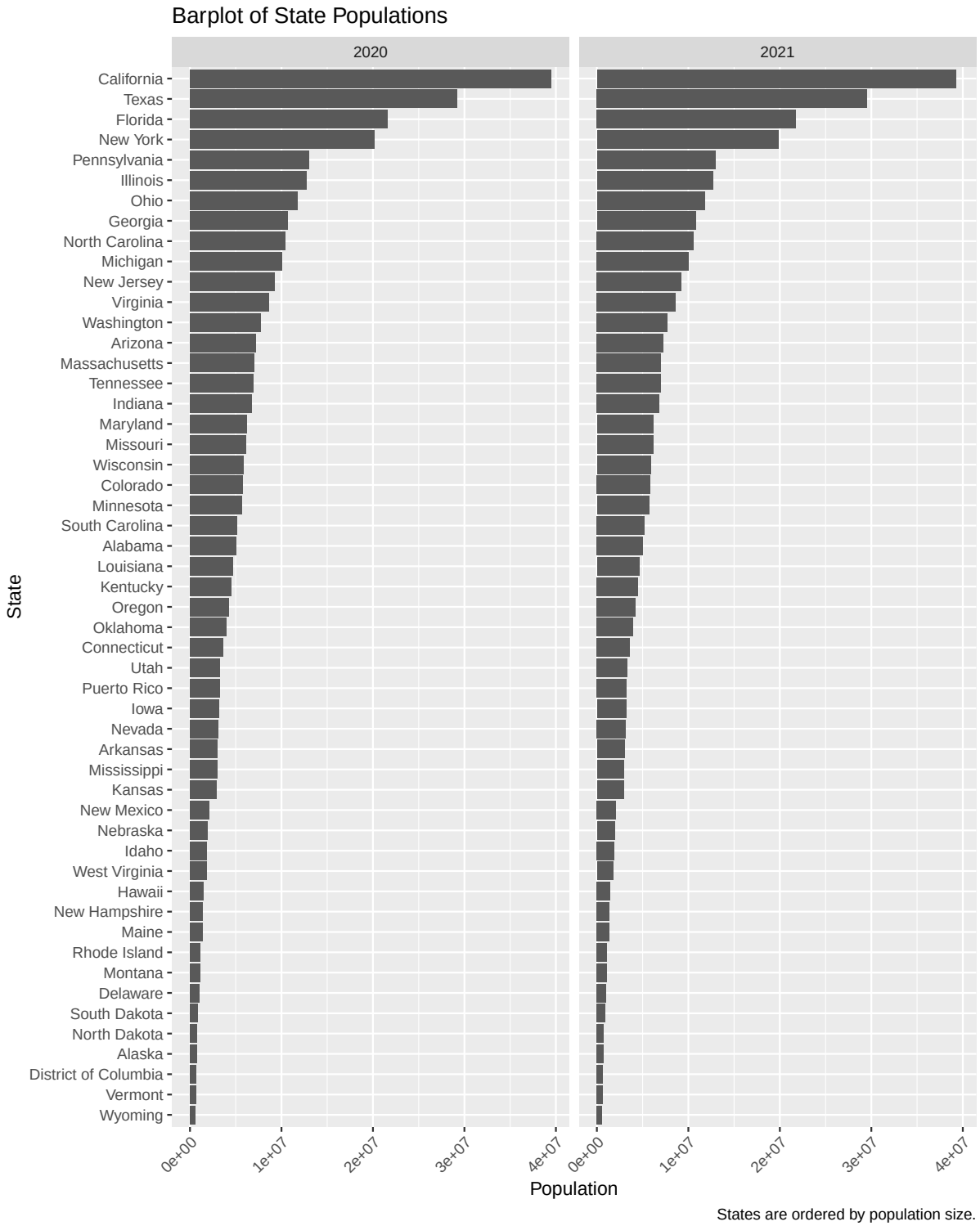
6. Examine the `population` matrix you just created. Notice that 1) it is not tidy, 2) the column types are not what we want, and 3) the first row is a header. Convert `population` to a tidy dataset. Remove the state ID column and change the name of the column with state names to `state_name`. Add a 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
  as_tibble() |>
  # make the first row the header
  row_to_names(row_number=1) |>
  # remove state ID column
  select(-state) |>
  # rename state column to state_name
  # remove POP_ from year
  rename("state_name"="NAME", "2020"="POP_2020","2021"="POP_2021") |>
  # add state abbreviations using state.abb variable
  mutate(state=state.abb[match(state_name, state.name)]) |>
  # 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",
                         .default = state)) |>
  # use pivot_longer to tidy
  pivot_longer(cols=c("2020","2021"),
               names_to="year",
               values_to="population") |>
  # parse all relevant columns to numeric
  mutate(year=as.numeric(year), population=as.numeric(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`.

```r
population |>
  # reorder state
  # assign aesthetic mapping
  ggplot(aes(x=reorder(state_name, population), y=population)) +
```

```
  # use geom_col to plot barplot
  geom_col() +
  # flip coordinates
  coord_flip() +
  # facet by year
  facet_wrap(~year) +
  # relabel plot
  ggtitle("Barplot of State Populations") +
  xlab("State") +
  ylab("Population") +
  labs(caption="States are ordered by population size.") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1))
```


Barplot of State Populations
2020
2021
California
Texas
Florida
New York
Pennsylvania
Illinois
Ohio
Georgia
North Carolina
Michigan
New Jersey
Virginia
Washington
Arizona
Massachusetts
Tennessee
Indiana
Maryland
Missouri
Wisconsin
Colorado
Minnesota
South Carolina
Alabama
Louisiana
Kentucky
Oregon
Oklahoma
Connecticut
Utah
Puerto Rico
Iowa
Nevada
Arkansas
Mississippi
Kansas
New Mexico
Nebraska
Idaho
West Virginia
Hawaii
New Hampshire
Maine
Rhode Island
Montana
Delaware
South Dakota
North Dakota
Alaska
District of Columbia
Vermont
Wyoming
State
0e+00
1e+07
2e+07
3e+07
4e+07
Population
States are ordered by population size.

8. The following URL:

```
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.

```
library(jsonlite)
library(purrr)
url <- "https://github.com/datasciencelabs/2025/raw/refs/heads/main/data/regions.json"
# regions <- use jsonlit JSON parser
# regions <- convert list to data frame. You can use map_df in purrr package
regions <- fromJSON(url) |>
  pmap_df(function(region, region_name, states) {
    tibble(region=region,
           region_name=region_name,
           state_name=states)
  } ) |>
  # shorten long region name
  mutate(region_name=
    case_when(
    region_name=="New York and New Jersey, Puerto Rico, Virgin Islands" ~
      "NY, NJ, Territories",
    .default=region_name)) |>
  filter(state_name %in% c(state.name, "District of Columbia", "Puerto Rico"))
```

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

```
# combine with population dataframe
population <- left_join(population, regions, by="state_name")
```

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"
response <- request(api) |>
  req_perform()
if (response$status_code == 200) {
```

```
  cases_raw <- response |> resp_body_json(simplifyVector=TRUE)
} else {
  stop(paste("Unsuccessful status code in response:", response))
}
```

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

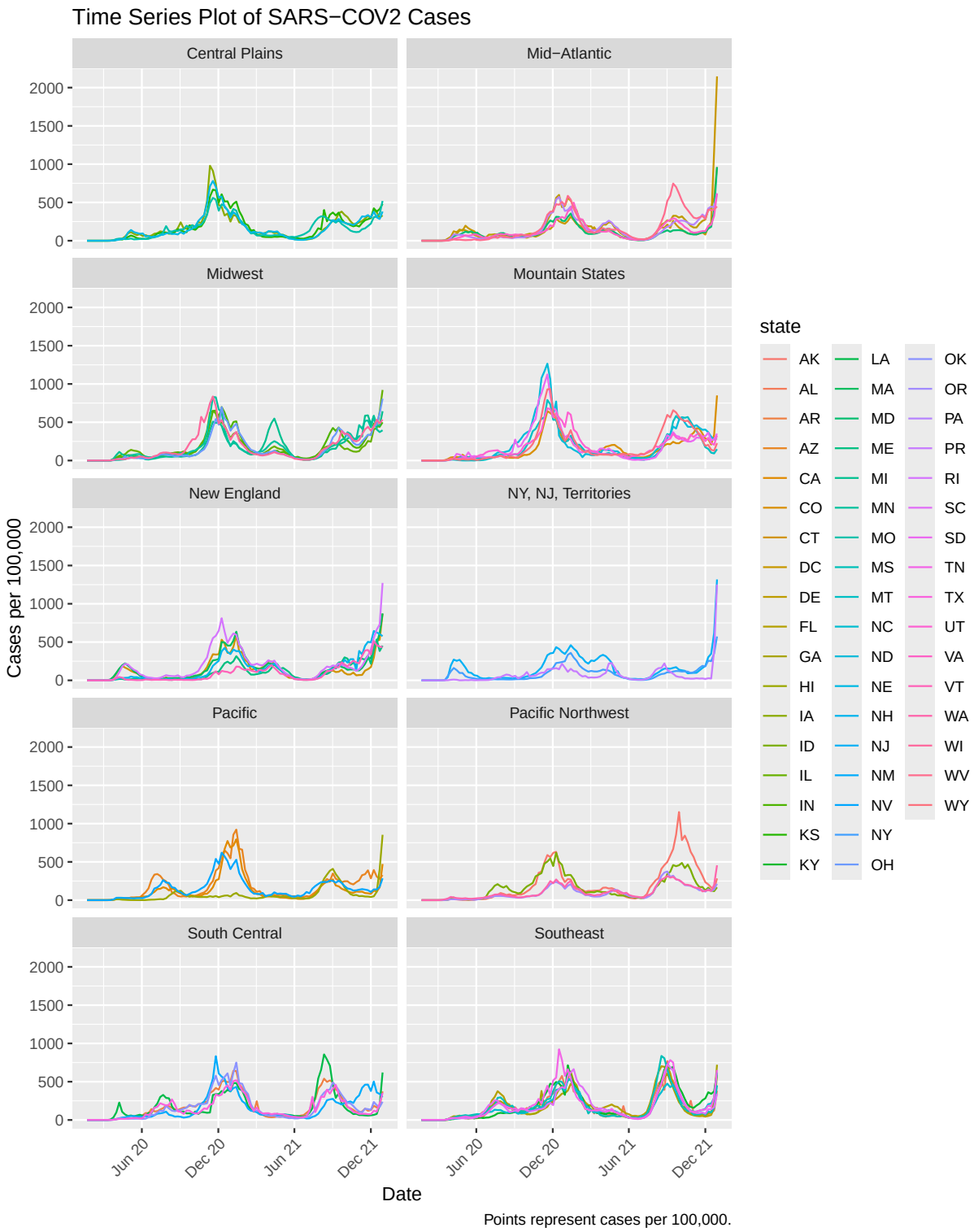
11. The reason you see exactly 1,000 rows is because CDC has a default limit. You can change this limit by adding `$limit=10000000000` to the request. Rewrite the previous request to ensure that you receive all the data. Then wrangle the resulting data frame to produce a data frame with columns `state`, `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) |>
  # add limit to request
  req_url_query(`$limit`=10000000000) |>
  # perform request
  req_perform() |>
  # get body from request
  resp_body_json(simplifyVector = TRUE) |>
  # limit to three columns
  select(state, date=end_date, cases=new_cases) |>
  # format: ISO-8601 dates and numeric cases
  mutate(date = as.Date(date), cases = as.numeric(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_raw |>
  # filter for 2020 and 2021
  filter(year(date) == "2020" | year(date) == "2021") |>
  mutate(year = year(date)) |>
  # join with population and regions tables
  inner_join(population,
             by=join_by(state == state, year == year)) |>
  # calculate cases per 100,000
  mutate(cases100k=cases / population * 100000) |>
  ggplot(aes(x=date, y=cases100k, group=state, color=state)) +
  geom_line() +
  # scale axes
  scale_x_date(date_breaks="6 months", date_labels="%b %y") +
```

```
  scale_y_continuous() +
  # stratify by region name
  facet_wrap(~region_name, ncol=2) +
  # label plot
  labs(title="Time Series Plot of SARS-COV2 Cases",
       caption="Points represent cases per 100,000.") +
  xlab("Date") +
  ylab("Cases per 100,000") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1))
```


Time Series Plot of SARS−COV2 Cases
Central Plains
Mid−Atlantic
Midwest
Mountain States
New England
NY, NJ, Territories
Pacific
Pacific Northwest
South Central
Southeast
Cases per 100,000
0
500
1000
1500
2000
Jun 20
Dec 20
Jun 21
Dec 21
Date
Points represent cases per 100,000.
state
AK
AL
AR
AZ
CA
CO
CT
DC
DE
FL
GA
HI
IA
ID
IL
IN
KS
KY
LA
MA
MD
ME
MI
MN
MO
MS
MT
NC
ND
NE
NH
NJ
NM
NV
NY
OH
OK
OR
PA
PR
RI
SC
SD
TN
TX
UT
VA
VT
WA
WI
WV
WY

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)
```

```
Warning: package 'knitr' was built under R version 4.3.3
```

```
cases_raw |>
  # use lubridate to parse date column
  mutate(month=month(date, label=TRUE), year=year(date)) |>
  # filter for 2020 and 2021
  filter(year=="2020" | year=="2021") |>
  # compute total cases across all states
  group_by(month, year) |>
  summarize(total_cases=sum(cases),
            .groups="drop_last") |>
  # order by month and year
  arrange(year, month) |>
  kable()
```

month	year	total_cases
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

month	year	total_cases
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.

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

if (response$status_code == 200) {
  deaths <- response |>
    resp_body_json(simplifyVector = TRUE) |>
    select(state,
           date=end_date,
           deaths=covid_19_deaths) |>
    mutate(date=as.Date(date), deaths=as.numeric(deaths)) |>
    filter(state %in% c(state.name, "District of Columbia", "Puerto Rico")) |>
    drop_na()
} else {
  print(paste("Request failed with response: ", response))
}
```

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 |>
  # calculate total deaths per state
  group_by(state) |>
  summarize(total_deaths=sum(deaths)) |>
  arrange(desc(total_deaths)) |>
  head(10) |>
  # order states by deaths
  ggplot(aes(x=reorder(state, desc(total_deaths)), y=total_deaths)) +
  geom_col() +
  # label graph
  labs(title="COVID-19 Deaths by State",
       caption="States with top 10 highest total COVID-19 deaths.") +
  xlab("State") +
  ylab("Total COVID-19 Deaths") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1))
```