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.

```
library(httr2)
request <- request(url) |>
 ↪  req_url_query(get=I("POP_2020,POP_2021,NAME"),
                                          `for`="state:*",
                                          key = census_key)
request$url # check match
```

```
[1] "https://api.census.gov/data/2021/pep/population?get=POP_2020,POP_2021,NAME&for=state%3A
```

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() # perform request
response # check output
```

```
<httr2_response>
```

```
GET
https://api.census.gov/data/2021/pep/population?get=POP_2020,POP_2021,NAME&for=state%3A%2A&k
```

```
Status: 200 OK
```

```
Content-Type: application/json
```

```
Body: In memory (2112 bytes)
```

```
resp_status(response) # check status - 200 is good
```

```
[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.

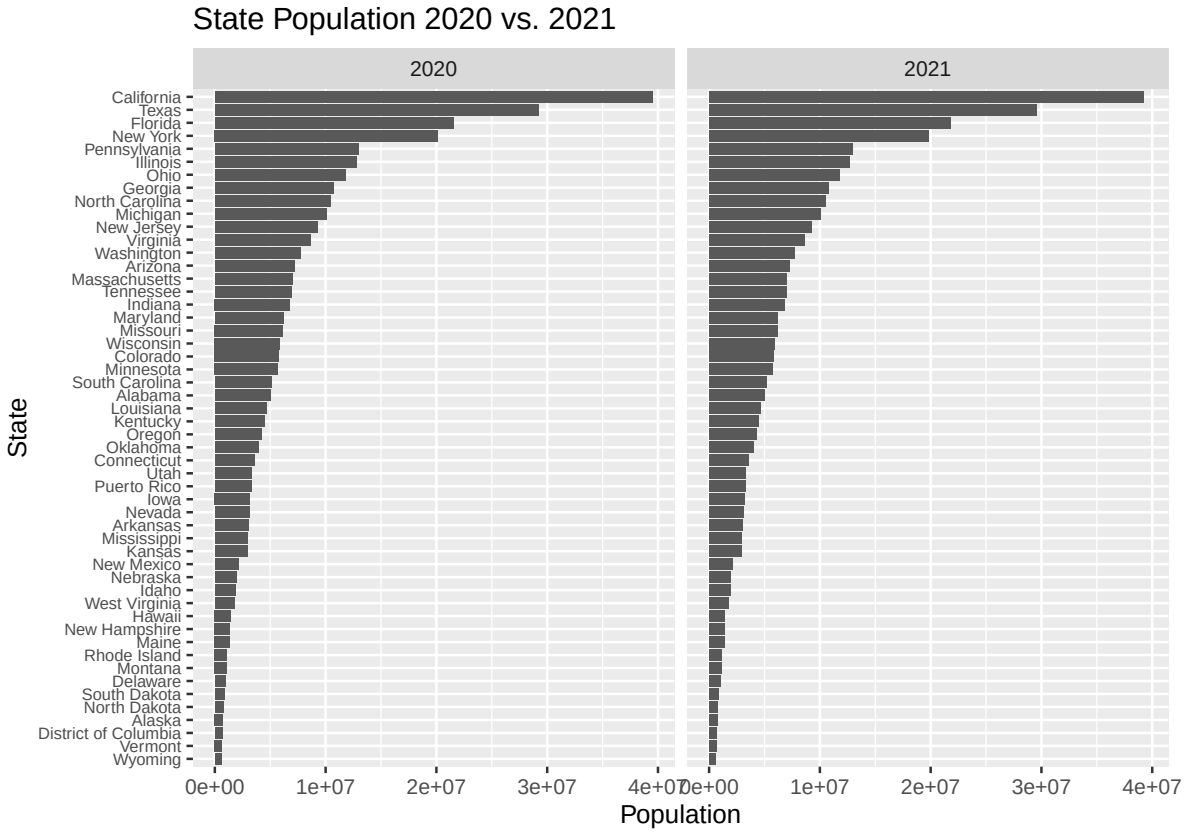
```
population <- resp_body_json(response, simplifyVector = TRUE)
```

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.

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

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(y=reorder(state_name, pop), x=pop)) +
  geom_col() + facet_wrap("year") + labs(x="Population",y="State",
   ↪  title="State Population 2020 vs. 2021") +
  theme(axis.text.y = element_text(size = 7)) # make state names smaller
   ↪  to fit
```



8. The following URL:

```
url2 <- "https://github.com/datasciencelabs/2025/raw/refs/heads/main/da
 ↪  ta/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)
url2 <- "https://github.com/datasciencelabs/2025/raw/refs/heads/main/da
 ↪  ta/regions.json"
regions <-  fromJSON(url2, simplifyVector = TRUE) #use jsonlit JSON
 ↪  parser
regions$region_name[regions$region_name == "New York and New Jersey,
 ↪  Puerto Rico, Virgin Islands"] = "NYNJ/Islands" # shorten region name
regions <-  map_df(seq_len(nrow(regions)), ~ { #for 1-n, where n is
 ↪  number of rows in df
  tibble( # create tibble with
    region = regions$region[.x], # region column as the region of row n
    region_name = regions$region_name[.x], # region name column as the
     ↪  region name of row n
    state_name= regions$states[[.x]] # state name column as each value
     ↪  within the states list
  )
}) |> #convert list to data frame. You can use map_df in purrr package
  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.

```
population <- left_join(population, regions, by="state_name") # match
 ↪  rows of regions to rows of population by state name column
```

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"
casereq <- api|> request() |> req_perform()
resp_check_status(casereq)
resp_check_content_type(casereq)
cases_raw <- casereq |> resp_body_json(simplifyVector = TRUE)
```

We see exactly 1,000 rows. We should be seeing over 52×3 rows per state. - This may be due to a limitation from the API on the number of rows that can be pulled at a time.

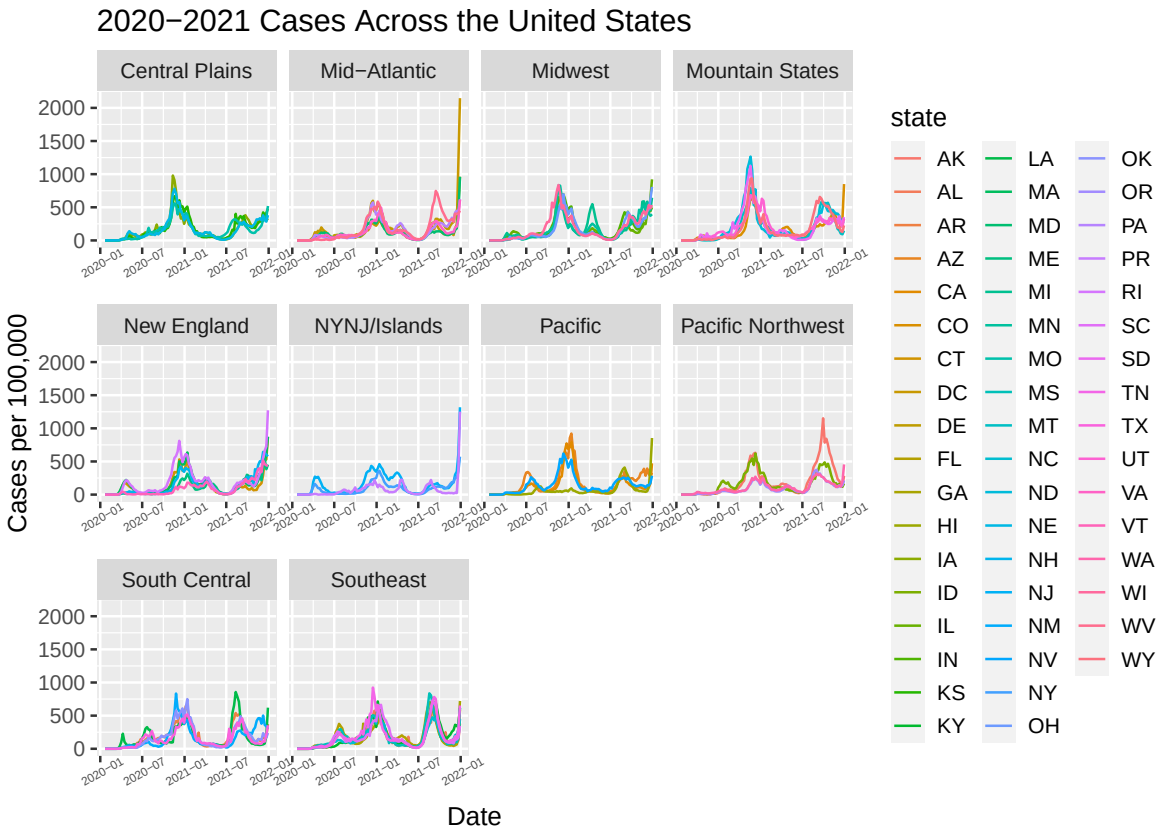
11. The reason you see exactly 1,000 rows is because CDC has a default limit. You can change this limit by adding `$limit=10000000000` to the request. Rewrite the previous

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"
casereq2 <- api|> request() |> req_url_query(`$limit` = 10000000000) |>
 ↪  req_perform()
resp_check_status(casereq2)
resp_check_content_type(casereq2)
cases_raw2 <- casereq2 |> resp_body_json(simplifyVector = TRUE) # redo
 ↪  with limit
Cases = cases_raw2 |> select(state,date=end_date,cases=new_cases) |> #
 ↪  choose relevant columns from dataset
  mutate(cases=as.numeric(cases),date=as.Date(date)) |> # fix type
  filter(state %in% c(state.abb, "DC", "PR"))
```

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_pop_20_21 = Cases |> filter(date>"2019-12-31" & date < "2022-1-1")
 ↪  |> # only 2020 and 2021 data
  mutate(date_to_year = as.numeric(format(date,"%Y"))) |> # get year
   ↪  from date to match population df
  left_join(y = population, by=c("state"="state_abb", "date_to_year" =
   ↪  "year")) |> # match population rows to case rows by state
   ↪  abbreviation and year
  na.omit() # remove rows with null values
cases_pop_20_21 |> ggplot(aes(x=date,y=100000*cases/pop, col=state)) +
  geom_line() + facet_wrap(~region_name, scales="free_x") + # region
   ↪  strata, give each plot x axis labels
  labs(x="Date", y="Cases per 100,000", title="2020-2021 Cases Across
   ↪  the United States") + # labels and title
  theme(legend.key.size = unit(0.5, "cm"), axis.text.x =
   ↪  element_text(size = 5, angle=30)) # fix spacing and label sizing
```



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)
Cases |> mutate(date = ymd(as.character(date)), year = year(date), # ymd
 ↪  from lubridate to parse date column, split into year and month
 ↪  columns
                month = month(date, label = TRUE, abbr = TRUE)) |>
  group_by(year, month) |> summarise(total_cases =
   ↪  sum(cases,na.rm=TRUE)) |> kable() # select only year and month
   ↪  data and combine for total cases
```

```
`summarise()` has grouped output by 'year'. You can override using the
```

`.groups` argument.

year	month	total_cases
2020	Jan	11
2020	Feb	68
2020	Mar	50335
2020	Apr	822648
2020	May	616691
2020	Jun	642552
2020	Jul	1977016
2020	Aug	1452393
2020	Sep	1401917
2020	Oct	1608932
2020	Nov	3887222
2020	Dec	6907540
2021	Jan	5649115
2021	Feb	2543964
2021	Mar	1928749
2021	Apr	1694189
2021	May	948953
2021	Jun	484817
2021	Jul	1120939
2021	Aug	3519407
2021	Sep	4960807
2021	Oct	2317854
2021	Nov	2289118
2021	Dec	5293391
2022	Jan	18871940
2022	Feb	5524778
2022	Mar	1202515
2022	Apr	1030259
2022	May	2422493
2022	Jun	3541044
2022	Jul	3337779
2022	Aug	3433327
2022	Sep	1663011
2022	Oct	1038235
2022	Nov	1382581
2022	Dec	1714778
2023	Jan	1467538
2023	Feb	1023913

year	month	total_cases
2023	Mar	866851
2023	Apr	436770
2023	May	156975

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.

```
deathsreq = deaths_url |> request() |> req_url_query(`$limit` =
 ↪  10000000000) |> req_perform() # increase limit to get full data
resp_check_status(deathsreq)
resp_check_content_type(deathsreq)
Deaths <- deathsreq |> resp_body_json(simplifyVector = TRUE) |>
  filter(sex=="All Sexes", age_group=="All Ages") |> # only use
   ↪  aggregate rows, not stratified rows
  select(state, date=end_date, deaths=covid_19_deaths) |> # choose
   ↪  relevant columns to keep
  mutate(date = as.Date(date), deaths=as.numeric(deaths)) |> # fix
   ↪  formatting
  filter(!is.na(deaths) & !is.na(date) & !is.na(state)) |> # remove
   ↪  nulls
  filter(state %in% c(state.name, "District of Columbia", "Puerto
   ↪  Rico"))
```

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.

```
Deaths |> group_by(state) |> summarise(sumdeaths=sum(deaths)) |> # total
 ↪  deaths by state
  arrange(-sumdeaths) |> slice(seq(1,10)) |> # sort high to low, and get
   ↪  top 10 (not including "United States")
  ggplot(aes(x=reorder(state, -sumdeaths), y=sumdeaths)) + geom_col() +
   ↪  # plot
```

```
labs(x="State", y="COVID-19 Death Count", title="Top 10 States by
 ↪  COVID-19 Deaths") # labels
```