Problem set 4 sample solutions

2024-10-04

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` as done in the file `census-key.R`.

Solution:

```
# Sample answer
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.

```
# Sample answer
library(httr2)
request <- request(url) |>
  req_url_query(get = I("POP_2020,POP_2021,NAME"),
                `for` = I("state:*"),
                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.

```
# Sample answer
response <- request |> req_perform()
resp_status(response)
```

```
[1] 200
```

4. What is the content type of your response?

```
# Sample answer
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.

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

6. Examine the matrix. 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.

```
# Sample answer
#| message: false
#| warning: false
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```
library(janitor)
```

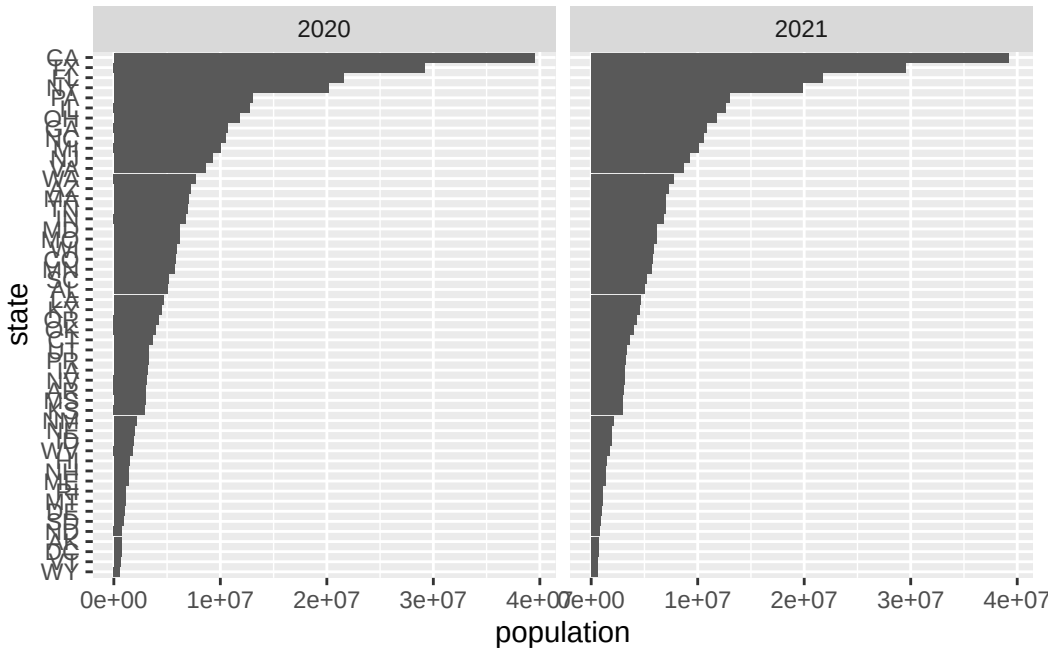
```
Attaching package: 'janitor'

The following objects are masked from 'package:stats':

    chisq.test, fisher.test
```

```
population <- population |> janitor::row_to_names(1) |>
  as_tibble() |>
  select(-state) |>
  rename(state_name = NAME) |>
```

```
  pivot_longer(-state_name, names_to = "year", values_to = "population") |>
  mutate(year = str_remove(year, "POP_")) |>
  mutate(across(-state_name, as.numeric)) |>
  mutate(state = state.abb[match(state_name, state.name)]) |>
  mutate(state = case_when(
    state_name == "District of Columbia" ~ "DC",
    state_name == "Puerto Rico" ~ "PR",
    .default = state))
```

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`.

```
# Sample answer
population |>
  mutate(state = reorder(state, population, mean)) |>
  ggplot(aes(state, population)) +
  geom_col() +
  coord_flip() +
  facet_wrap(~year)
```



8. The following URL:

```
url <-
 ↪  "https://github.com/datasciencelabs/2024/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.

```
# Sample answer
library(jsonlite)
```

```
Attaching package: 'jsonlite'
```

```
The following object is masked from 'package:purrr':

    flatten
```

```
library(purrr)
url <-
 ↪  "https://github.com/datasciencelabs/2024/raw/refs/heads/main/data/regions.json"
regions <- fromJSON(url, simplifyDataFrame = FALSE)
regions <- map_df(regions, function(x){
  data.frame(state_name = x$states, region = x$region, region_name =
   ↪  x$region_name)
}) |>
  mutate(region = factor(as.numeric(region))) |>
  mutate(region_name = ifelse(nchar(region_name) > 50, "NY,NJ,PR,USVI",
   ↪  region_name))
```

9. Add a region column to the `population` data frame.

```
# Sample answer
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.` 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.

```
# Sample answer
api <- "https://data.cdc.gov/resource/pwn4-m3yp.json"
cases_raw <- request(api) |> req_perform() |>
  resp_body_json(simplifyVector = TRUE)
dim(cases_raw)
```

```
[1] 1000   10
```

```
table(cases_raw$state)
```

```
 AK  AL  AR  AS  AZ  CA  DC  FL  GA  IL  KS  LA  NV NYC  OK  VA
164 170 168 173 173 135   2   1   1   1   2   5   1   1   1   2
```

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 recieve 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.

```
# Sample answer
api <- "https://data.cdc.gov/resource/pwn4-m3yp.json"
cases_raw <- request(api) |>
  req_url_query("$limit" = 10000000) |>
  req_perform() |>
  resp_body_json(simplifyVector = TRUE)
cases <- cases_raw |>
  mutate(cases = parse_number(new_cases),
         date = as_date(ymd_hms(end_date))) |>
  filter(state %in% population$state) |>
  select(state, date, cases) |>
  arrange(state, date)
table(cases$state)
```

```
 AK  AL  AR  AZ  CA  CO  CT  DC  DE  FL  GA  HI  IA  ID  IL  IN  KS  KY  LA  MA
173 173 173 173 173 173 173 173 173 173 173 173 173 173 173 173 173 173 173 173
```

```
 MD  ME  MI  MN  MO  MS  MT  NC  ND  NE  NH  NJ  NM  NV  NY  OH  OK  OR  PA  PR
173 173 173 173 173 173 173 173 173 173 173 173 173 173 173 173 173 173 173 173
 RI  SC  SD  TN  TX  UT  VA  VT  WA  WI  WV  WY
173 173 173 173 173 173 173 173 173 173 173 173
```

12. For 2020 and 2021, make a time series plot of cases per 100,000 versus time for each state. Stratify the plots by region.

```
# Sample answer
cases |>
  mutate(year = year(date)) |>
  filter(year >= 2020 & year <= 2021) |>
  left_join(population, by = c("state","year")) |>
  mutate(rate = cases/population*10^5) |>
  ggplot(aes(date, rate, color = state)) +
  geom_line(show.legend = FALSE) +
  scale_y_continuous(labels = scales::comma) +
  facet_wrap(~region_name) +
  labs(x = "Date", y = "Cases per 100,000", title = "SARS-Cov-2 cases per
   ↪  100,000 by state")
```