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`.

```r
# runs different R file and makes its variables available in current file
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:

```r
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 = "POP_2020,POP_2021,NAME", # attributes to GET
    `for` = "state:*", # for is required by the API
    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 <- req_perform(request)
response |> resp_status_desc() # checking response status
```

```
[1] "OK"
```

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

```r
response |> resp_content_type() # checking response data 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, simplifyVector = TRUE)
head(population)
```

```
     [,1]       [,2]       [,3]           [,4]   
[1,] "POP_2020" "POP_2021" "NAME"         "state"
[2,] "3962031"  "3986639"  "Oklahoma"     "40"   
[3,] "1961455"  "1963692"  "Nebraska"     "31"   
[4,] "1451911"  "1441553"  "Hawaii"       "15"   
[5,] "887099"   "895376"   "South Dakota" "46"   
[6,] "6920119"  "6975218"  "Tennessee"    "47"   
```

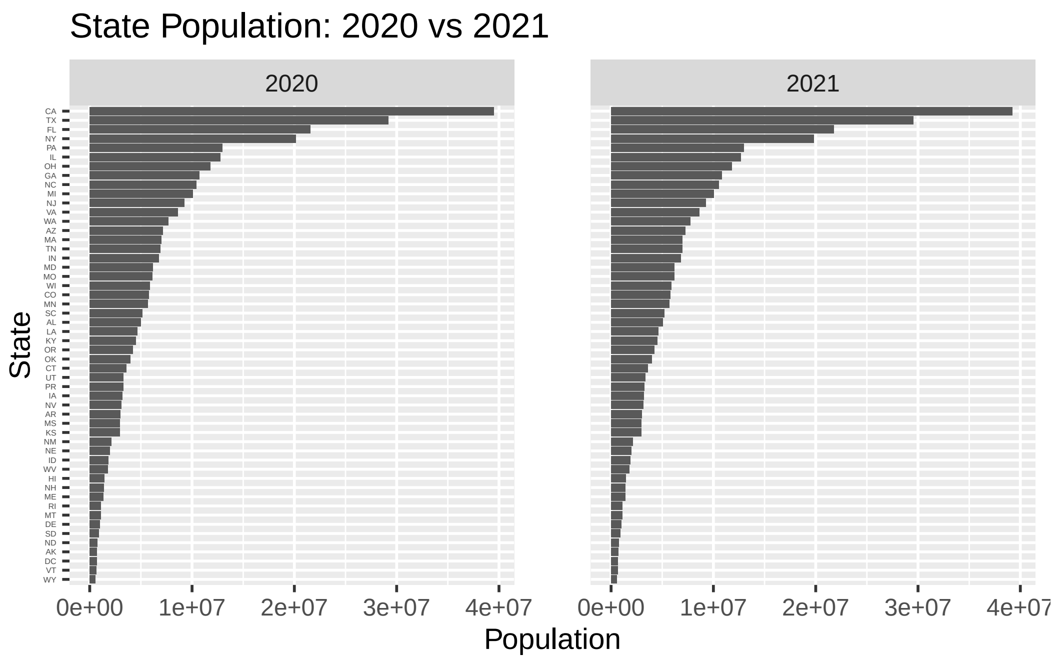
6. Examine the `population` matrix you just created. Notice that 1) it is not tidy, 2) the column types are not what we want, and 3) the first row is a header. Convert `population` to a tidy dataset. Remove the state ID column and change the name of the column with state names to `state_name`. Add a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

population
```

```
# A tibble: 104 x 4
   state_name    year state_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 South Dakota  2020           887099 SD
 8 South Dakota  2021           895376 SD
 9 Tennessee     2020            6920119 TN
10 Tennessee     2021            6975218 TN
# i 94 more rows
```

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 |>
  mutate(state = reorder(state, state_population, FUN = mean)) |> # reorder
   ↪  state
  ggplot(aes(state, state_population)) + # assign aesthetic mapping
    geom_col() + # use geom_col to plot barplot
    coord_flip() + # flip coordinates
    facet_wrap(~year) + # facet by year
    labs(
      title = "State Population: 2020 vs 2021",
      x = "State",
      y = "Population"
    ) +
    theme(
      axis.text.y = element_text(size = 3), # make state names smaller so
       ↪  they don't overlap
      panel.spacing = unit(2, "lines") # put more space between the graphs
      )
```



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"

response <- request(url) |>
  req_perform()

# check response status
if (resp_status(response) == 200) {
  response |>
  resp_body_string() |>
  fromJSON() -> regions # use jsonlit JSON parser
} else {
  stop(paste("request failed with, status:", resp_status(resp)))
}


regions <- regions |>
  unnest(states) |> # creates row for each state
  mutate(region = map_int(region, 1)) |> # pulls out region number from list
   ↪ of length 1
  select(state_name = states, region = region, region_name = region_name) #
   ↪ reordering and renaming

regions$region_name[regions$region_name == "New York and New Jersey, Puerto
 ↪ Rico, Virgin Islands"] <- "NY & NJ, PR, VI"
head(regions, 20)
```

```
# A tibble: 20 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
 7 New Jersey                2 NY & NJ, PR, VI
 8 New York                  2 NY & NJ, PR, VI
 9 Puerto Rico               2 NY & NJ, PR, VI
10 Virgin Islands            2 NY & NJ, PR, VI
11 Delaware                  3 Mid-Atlantic
12 District of Columbia      3 Mid-Atlantic
13 Maryland                  3 Mid-Atlantic
14 Pennsylvania              3 Mid-Atlantic
15 Virginia                  3 Mid-Atlantic
16 West Virginia             3 Mid-Atlantic
17 Alabama                   4 Southeast
18 Florida                   4 Southeast
19 Georgia                   4 Southeast
20 Kentucky                  4 Southeast
```

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

```r
population <- population |>
  left_join(regions, by = "state_name")

head(population, 10)
```

```
# A tibble: 10 x 6
   state_name    year state_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
 7 South Dakota  2020           887099 SD         8 Mountain States
 8 South Dakota  2021           895376 SD         8 Mountain States
 9 Tennessee     2020          6920119 TN         4 Southeast
10 Tennessee     2021          6975218 TN         4 Southeast
```

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"

response <- request(api) |>
  req_perform()

# check response status
if (resp_status(response) == 200) {
  response |>
  resp_body_json(simplifyVector = TRUE) -> cases_raw
} else {
  stop(paste("request failed with, status:", resp_status(resp)))
}

head(cases_raw, 10)
```

```
             date_updated state              start_date
1 2023-02-23T00:00:00.000    AZ 2023-02-16T00:00:00.000
2 2022-12-22T00:00:00.000    LA 2022-12-15T00:00:00.000
```

```
3  2023-02-23T00:00:00.000    GA 2023-02-16T00:00:00.000
4  2023-03-30T00:00:00.000    LA 2023-03-23T00:00:00.000
5  2023-02-02T00:00:00.000    LA 2023-01-26T00:00:00.000
6  2023-03-23T00:00:00.000    LA 2023-03-16T00:00:00.000
7  2023-04-27T00:00:00.000    LA 2023-04-20T00:00:00.000
8  2023-03-16T00:00:00.000    NV 2023-03-09T00:00:00.000
9  2023-05-11T00:00:00.000    FL 2023-05-04T00:00:00.000
10 2022-10-27T00:00:00.000   NYC 2022-10-20T00:00:00.000
                  end_date tot_cases new_cases tot_deaths new_deaths
1  2023-02-22T00:00:00.000 2434631.0    3716.0    33042.0       39.0
2  2022-12-21T00:00:00.000 1507707.0    4041.0    18345.0       21.0
3  2023-02-22T00:00:00.000 3061141.0    5298.0    42324.0       88.0
4  2023-03-29T00:00:00.000 1588259.0    2203.0    18858.0       23.0
5  2023-02-01T00:00:00.000 1548508.0    5725.0    18572.0       47.0
6  2023-03-22T00:00:00.000 1580709.0    1961.0    18835.0       35.0
7  2023-04-26T00:00:00.000 1597070.0    1884.0    18937.0       25.0
8  2023-03-15T00:00:00.000  891702.0    1233.0    11937.0       15.0
9  2023-05-10T00:00:00.000 7572282.0    6937.0    88248.0        0.0
10 2022-10-26T00:00:00.000 2928439.0   14590.0    42863.0       91.0
   new_historic_cases new_historic_deaths
1               23150                   0
2               21397                   0
3                6800                   0
4                5347                   0
5                4507                   0
6                2239                   0
7                 949                   0
8                 347                   0
9                   0                1109
10                  0                 557
```

```r
dim(cases_raw)
```

```
[1] 1000   10
```

The data is not all there. There should be data for every state, for every week for 3 years. This table only has 1000 rows due to the default API limit.

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.

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

response <- request(api) |>
  req_url_query(
    `$limit` = 10000000000 # bypasses default limit
  ) |>
  req_perform()

# check response status
if (resp_status(response) == 200) {
  response |>
  resp_body_json(simplifyVector = TRUE) -> cases_raw
} else {
  stop(paste("request failed with, status:", resp_status(resp)))
}

cases <- cases_raw |>
  select(state = state, date = end_date, cases = new_cases) |> # reorder and
   ↪  rename columns
  mutate(cases = as.numeric(cases), date = as.Date(date)) # proper data types
   ↪

head(cases, 10)
```

```
   state       date cases
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
7     LA 2023-04-26  1884
8     NV 2023-03-15  1233
9     FL 2023-05-10  6937
10   NYC 2022-10-26 14590
```

```
dim(cases)
```

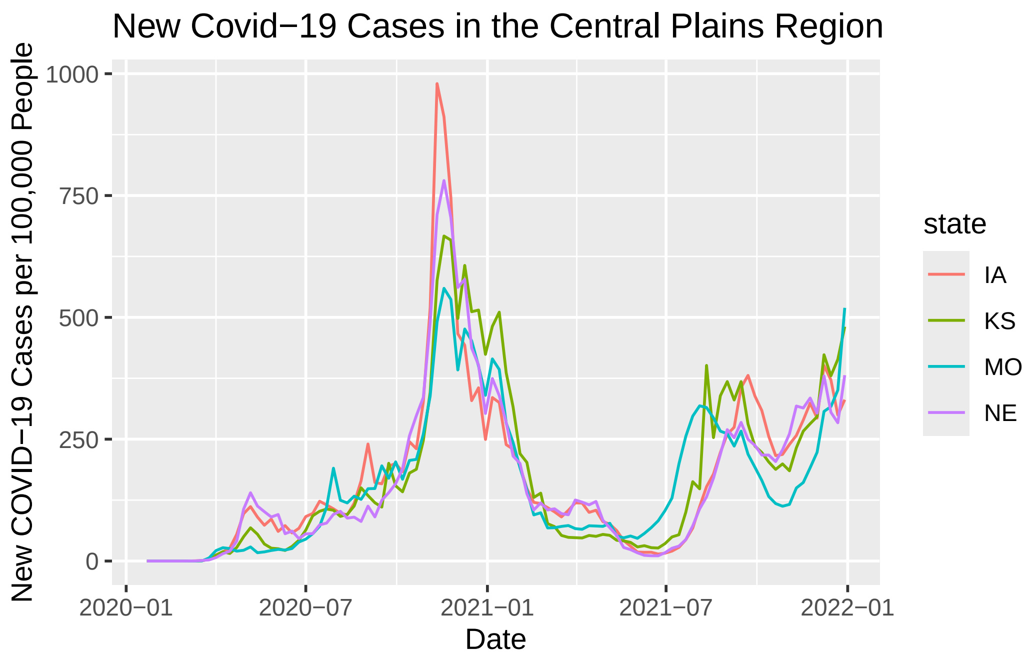
```
[1] 10380     3
```

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.

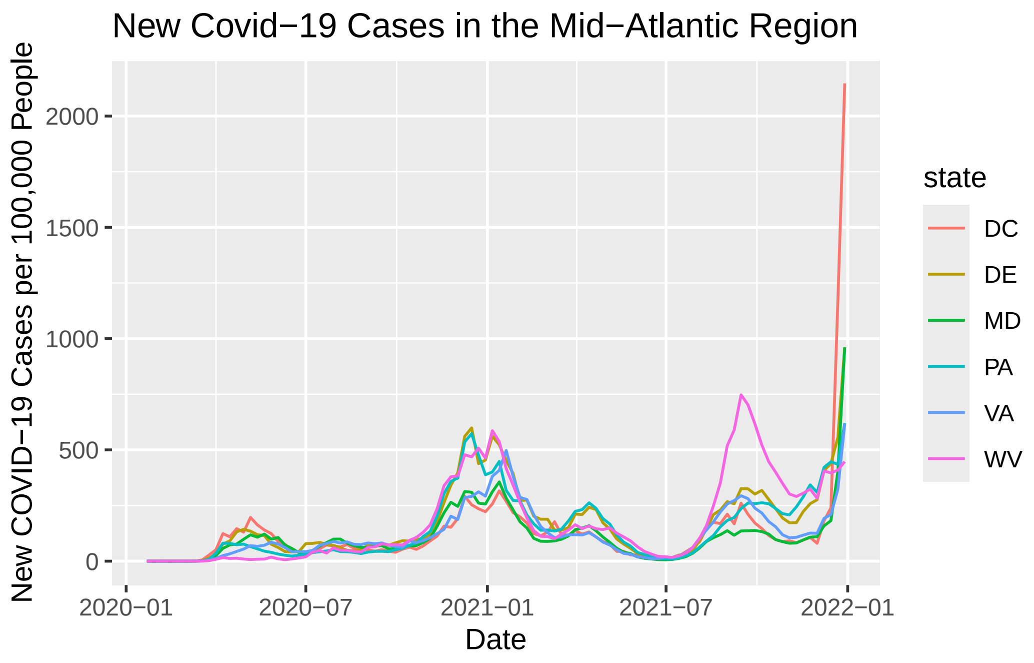
```
cases |>
  filter(date >= as.Date("2020-01-01") & date <= as.Date("2021-12-31")) |> #
   ↪ filter for 2020-2021
  left_join(population, by = "state") |>
  filter(year(date) == year) |> # removes duplicate rows, only keeps the row
   ↪ that has correct cases$date and population$year combo
  mutate(per_100000 = cases/state_population*100000) |> # makes new column of
   ↪ cases per 100,000

  group_split(region_name) |> # stratify by region name
  map(~ggplot(.x, aes(x=date, y=per_100000, colour = state)) +
    geom_line() +
    labs(
      title = paste("New Covid-19 Cases in the", unique(.x$region_name),
       ↪ "Region"),
      y = "New COVID-19 Cases per 100,000 People",
      x = "Date"
      ))
```

```
Warning in left_join(filter(cases, date >= as.Date("2020-01-01") & date <= : Detected an une
i Row 1 of `x` matches multiple rows in `y`.
i Row 103 of `y` matches multiple rows in `x`.
i If a many-to-many relationship is expected, set `relationship =
  "many-to-many"` to silence this warning.
```

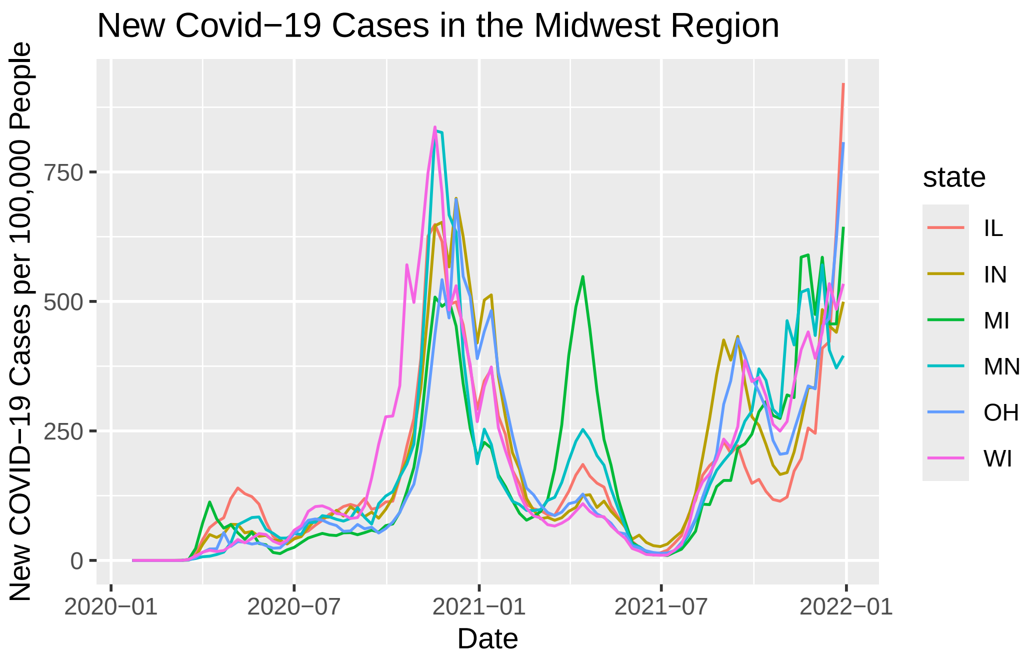
```
[[1]]
```


New Covid−19 Cases in the Central Plains Region
New COVID−19 Cases per 100,000 People
1000
750
500
250
0
2020−01
2020−07
2021−01
2021−07
2022−01
Date
state
IA
KS
MO
NE

```
[[2]]
```

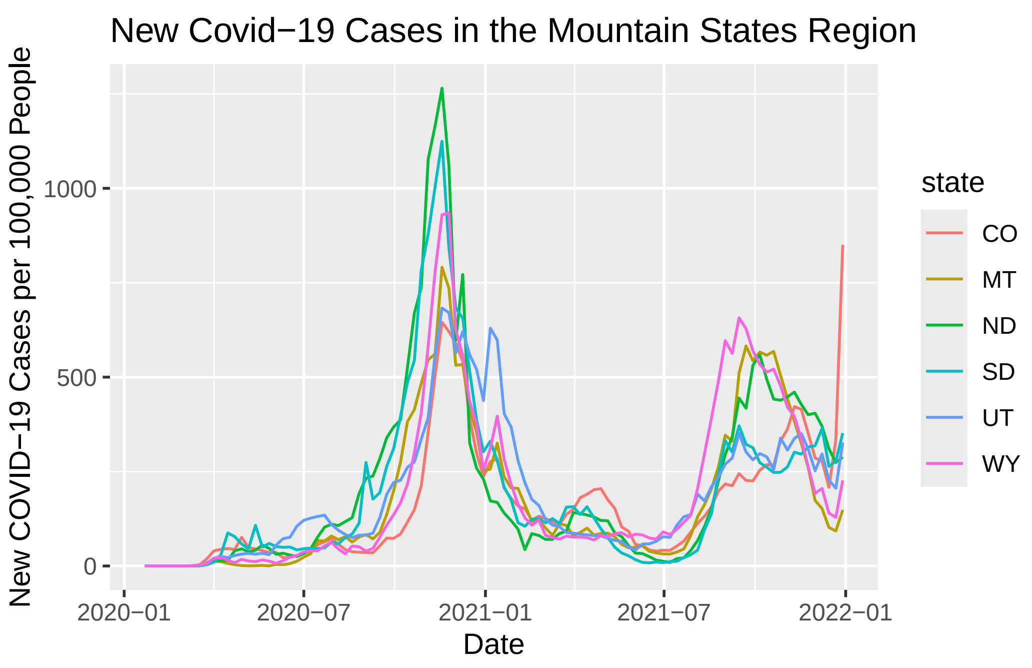

New Covid−19 Cases in the Mid−Atlantic Region
New COVID−19 Cases per 100,000 People
2000
1500
1000
500
0
2020−01
2020−07
2021−01
2021−07
2022−01
Date
state
DC
DE
MD
PA
VA
WV

```
[[3]]
```

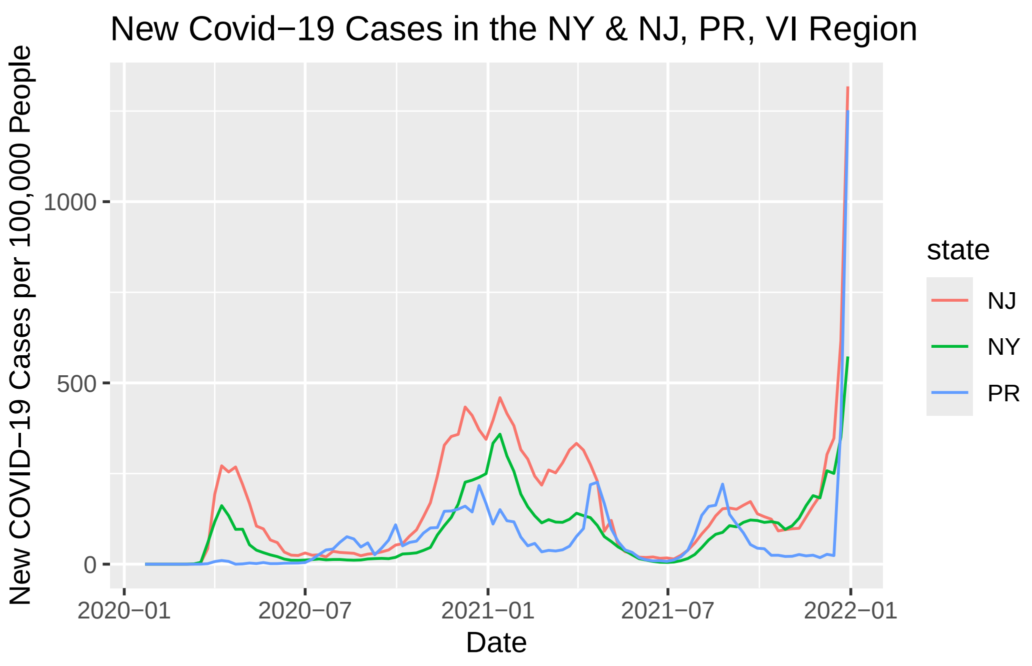


New Covid−19 Cases in the Midwest Region
New COVID−19 Cases per 100,000 People
0
250
500
750
2020−01
2020−07
2021−01
2021−07
2022−01
Date
state
IL
IN
MI
MN
OH
WI

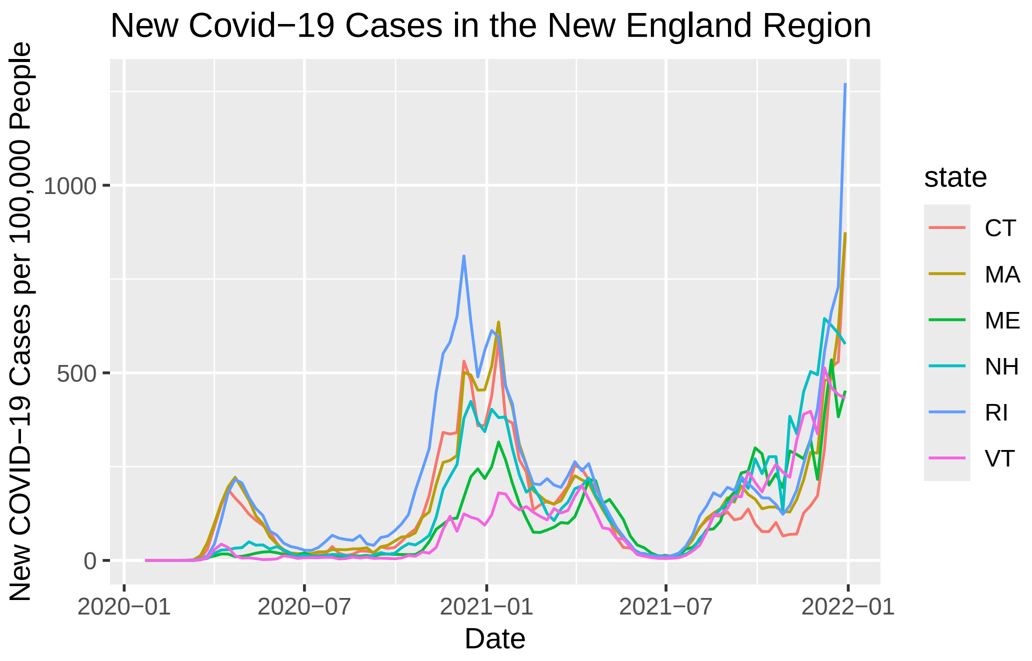
```
[[4]]
```


New Covid−19 Cases in the Mountain States Region
New COVID−19 Cases per 100,000 People
0
500
1000
2020−01
2020−07
2021−01
2021−07
2022−01
Date
state
CO
MT
ND
SD
UT
WY

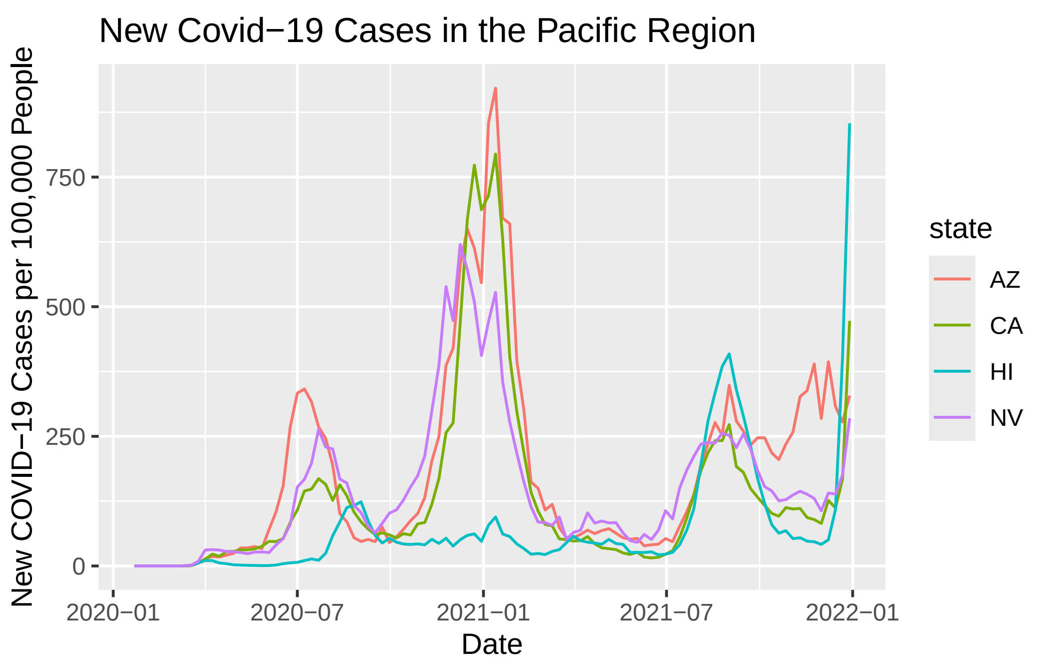
```
[[5]]
```


New Covid−19 Cases in the NY & NJ, PR, VI Region
New COVID−19 Cases per 100,000 People
0
500
1000
2020−01
2020−07
2021−01
2021−07
2022−01
Date
state
NJ
NY
PR

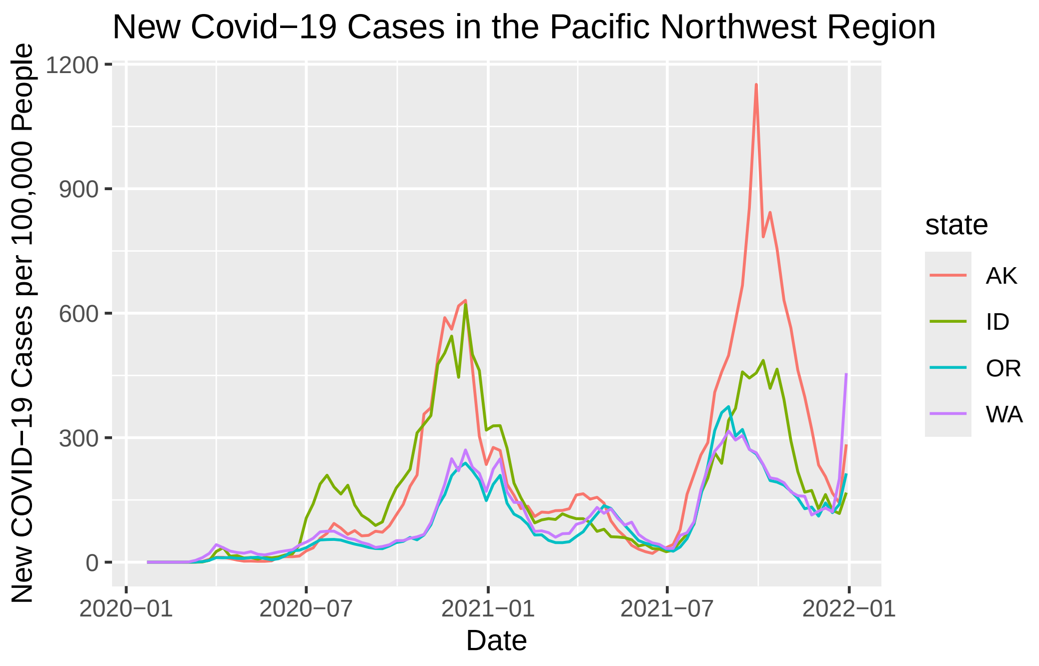
[[6]]


New Covid−19 Cases in the New England Region
New COVID−19 Cases per 100,000 People
1000
500
0
2020−01
2020−07
2021−01
2021−07
2022−01
Date
state
CT
MA
ME
NH
RI
VT

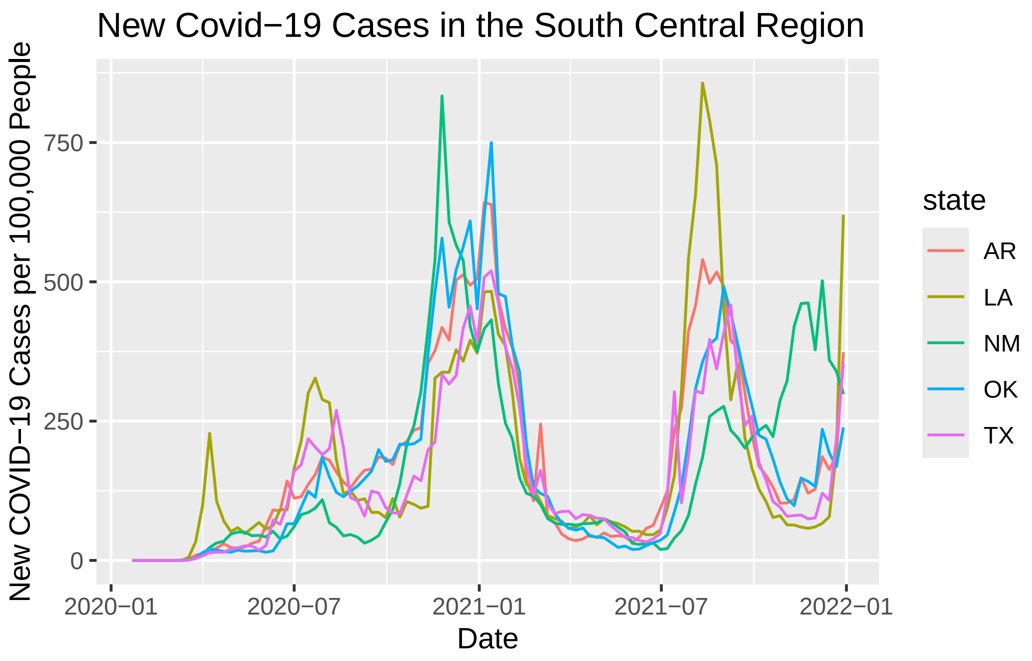
[[7]]


New Covid−19 Cases in the Pacific Region
New COVID−19 Cases per 100,000 People
0
250
500
750
2020−01
2020−07
2021−01
2021−07
2022−01
Date
state
AZ
CA
HI
NV

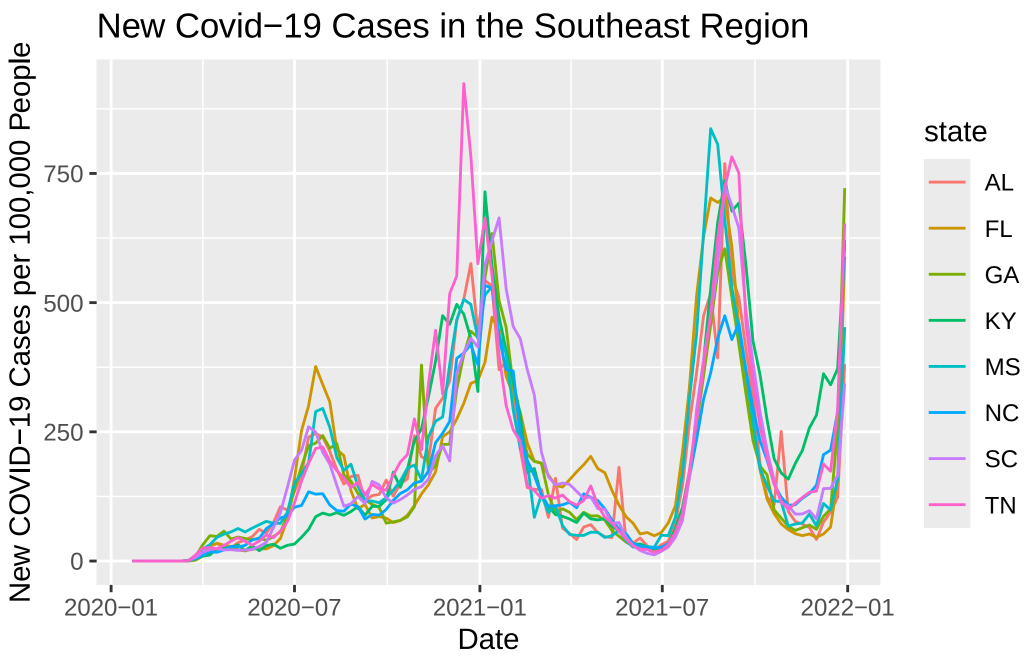
```
[[8]]
```


New Covid−19 Cases in the Pacific Northwest Region
New COVID−19 Cases per 100,000 People
0
300
600
900
1200
2020−01
2020−07
2021−01
2021−07
2022−01
Date
state
AK
ID
OR
WA

```
[[9]]
```


New Covid−19 Cases in the South Central Region
New COVID−19 Cases per 100,000 People
0
250
500
750
2020−01
2020−07
2021−01
2021−07
2022−01
Date
state
AR
LA
NM
OK
TX

```
[[10]]
```



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 |>
  filter(date >= as.Date("2020-01-01") & date <= as.Date("2021-12-31")) |> #
   ↪  filter for 2020-2021
  mutate(month = month(date), year = year(date)) |> # correct data types
  group_by(year, month) |>
  summarise(total_cases = sum(cases),.groups = "drop") |>
  mutate(month = month.name[month]) |> # get month names from number
  kable(caption = "New Nationwide COVID-19 Cases by Month")
```

Table 1: New Nationwide COVID-19 Cases by Month

year	month	total_cases
2020	January	11
2020	February	68

year	month	total_cases
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

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"
```

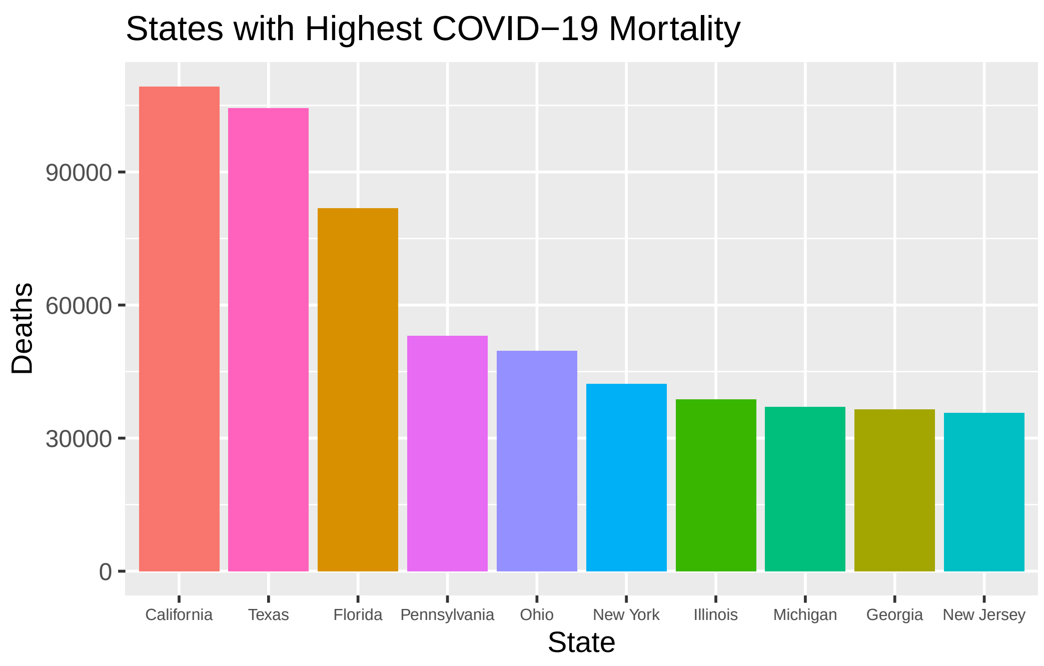
Use **httr2** to download COVID-19 death data from this endpoint. Make sure to remove the default limit to get all available data. Create a clean dataset called `deaths` with columns `state`, `date`, and `deaths` (renamed from the original column name). Ensure dates are in proper Date format and deaths are numeric.

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

# check response status
if (resp_status(response) == 200) {
  response |>
  resp_body_json(simplifyVector = TRUE) -> deaths
} else {
  stop(paste("request failed with, status:", resp_status(resp)))
}

deaths <- deaths |>
  filter(sex == "All Sexes" & age_group == "All Ages" & group == "By Total")
   ↪  |> # only use totals
  select(state = state, date = end_date, deaths = covid_19_deaths) |>
  mutate(date = as.Date(date), deaths = as.numeric(deaths))

head(deaths,10)
```

```
                  state       date  deaths
1         United States 2023-09-23 1146774
2               Alabama 2023-09-23   21520
3                Alaska 2023-09-23    1492
4               Arizona 2023-09-23   30307
5              Arkansas 2023-09-23   12663
6            California 2023-09-23  109248
7              Colorado 2023-09-23   15378
8           Connecticut 2023-09-23   12571
9              Delaware 2023-09-23    3445
10 District of Columbia 2023-09-23    2231
```

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" & state != "New York City") |> # exclude
   ↪  non-states
  arrange(desc(deaths)) |> # sort before taking top 10
  head(10) |>
  ggplot(aes(x=reorder(state,-deaths), y=deaths, fill=state)) + # reorder
   ↪  sorts in the plot
    geom_col() +
    labs(
      title = "States with Highest COVID-19 Mortality",
      x = "State",
      y = "Deaths"
    ) +
    theme(axis.text.x = element_text(size = 6), legend.position = "none") #
     ↪  make state names smaller so they don't overlap, drop legend
```


States with Highest COVID−19 Mortality
Deaths
California
Texas
Florida
Pennsylvania
Ohio
New York
Illinois
Michigan
Georgia
New Jersey
State