

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")
census_key
```

```
[1] "1183799c5047834fa6fd975e4f3ee8c6f89b18f2"
```

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 an 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()
response$status_code
```

```
[1] 200
```

```
class(response)
```

```
[1] "httr2_response"
```

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.

```
population1<-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<-population1 |>
  row_to_names(row_number=1) |>
  as_tibble() |> # convert to tibble
  select(-state) |> # remove state column
  rename(state_name=NAME, # rename NAME column to state_name
         "2020"=POP_2020, # remove POP_ from year
         "2021"=POP_2021) |> # remove POP_ from year
  mutate(state = case_when( # use case_when to add abbreviations for DC
    <- and PR
    state_name == "District of Columbia" ~ "DC",
    state_name == "Puerto Rico" ~ "PR",
    TRUE ~ state.abb[match(state_name, state.name)] # add state abbreviations
    <- using state.abb variable
  )) |>
  pivot_longer( #create a year column.
    cols = c("2020", "2021"),
    names_to = "year",
    values_to = "population"
  ) |>
  arrange(state_name) |> #arrange states in alphabetical order
  mutate(population=as.numeric(population)) #make pop var numeric

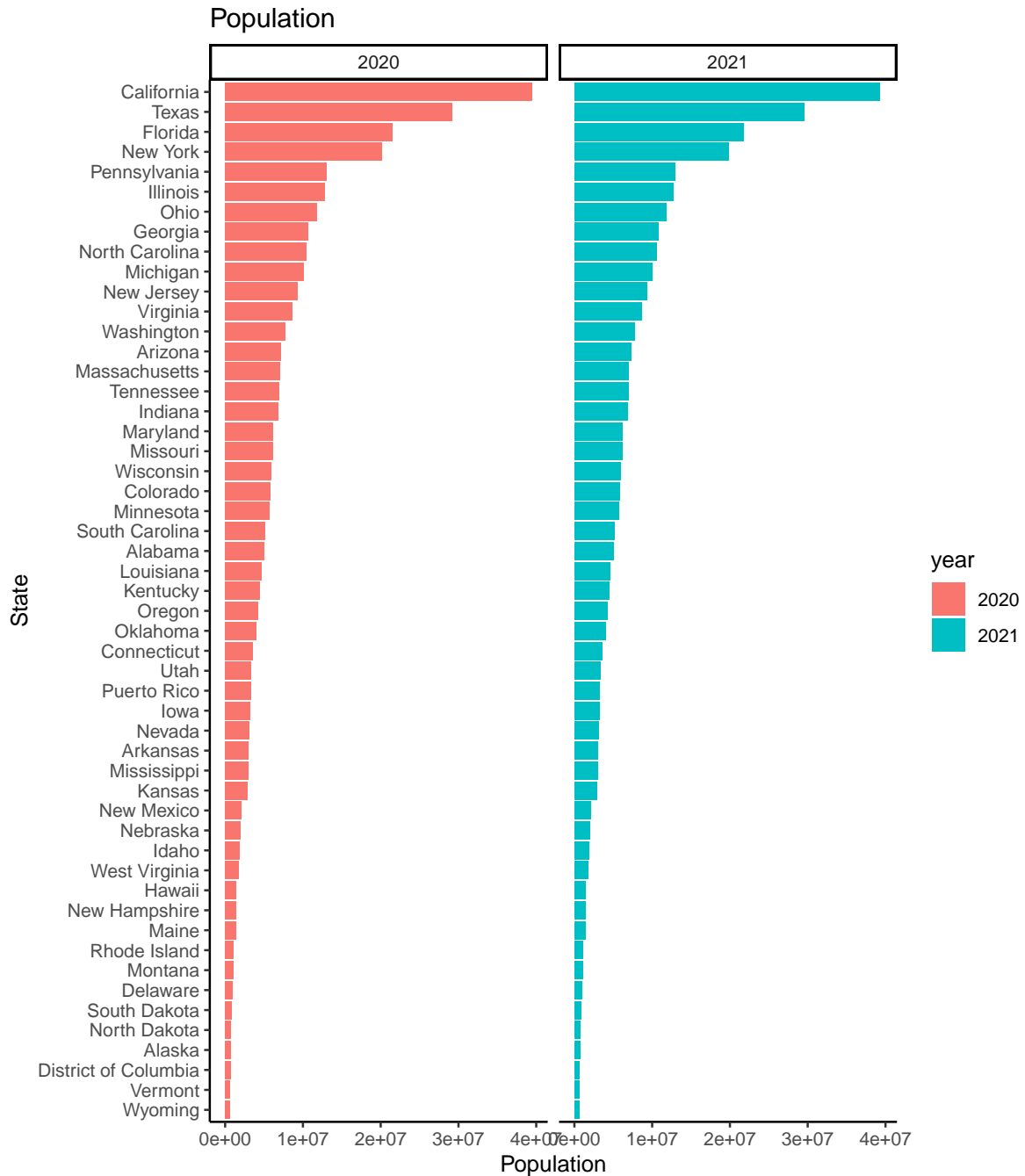
#population <- population |> ## Use janitor row to names function
# convert to tibble
# remove stat column
# rename state column to state_name
# use pivot_longer to tidy
# remove POP_ from year
```

```
# parse all relevant columns 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**.

```
library(ggplot2)

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



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 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/reg_
  ↪ ions.json"
# regions <- use jsonlit JSON parser
# regions <- convert list to data frame. You can use map_df in purrr package

regions<-fromJSON(url, simplifyVector = FALSE) |>
  map_df(function(x) {
    tibble(
      region = x[[ "region" ]],
      region_name = x[[ "region_name" ]],
      state_name = x[[ "states" ]]
    }) |>
  unnest(state_name) |>
  mutate(region_name= ifelse(region_name == "New York and New Jersey, Puerto
    ↪ Rico, Virgin Islands", "NY/NJ/PR/VI", region_name))
```

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

```
population <- population |>
  left_join(regions)
```

Joining with ``by = join_by(state_name)``

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"

response.10 <- request(api) |> req_perform() |> resp_body_json(simplifyVector
  ↪ = TRUE)

nrow(response.10)
```

[1] 1000

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

All of the data is not there. This is because the API limits the number of entries that we are able to download. In this case, the limit is only 1000 entries.

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.

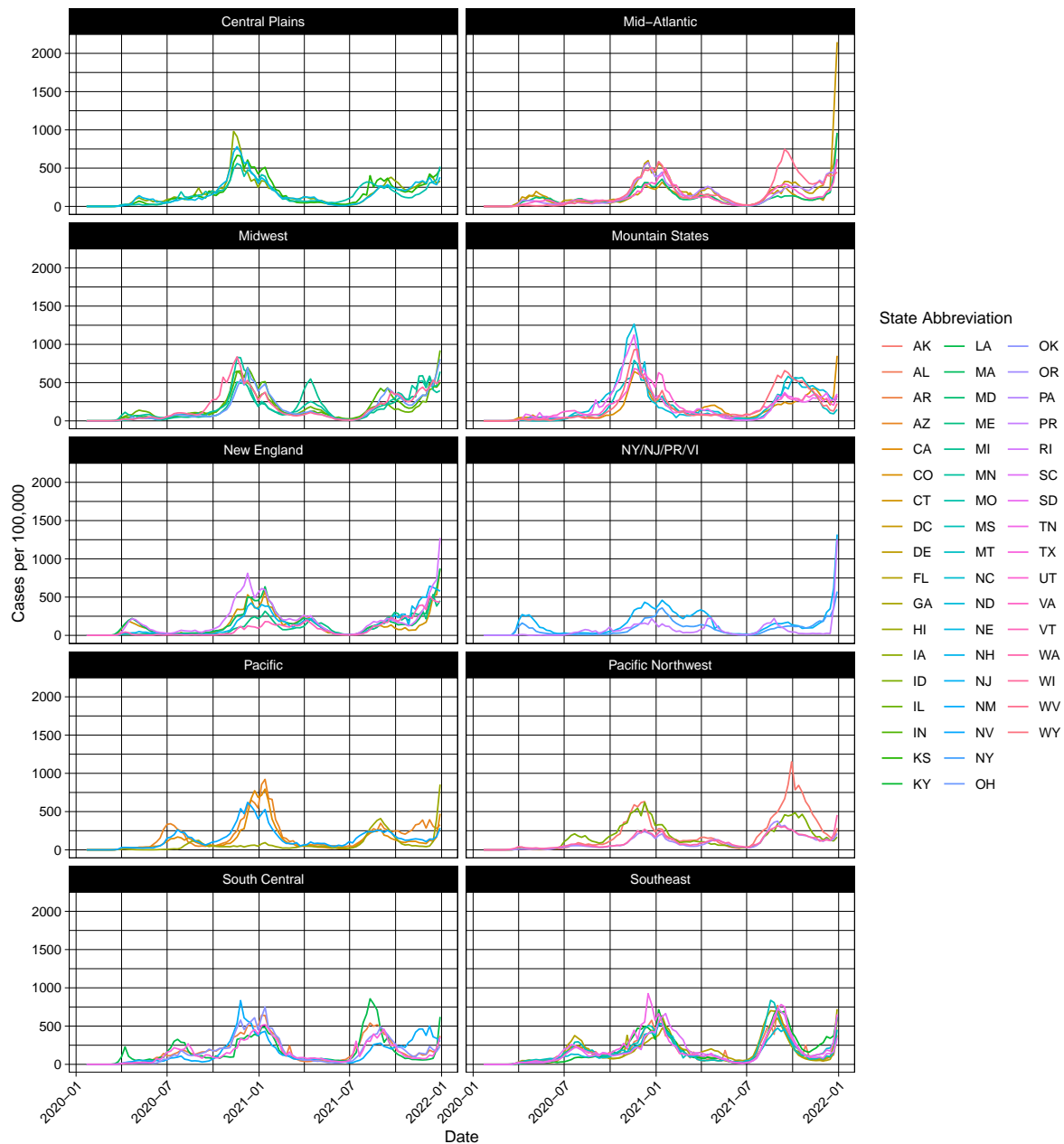
```
api <- "https://data.cdc.gov/resource/pwn4-m3yp.json"

cases_raw <- request(api) |> req_url_query(`$limit` = 10000000000) |>
  req_perform() |>
  resp_body_json(simplifyVector = TRUE) |>
  select(state, end_date, new_cases) |>
  rename(date=end_date, cases=new_cases) |>
  mutate(cases=as.numeric(cases),
         date=as.Date(date))
```

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 |>
  mutate(year = as.character(year(date))) |> #
  filter(year %in% c("2020", "2021")) |>
  right_join(population, by = c("state", "year")) |>
  mutate(cases_per=100000*(cases/population)) |>
  ggplot(aes(x=date, y=cases_per, color=state)) +
  geom_line() +
  labs(title = "Time Series Plot for Covid-19 Cases per 100,000", y = "Cases
    ↪ per 100,000", x="Date", color = "State Abbreviation")+
  theme_linedraw()+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))+
  facet_wrap(~region_name, nrow=5)
```

Time Series Plot for Covid-19 Cases per 100,000



- 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_raw |>
  #I already changed date from a character string in an earlier step
  mutate(year = year(date),
         month = month(date, label=TRUE)) |>
  group_by(year, month)|>
  summarise(Total_Covid_Cases=sum(cases)) |>
  arrange(year, month) |>
  filter(year %in% c(2020, 2021)) |>
  kable(col.names = c("Year", "Month", "Total Covid Cases"))
```

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

Year	Month	Total Covid Cases
2020	Jan	11
2020	Feb	68
2020	Mar	68245
2020	Apr	974032
2020	May	650943
2020	Jun	654904
2020	Jul	1989512
2020	Aug	1461283
2020	Sep	1415438
2020	Oct	1628598
2020	Nov	3932646
2020	Dec	7027128
2021	Jan	5808063
2021	Feb	2667511
2021	Mar	2068441
2021	Apr	1773591
2021	May	972915
2021	Jun	493635
2021	Jul	1137440
2021	Aug	3572562
2021	Sep	5027537
2021	Oct	2356302
2021	Nov	2322814
2021	Dec	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.

```
deaths<-request(deaths_url) |> req_url_query(`$limit` = 10000000000) |>
  req_perform() |>
  resp_body_json(simplifyVector = TRUE) |>
  select(state, covid_19_deaths, end_date) |>
  rename(deaths=covid_19_deaths, date=end_date) |>
  mutate(deaths=as.numeric(deaths),
         date=as.Date(date))
```

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 |>
  filter(state != "United States") |>
  group_by(state) |>
  filter(state!="New York City") |>
  summarize(total=sum(deaths, na.rm=TRUE)) |>
  arrange(desc(total)) |>
  head(10) |>
  ggplot(aes(x=reorder(state, -total), y=total, fill=state))+
  geom_col(show.legend = FALSE) +
  labs(title = "Top 10 Highest Covid Deaths by State", y = "Total Deaths
    ↪ (2020-2023)", x="State")+
  theme_linedraw()+
  theme(axis.title = element_text(hjust = 0.5, face = "bold"), plot.title =
    ↪ element_text(hjust = 0.5, face = "bold"))
```

