Solutions for the exercise session 2

1. SQL queries

Problem 1.1: Bikeshare trip durations in San Francisco.

1. Chatgpt uses a different dataset (instead of `san_francisco_bikeshare` it searches for a table in `san_francisco dataset`).
2. The code given by chatgpt does not actually need to join tables, we only need table that contains information on durations of trips (i.e. the table `bikeshare_trips`).

Below the correct and simplified code is presented.

```
#| eval: false
SELECT start_station_id AS station_name,
       AVG(trips.duration_sec / 60) AS avg_rent_time_minutes
FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips` AS trips
GROUP BY station_name
ORDER BY avg_rent_time_minutes DESC
```

Problem 1.2: NHTSA Traffic Fatalities dataset.

1. The dataset `bigquery-public-data.nhtsa_traffic_fatalities.accident_2020` seems to be unavailable, we analyze the dataset

   `bigquery-public-data.nhtsa_traffic_fatalities.accident_2016`.

2. No need to filter the year, since we already access the data with specific year.
3. No need to set the limit, since there might be several days of week with the maximal number of motor accidents.

Below the correct and simplified code is presented.

```
#| eval: false
SELECT FORMAT_TIMESTAMP('%A', TIMESTAMP(timestamp_of_crash)) AS day_of_week,
       COUNT(*) AS total_fatalities
FROM `bigquery-public-data.nhtsa_traffic_fatalities.accident_2016`
GROUP BY day_of_week
ORDER BY total_fatalities DESC
```

Problem 1.3: Austin longest duration bike rides.

```
#| eval: false
SELECT start_station_name
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips` AS A
ORDER BY duration_minutes DESC
LIMIT 5
```

Problem 1.4: Austin stolen bikes.

```
#| eval: false
SELECT COUNT(trip_id) AS count
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips` AS A
WHERE A.end_station_name = "Stolen"
```

Problem 1.5: Swiss customers

First we create our database.

```
#| warning: false
library(DBI); library(RSQLite)
con <- dbConnect(SQLite(), ":memory:")

DBI::dbExecute(con, "
CREATE TABLE customers (
  customer_id INTEGER PRIMARY KEY,
  name TEXT NOT NULL,
  country TEXT NOT NULL
);
")
```

```
DBI::dbExecute(con, "
CREATE TABLE orders (
  order_id INTEGER PRIMARY KEY,
  customer_id INTEGER NOT NULL,
  order_date TEXT NOT NULL, -- ISO8601
  amount REAL NOT NULL,
  FOREIGN KEY(customer_id) REFERENCES customers(customer_id)
);
")

DBI::dbWriteTable(con, "customers",
  data.frame(customer_id=1:5,
             name=c("Ada","Bruno","Chloé","Dinesh","Elena"),
             country=c("CH","FR","CH","IN","US")),
  append=TRUE)

DBI::dbWriteTable(con, "orders",
  data.frame(order_id=1:10,
             customer_id=c(1,1,3,3,3,3,4,4,5,5),
             order_date=as.character(as.Date("2025-09-20")
             + c(0,1,0,2,5,6,1,7,3,9)),
             amount=c(120,75,300,42,88,150,60,500,20,220)),
  append=TRUE)

DBI::dbListTables(con)
```

Now we select names of swiss customers.

```
DBI::dbGetQuery(con, "
    SELECT name
    FROM customers
    WHERE country = 'CH'
    ORDER by name
    LIMIT 3;
")
```

Though we only display top 3 names, there was no difference if we would not set the limit (check it!).

Problem 1.6: Average number of orders per customer.

```
DBI::dbGetQuery(con, "
  SELECT c.name, AVG(o.amount) AS avg
  FROM customers AS c
  INNER JOIN orders AS o
    ON c.customer_id = o.customer_id
  GROUP BY c.name
  ORDER BY avg DESC
  LIMIT 3;
")
```

Problem 1.7: Customers with no orders.

```
DBI::dbGetQuery(con, "
  SELECT c.customer_id, c.name
  FROM customers c
  LEFT JOIN orders o
    ON o.customer_id = c.customer_id
  WHERE o.order_id IS NULL;
")
```

2. Accessing data via API keys.

Problem 2.1: Weather in Bern.

```
#| warning: false
#| eval: false
# packages
library(httr2)
library(dplyr)

# store your key in an env var before class:
Sys.setenv(OPENWEATHERMAP_API_KEY = your_key)
okey <- Sys.getenv("OPENWEATHERMAP_API_KEY")

resp <- request("https://api.openweathermap.org/data/2.5/weather") |>
  req_url_query(q = "Bern", units = "metric", appid = okey) |>
```

```
  req_perform()

wx <- resp_body_json(resp, simplifyVector = TRUE)

# extract a compact summary row
wx_row <- tibble::tibble(
  city        = wx$name,
  country     = wx$sys$country,
  temperature = wx$main$temp,
  feels_like  = wx$main$feels_like,
  humidity    = wx$main$humidity,
  wind_ms     = wx$wind$speed,
  condition   = wx$weather$description,
  timestamp   = as.POSIXct(wx$dt, origin = "1970-01-01", tz = "UTC")
)
wx_row
```

Problem 2.2: Information on OMDB movies.

```
#| eval: false
library(httr2)
Sys.setenv(OMDB_API_KEY = your_key)
mkey <- Sys.getenv("OMDB_API_KEY")
# Search first
s <- request("https://www.omdbapi.com/") |>
  req_url_query(apikey = mkey,
  s = "The Lord of the Rings: The Fellowship of the Ring", y = 2001) |>
  req_perform() |>
  resp_body_json(simplifyVector = TRUE)
head(s$Search)

# Then fetch details by IMDb id
movie <- request("https://www.omdbapi.com/") |>
  req_url_query(apikey = mkey, i = "tt0120737") |>
  req_perform() |>
  resp_body_json(simplifyVector = TRUE)

unlist(movie[c("Title","Year","Genre","Director","imdbRating")])
```

```python
import requests
import pandas as pd

url = "https://en.wikipedia.org/wiki/Python_(programming_language)"
headers = {
    # Be polite & descriptive per Wikimedia's policy:
    "User-Agent": "Timofei-Education (timofei98shashkov@gmail.com) requests"
}
resp = requests.get(url, headers=headers, timeout=60)
resp.raise_for_status()  # will raise if still not 200

tables = pd.read_html(resp.text)  # requires lxml or html5lib installed
print(len(tables))
print(tables[5].shape)
```

```
17
(16, 4)
```

```
C:\Users\Timofei\AppData\Local\Temp\ipykernel_14288\3833656271.py:12: FutureWarning: Passing
  tables = pd.read_html(resp.text)  # requires lxml or html5lib installed
```

Problem 2.3: Alpha Vantage

```r
library(httr2)
library(purrr)
library(dplyr)

akey <- Sys.getenv("ALPHAVANTAGE_API_KEY", unset = "demo")
resp <- request("https://www.alphavantage.co/query") |>
  req_url_query(`function` = "TIME_SERIES_DAILY", symbol = "IBM", apikey = akey) |>
  req_perform()
js <- resp_body_json(resp, simplifyVector = TRUE)

daily <- js[["Time Series (Daily)"]]
prices <- imap_dfr(daily, ~ tibble::tibble(
  date   = as.Date(.y),
  open   = as.numeric(.x[["1. open"]]),
  high   = as.numeric(.x[["2. high"]]),
  low    = as.numeric(.x[["3. low"]]),
```

```
  close  = as.numeric(.x[["4. close"]]),
  volume = as.numeric(.x[["5. volume"]])
)) |>
  arrange(date)
prices
```

3. Webscraping exercises

We will explore the webpage Python (programming language).

Problem 3.1. Hunting for table CSS selector.

Using selector gadget extension to hunt for CSS selectors for sections. The selector `.mw-heading2` describes exactly what we wanted.

Problem 3.2. Scraping a table.

Next we scrape the table `Summary of Python 3's built-in types` in **R** using `html_table`.

```
#| warning: false
library(rvest)

url <- "https://en.wikipedia.org/wiki/Python_(programming_language)"
tables <- url %>%
  read_html() %>%
  html_table(fill = TRUE)
tables
length(tables)
dim(tables[[6]])
```

Alternatively a table can be scraped using, for example, selector gadget.

Problem 3.3. Count domain frequences.

```
library(rvest)
library(urltools)
library(dplyr)

url <- "https://en.wikipedia.org/wiki/Python_(programming_language)"
links <- read_html(url) %>%
  html_nodes("a") %>%
  html_attr("href") %>%
  na.omit()

links <- links[grepl("^http", links) | grepl("^/wiki/", links)]
links <- unique(links)[1:20]
links <- ifelse(grepl("^/wiki/", links), paste0("https://en.wikipedia.org", links), links)

domains <- domain(links)
table(domains)
```

Problem 3.4. Ethical issues.

You can add options to executable code like this

```
library(polite)
robotstxt::paths_allowed("https://en.wikipedia.org/wiki/Python_(programming_language)")
```