Homework 4

Eric Tang

2025-02-26

Data Structures

1. In R, the `scan()` function is used to read (typically numeric) data into vectors, whereas `readLines()` reads entire lines from text files into a character vector. `read_html` loads data from HTML files into an XML document and can used for web scraping. Finally, `readxl` reads Excel files into a tibble without the use of Java like `read.xlsx` requires.

2. The S3 class in R is the simplest and most commonly used object-oriented programming system. This class uses generic functions and method dispatch and is thus simple and lightweight. The S4 class is stricter and requires formal class definitions with strict validation. The R6 classes provides mutable objects, whereas the previous classes are based on copy-on-modify. R6 is used with performance-sensitive applications, such as Shiny or APIs.

Data Structures

1. Completed!

2. See this Git repository.

Tidyverse

1. d. `co2` is not tidy: to be tidy we would have to wrangle it to have three columns (year, month and value), then each co2 observation would have a row.

2. b. `ChickWeight` is tidy: each observation (a weight) is represented by one row. The chick from which this measurement came is one of the variables.

3. c. `BOD` is tidy: each row is an observation with two values (time and demand)

4. `DNase`, `Formaldehyde`, and `Orange` are tidy.

5.

```r
murders <- mutate(murders, rate = total / (population / 10^5))
```

6.

```r
murders <- mutate(murders, rank = rank(-rate))
```

7.

```r
select(murders, state, abb) %>% head()
```

```
##        state abb
## 1    Alabama  AL
## 2     Alaska  AK
## 3    Arizona  AZ
## 4   Arkansas  AR
## 5 California  CA
## 6   Colorado  CO
```

8.

```r
filter(murders, rank <= 5)
```

```
##                  state abb        region population total      rate rank
## 1 District of Columbia  DC         South     601723    99 16.452753    1
## 2            Louisiana  LA         South    4533372   351  7.742581    2
## 3             Maryland  MD         South    5773552   293  5.074866    4
## 4             Missouri  MO North Central    5988927   321  5.359892    3
## 5       South Carolina  SC         South    4625364   207  4.475323    5
```

9.

```r
no_south <- filter(murders, region != "South")
nrow(no_south)
```

```
## [1] 34
```

10.

```r
murders_nw <- filter(murders, region %in% c("Northeast", "West"))
nrow(murders_nw)
```

```
## [1] 22
```

11.

```r
my_states <- filter(murders, region %in% c("Northeast", "West") & rate < 1)
select(my_states, state, rate, rank)
```

```
##           state      rate rank
## 1        Hawaii 0.5145920   49
## 2         Idaho 0.7655102   46
## 3         Maine 0.8280881   44
## 4 New Hampshire 0.3798036   50
## 5        Oregon 0.9396843   42
## 6          Utah 0.7959810   45
## 7       Vermont 0.3196211   51
## 8       Wyoming 0.8871131   43
```

12.

```r
filter(murders, region %in% c("Northeast", "West") & rate < 1) %>%
  select(state, rate, rank)
```

```
##           state      rate rank
## 1        Hawaii 0.5145920   49
## 2         Idaho 0.7655102   46
## 3         Maine 0.8280881   44
## 4 New Hampshire 0.3798036   50
## 5        Oregon 0.9396843   42
## 6          Utah 0.7959810   45
## 7       Vermont 0.3196211   51
## 8       Wyoming 0.8871131   43
```

13.

```r
data(murders)
my_states <- murders %>%
  mutate(rate = total / (population / 10^5)) %>%
  mutate(rank = rank(-rate)) %>%
  filter(region %in% c("Northeast", "West") & rate < 1) %>%
  select(state, rate, rank)
print(my_states)
```

```
##           state      rate rank
## 1        Hawaii 0.5145920   49
## 2         Idaho 0.7655102   46
## 3         Maine 0.8280881   44
## 4 New Hampshire 0.3798036   50
## 5        Oregon 0.9396843   42
## 6          Utah 0.7959810   45
## 7       Vermont 0.3196211   51
## 8       Wyoming 0.8871131   43
```

14.

```r
ref <- filter(NHANES, AgeDecade == " 20-29" & Gender == "female") %>%
  summarize(
    avg = mean(BPSysAve, na.rm = T),
    std = sd(BPSysAve, na.rm = T))
print(ref)
```

```
## # A tibble: 1 x 2
##     avg   std
##   <dbl> <dbl>
## 1  108.  10.1
```

15.

```r
ref_avg <- filter(NHANES, AgeDecade == " 20-29" & Gender == "female") %>%
  summarize(
    avg = mean(BPSysAve, na.rm = T)) %>%
  pull(avg)
```

16.

```r
filter(NHANES, AgeDecade == " 20-29" & Gender == "female") %>%
  summarize(
    min = min(BPSysAve, na.rm = T),
    max = max(BPSysAve, na.rm = T)) %>%
  pull(min, max)
```

```
## 179
##  84
```

17.

```r
female_avg <- filter(NHANES, Gender == "female") %>%
  group_by(AgeDecade) %>%
  summarize(
    avg = mean(BPSysAve, na.rm = T),
    std = sd(BPSysAve, na.rm = T))
```

18.

```r
male_avg <- filter(NHANES, Gender == "male") %>%
  group_by(AgeDecade) %>%
  summarize(
    avg = mean(BPSysAve, na.rm = T),
    std = sd(BPSysAve, na.rm = T))
```

19.

```r
combined_avg <- group_by(NHANES, AgeDecade, Gender) %>%
  summarize(
    avg = mean(BPSysAve, na.rm = T),
    std = sd(BPSysAve, na.rm = T))
```

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

20.

```r
filter(NHANES, Gender == "male" & AgeDecade == " 40-49") %>%
  group_by(Race1) %>%
  summarize(
    avg = mean(BPSysAve, na.rm = T)) %>%
  arrange(avg)
```

```
## # A tibble: 5 x 2
##   Race1      avg
##   <fct>    <dbl>
## 1 White     120.
## 2 Other     120.
## 3 Hispanic  122.
## 4 Mexican   122.
## 5 Black     126.
```

21. b. murders is in tidy format and is stored in a data frame.

22.

```
murders_tibble <- as_tibble(murders)
class(murders_tibble)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

23.

```
murders_tibble <- murders %>%
  as_tibble() %>%
  group_by(region)
```

24.

```
murders %>%
  .$population %>%
  log() %>%
  mean() %>%
  exp()
```

```
## [1] 3675209
```

25.

```
df <- map_df(1:100, function(n) {
  data.frame(
    n = n,
    s_n = sum(1:n),
    s_n_2 = sum(1:n)
  )
})
```

R Packages and Shiny

1. First, the app is initialized, but no content has been added. The addition of `titlePanel("k-means clustering")`, adds a title to the page. UI inputs are added with the `selectInput()` functions, which allow the user to choose values to plot on the X and Y axes. A plot is generated in the main panel with `mainPanel()` and `output$plot1`. Adding k-means clusters the data into multiple groups, from which the centers can be calculated. The final app fully colors data points and groups them into clusters depending on the user input.

2. See https://github.com/ericktang/FDS_homework4/tree/main/kmeans