# Data preparation

## Instructions

- You only need to submit the .Rmd of this file, not a PDF.
- You should **comment** your code clearly to show what you've done to prepare the data.
- The purpose of this file is to use the data in the `data-raw` folder to create the data you will use in the report. The data you will use in the report should be saved in the `data` folder. It is good professional practice to make sure you're never directly modifying your raw data, but instead creating new datasets based on merges/manipulations that you need to reuse.
- Make sure you've taken a look at the hints for the web scraping and census API.
- You may find the `write_rds()` function from the `readr` package helpful (it is loaded as part of the `tidyverse`).
- You do not need to keep the structure below.

## Set up

```r
# Set up any libraries you need
library(tidyverse)
library(rvest)
library(polite)
library(cancensus)
library(haven)
library(lubridate)
```

## Loading client data

```r
# Load all client related Rds files
cust_dev = read_rds("data-raw/cust_dev.Rds")
customer = read_rds("data-raw/customer.Rds")
cust_sleep = read_rds("data-raw/cust_sleep.Rds")
device = read_rds("data-raw/device.Rds")
```

## Getting external data

### Web scraping industry data

```r
# Scraping the industry data
url <- "https://fitnesstrackerinfohub.netlify.app/"

# Make sure this code is updated appropriately to provide
# informative user_agent details
```

```
target <- bow(url,
              user_agent = "jiahao.bai@mail.utoronto.ca for STA303/1002 project",
              force = TRUE)

# Any details provided in the robots text on crawl delays and
# which agents are allowed to scrape
target
```

```
## <polite session> https://fitnesstrackerinfohub.netlify.app/
##     User-agent: jiahao.bai@mail.utoronto.ca for STA303/1002 project
##     robots.txt: 2 rules are defined for 2 bots
##    Crawl delay: 12 sec
##   The path is scrapable for this user-agent
```

```
html <- scrape(target)

device_data <- html %>%
  html_elements("table") %>%
  html_table() %>%
  pluck(1) # added, in case you're getting a list format
```

## Census API

```
# Scraping census data
options(cancensus.api_key = "CensusMapper_b041be566f3982329277d23be6e52e8d",
        cancensus.cache_path = "cache") # this sets a folder for your cache


# get all regions as at the 2016 Census (2020 not up yet)
regions <- list_census_regions(dataset = "CA16")

regions_filtered <-  regions %>%
  filter(level == "CSD") %>% # Figure out what CSD means in Census data
  as_census_region_list()

# This can take a while
# We want to get household median income
census_data_csd <- get_census(dataset='CA16', regions = regions_filtered,
                          vectors=c("v_CA16_2397"),
                          level='CSD', geo_format = "sf")

# Simplify to only needed variables
median_income <- census_data_csd %>%
  as_tibble() %>%
  select(CSDuid = GeoUID, contains("median"), Population) %>%
  mutate(CSDuid = parse_number(CSDuid)) %>%
  rename(hhld_median_inc = 2)
```

## Postal code

```
# Load postal code data
dataset = read_rds("data-raw/break_glass_in_case_of_emergency.Rds")
```

```r
postcode <- dataset %>%
  select(PC, CSDuid)
```

# Data Merging and Cleaning

## Customer Data

```r
# Join customer data and device data
customer_new = customer %>% left_join(cust_dev, by = "cust_id") %>%
               left_join(device, by = "dev_id")

# Postcode
postcode_new = postcode %>% distinct(PC, .keep_all = TRUE) %>%
               left_join(median_income, by = "CSDuid")

# Merge customer and postcode data
customer_new = customer_new %>%
               left_join(postcode_new, by = c("postcode" = "PC"))

# Remove unnecessary columns of customer data, NA/Intersex customer data,
# calculate remaining customers age
customer_final = customer_new %>%
                 select(-pronouns, -dev_id, -postcode, -released) %>%
                 filter(!is.na(sex)) %>%
                 filter(sex %in% c("Female", "Male")) %>%
                 # mutate(age = decimal_date(today()) - decimal_date(dob)) %>%
                 mutate(age = 2022 - as.numeric(substr(dob, 1, 4))) %>%
                 select(-dob)

# Check number of distinct emoji modifiers, create new column to represent
# different skin color, and remove emoji_modifier.
unique(customer_final$emoji_modifier) # 6 distinct emoji modifiers in total
```

```
## [1] "U+1F3FF" NA        "U+1F3FD" "U+1F3FC" "U+1F3FB" "U+1F3FE"
```

```r
customer_final = customer_final %>%
                 mutate(skinColor = case_when(
                            emoji_modifier == "U+1F3FF" ~ "Dark",
                            is.na(emoji_modifier) ~ "Default",
                            emoji_modifier == "U+1F3FD" ~ "Medium",
                            emoji_modifier == "U+1F3FC" ~ "Medium-light",
                            emoji_modifier == "U+1F3FB" ~ "Light",
                            emoji_modifier == "U+1F3FE" ~ "Medium-dark")) %>%
                 select(-emoji_modifier)
```

## Customer Sleep Data

```r
# Merge customer and customer sleep data
sleep_data = cust_sleep %>% left_join(customer_final, by = "cust_id")
```

## Export Data

```r
# Export Data to data folder
write_rds(customer_final, "data/customer_data.Rds")
write_rds(sleep_data, "data/customer_sleep_data.Rds")
```