Problem set 4 - Allen Gu

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.

Task 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`.

```
## Your code here
census_key = readLines("census-key.R", n=1)
```

Task 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)
library(knitr)
url_endpoint = paste("https://api.census.gov/data/2021/pep/population?get=PO
↪  P_2020,POP_2021,NAME&for=state:*&key=", census_key, sep="")
request <- request(url_endpoint)
# print(request)
```

`print(request)` confirms the creation of a `httr2_request` object with the correct GET URL that contains my API key.

Task 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()
# print(response)
```

`print(response)` confirms a HTTP Status 200.

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

```
# Your code here
response |> resp_content_type()
```

```
[1] "application/json"
```

Task 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_raw <- response |> resp_body_json(simplifyVector = T)

# checking output, does not affect the fact that the above code took one line
population_raw |> head() |> kable()
```

POP_2020	POP_2021	NAME	state
3962031	3986639	Oklahoma	40
1961455	1963692	Nebraska	31
1451911	1441553	Hawaii	15
887099	895376	South Dakota	46
6920119	6975218	Tennessee	47

_

Task 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_raw |>
  # Use janitor row to names function
  row_to_names(1) |>
  # convert to tibble
  as.tibble() |>
  # remove stat column
  select(!state) |>
  # rename state column to state_name
```

```
  rename("state_name"="NAME") |>
  # use pivot_longer to tidy
  pivot_longer(!state_name, names_to="year", values_to="pop") |>
  # remove POP_ from year
  mutate(year=sapply(year, function(x) substr(x,5,8))) |>
  # parse all relevant columns to numeric
  mutate(year=as.numeric(year), pop=as.numeric(pop)) |>
  # 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 = as.character(state)))

population |> head() |> kable()
```

state_name	year	pop	state
Oklahoma	2020	3962031	OK
Oklahoma	2021	3986639	OK
Nebraska	2020	1961455	NE
Nebraska	2021	1963692	NE
Hawaii	2020	1451911	HI
Hawaii	2021	1441553	HI

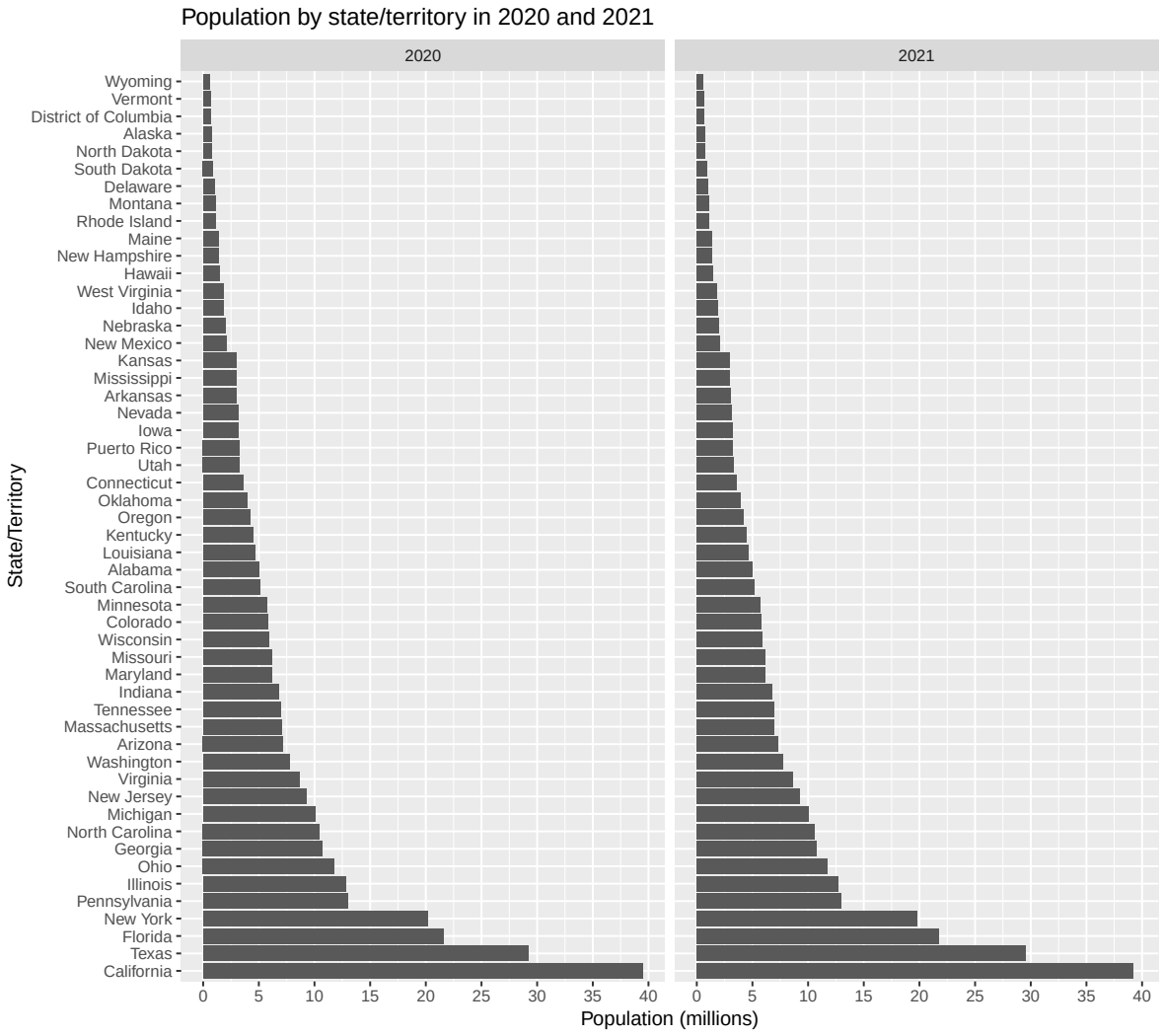
The following code confirms that the District of Columbia and Puerto Rico have had their abbreviations successfully inserted:

```
population |> filter(state %in% c("DC", "PR"))
```

```
# A tibble: 4 x 4
  state_name            year     pop state
  <chr>                <dbl>   <dbl> <chr>
1 District of Columbia  2020  690093 DC
2 District of Columbia  2021  670050 DC
3 Puerto Rico           2020 3281538 PR
4 Puerto Rico           2021 3263584 PR
```

Task 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 |>
  # reorder state
  arrange(fct_reorder(state_name, -pop)) |>
  # assign aesthetic mapping
  ggplot(aes(fct_inorder(state_name), pop/1e6)) +
  scale_y_continuous(breaks=seq(0,max(population$pop)/1e6+10,5)) +
  ylab("Population (millions)") +
  xlab("State/Territory") +
  labs(title="Population by state/territory in 2020 and 2021") +
  # use geom_col to plot barplot
  geom_col() +
  # flip coordinates
  coord_flip() +
  # facet by year
  facet_wrap(~year)
```



Task 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 three 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_raw <- #use jsonlit JSON parser
  request(url) |>
  req_perform()

regions_raw |> resp_content_type() # returns "text/plain"
```

```
[1] "text/plain"
```

```
regions_raw <- regions_raw |>
  resp_body_string() |>
  fromJSON(simplifyVector=F)

regions <- #convert list to data frame. You can use map_df in purrr package
  map_df(regions_raw, as_tibble) |>
  mutate(region_name=case_when(
    # renaming long region name
    region_name=="New York and New Jersey, Puerto Rico, Virgin Islands" ~
↪  "NY,NJ,PR,VI",
    .default = as.character(region_name))) |>
  rename("state_name"="states") |>
  # ensuring columns have right datatypes
  mutate(state_name=as.character(state_name),
         region_name=as.character(region_name),
         region=as.numeric(region)) |>
  filter(state_name %in% population$state_name)

regions |> head() |> kable()
```

region	region_name	state_name
1	New England	Connecticut
1	New England	Maine
1	New England	Massachusetts
1	New England	New Hampshire
1	New England	Rhode Island
1	New England	Vermont

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

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

state_name	year	pop	state	region	region_name
Oklahoma	2020	3962031	OK	6	South Central
Oklahoma	2021	3986639	OK	6	South Central
Nebraska	2020	1961455	NE	7	Central Plains
Nebraska	2021	1963692	NE	7	Central Plains
Hawaii	2020	1451911	HI	9	Pacific
Hawaii	2021	1441553	HI	9	Pacific

Task 10: From reading https://data.cdc.gov/ we learn the endpoint `https://data.cdc.gov/resource/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"
cases_raw <- request(api) |>
  req_perform()
cases_raw |> resp_content_type()
```

```
[1] "application/json"
```

```
cases_raw = cases_raw |>
  resp_body_string() |>
  fromJSON(simplifyVector=F) |>
  map_df(as.tibble)
sprintf("Rows received: %s", nrow(cases_raw))
```

```
[1] "Rows received: 1000"
```

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

In fact, if we go to the actual metadata page for this dataset, we see that there should be ~10,400 rows, not 1,000.

Task 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) |>
  # amending the limit; I don't like adding query arguments directly into
   ↪ URLs
  req_url_query("$limit"=10000000000) |>
  req_perform() |>
  resp_body_string() |>
  fromJSON(simplifyVector=F) |>
  map_df(as.tibble)
sprintf("Rows received: %s", nrow(cases_raw))
```

```
[1] "Rows received: 10380"
```

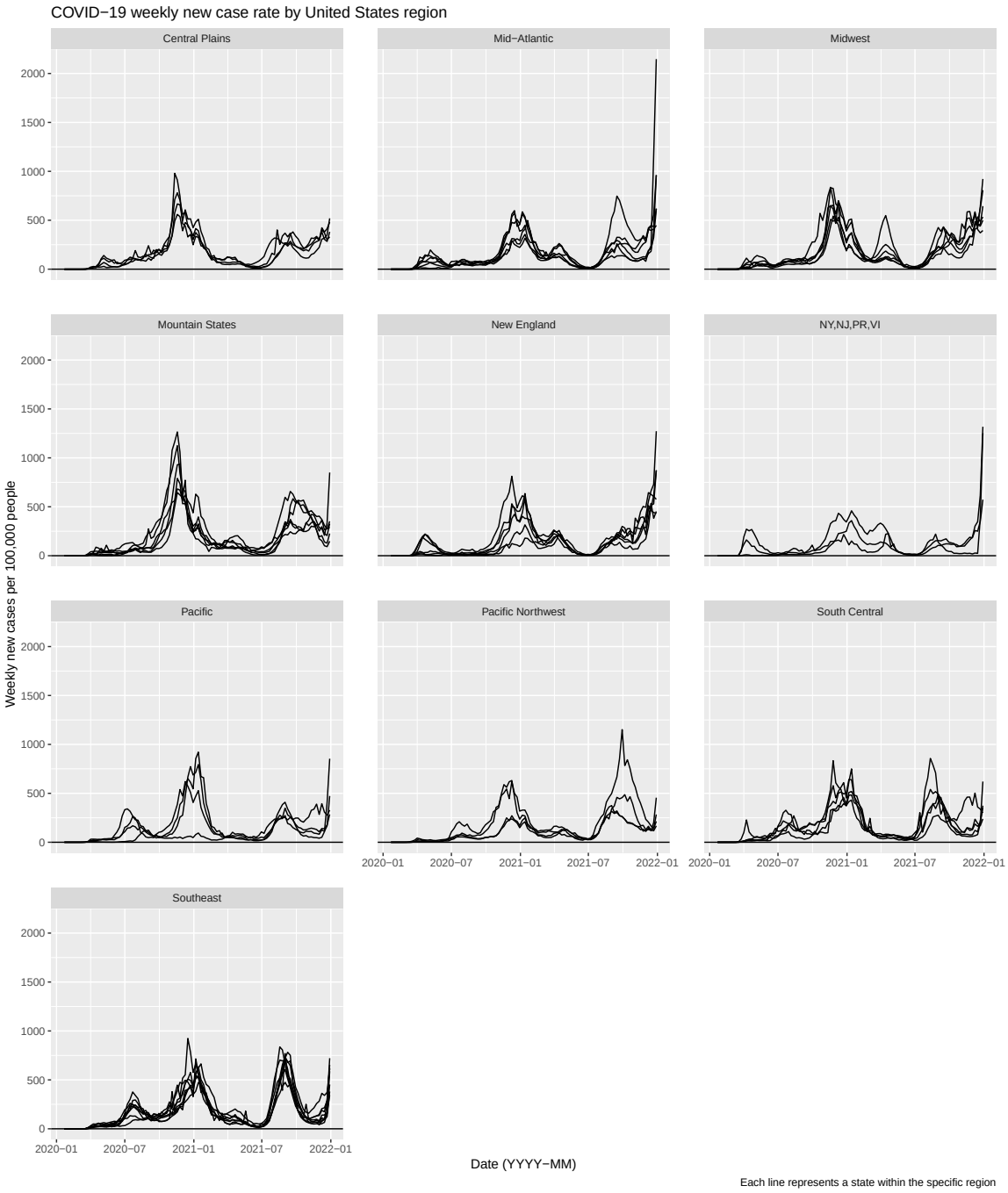
```
cases <- cases_raw |>
  select(state=state, date=end_date, cases=new_cases) |>
  mutate(cases=as.numeric(cases), date=substr(date,1,10))

cases |> head() |> kable()
```

state	date	cases
AZ	2023-02-22	3716
LA	2022-12-21	4041
GA	2023-02-22	5298
LA	2023-03-29	2203
LA	2023-02-01	5725
LA	2023-03-22	1961

state	date	cases

Task 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 |>
  #add a column for the year
  mutate(date=ymd(date)) |>
  mutate(year=year(date)) |>
  # inner-join with population on both state and year
  inner_join(population, by=c("state", "year")) |>
  # cases per 100000
  mutate(cases_pc=cases/pop*1e5) |>
  # plotting
  ggplot(aes(date, cases_pc, group=state_name)) +
  geom_line() +
  geom_hline(yintercept = 0) +
  facet_wrap(~region_name, ncol=3) +
  theme(panel.spacing=unit(2, "lines"),
        plot.margin=margin(0.5,1,0.5,0.5,"cm")) +
  xlab("Date (YYYY-MM)") + ylab("Weekly new cases per 100,000 people") +
  labs(title="COVID-19 weekly new case rate by United States region",
       caption="Each line represents a state within the specific region")
```



Task 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.

```
cases |>
  # parsing date with lubridate:ymd()
  mutate(date=ymd(date)) |>
  mutate(year=year(date)) |>
  mutate(month=month(date,label=T, abbr=F)) |>
  filter(year %in% c(2020,2021)) |>
  select(-c("state", "date")) |>
  # summing cases
  group_by(year, month) |>
  summarize(total_cases=sum(cases)) |>
  arrange(year, month) |>
  rename("Year"="year", "Month"="month", "Total cases"="total_cases") |>
  kable()
```

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

Year	Month	Total cases
2020	January	11
2020	February	68
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

Year	Month	Total cases
2021	May	972915
2021	June	493635
2021	July	1137440
2021	August	3572562
2021	September	5027537
2021	October	2356302
2021	November	2322814
2021	December	5615644

Task 14: The following URL provides additional COVID-19 data from the CDC in JSON format:

```
# Your code here
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_raw <- request(deaths_url) |>
  req_url_query("$limit"=10000000000) |>
  req_perform()
deaths_raw |> resp_content_type()
```

```
[1] "application/json"
```

```
deaths_raw = deaths_raw |>
  resp_body_string() |>
  fromJSON(simplifyVector=F) |>
  map_df(as.tibble)
sprintf("Received: %s rows and %s columns", nrow(deaths_raw),
 ↪  ncol(deaths_raw))
```

```
[1] "Received: 137700 rows and 16 columns"
```

Information about this dataset is available from here. This confirms that there should be about ~138,000 rows and 16 columns.

Interestingly, New York City is included in the dataset as an additional "state" (alongside the "United States"):

```
unique_states = unique(deaths_raw$state)
length(unique_states)
```

```
[1] 54
```

```
#
unique_states[!(unique_states %in% population$state_name)]
```

```
[1] "United States" "New York City"
```

In the CDC metadata, it is stated that "New York state estimates exclude New York City". Therefore we should be safe to re-name `New York City` to be `New York`, as we'll later be summing up by state anyway.

```
deaths <- deaths_raw |>
  select(state=state, date=end_date, n_deaths=covid_19_deaths) |>
  # coercing data values
  mutate(n_deaths=as.numeric(n_deaths), date=substr(date,1,10)) |>
  # removing US as a state
  filter(state != "United States") |>
  # renaming NYC to be NY
  mutate(state=case_when(state=="New York City" ~ "New York",
   ↪  .default=state))
```

Task 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.

```
print(min(deaths$date))
```

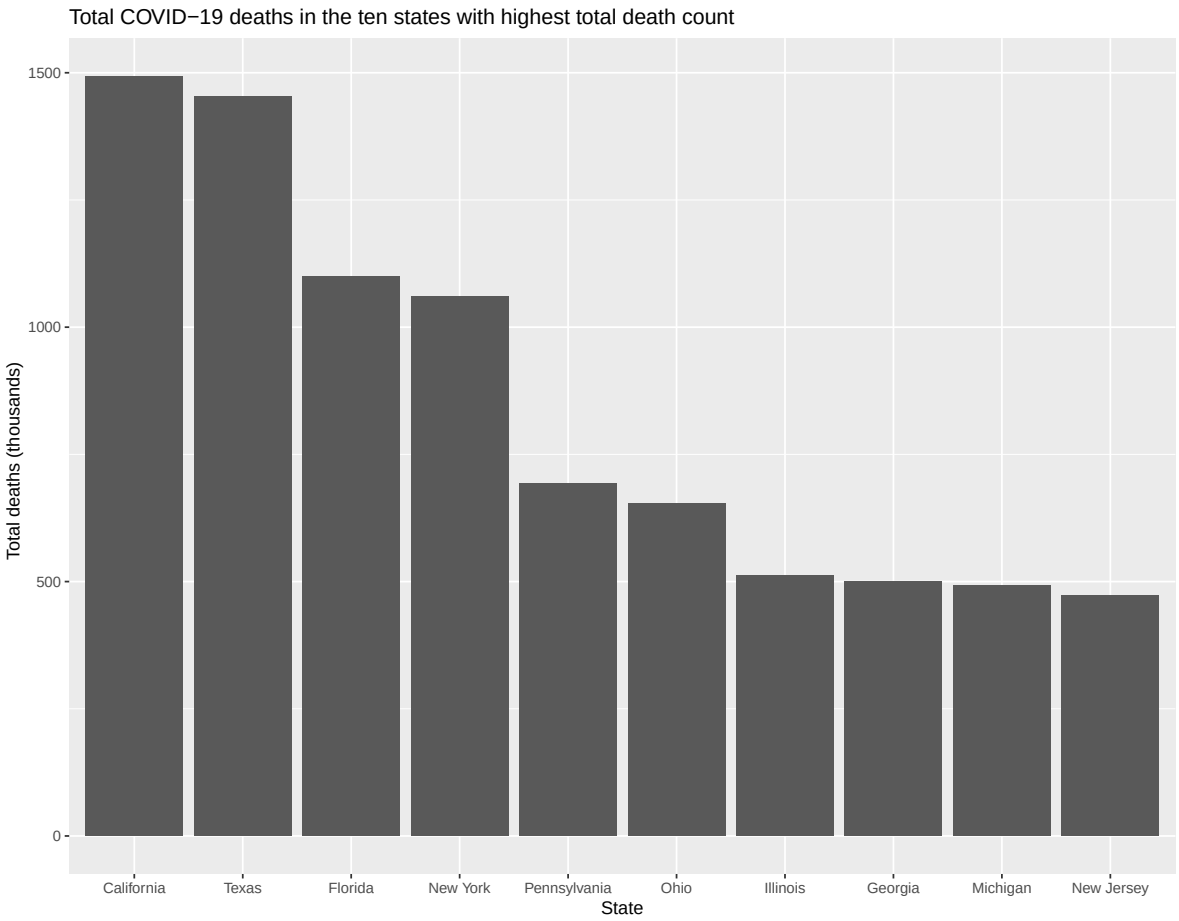
```
[1] "2020-01-31"
```

```
print(max(deaths$date))
```

```
[1] "2023-09-23"
```

The above code shows the range of the data.

```
# Your code here
deaths |>
  # add up all deaths in state
  group_by(state) |>
  summarise(total_deaths=sum(n_deaths, na.rm=T)) |>
  # get only top 10 total death counts
  arrange(desc(total_deaths)) |>
  slice(1:10) |>
  # plot
  ggplot(aes(fct_inorder(state), total_deaths/1e3)) +
  geom_col() +
  xlab("State") + ylab("Total deaths (thousands)  ") +
  labs(title="Total COVID-19 deaths in the ten states with highest total
   ↪  death count",
       caption="Date range: Jan 2020 - Sep 2023. Source: CDC")
```


Total COVID−19 deaths in the ten states with highest total death count
1500
1000
500
0
Total deaths (thousands)
California
Texas
Florida
New York
Pennsylvania
Ohio
Illinois
Georgia
Michigan
New Jersey
State
Date range: Jan 2020 − Sep 2023. Source: CDC