Challenge-4

Wong Zi Xin

2023-09-17

Questions

Load the "CommQuest2023.csv" dataset using the `read_csv()` command and assign it to a variable named "comm_data."

```
# Enter code here
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.2     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.4.3     ✔ tibble    3.2.1
## ✔ lubridate 1.9.2     ✔ tidyr     1.3.0
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
comm_data <- read.csv("CommQuest2023_Larger.csv")
```

Question-1: Communication Chronicles

Using the select command, create a new dataframe containing only the "date," "channel," and "message" columns from the "comm_data" dataset.

Solution:

```
# Enter code here
select(comm_data,date,channel,message)
```

Question-2: Channel Selection

Use the filter command to create a new dataframe that includes messages sent through the "Twitter" channel on August 2nd.

Solution:

```
# Enter code here
comm_data %>% filter(date == "2023-08-02",channel == "Twitter") %>% select(date,channel,message)
```

```
##         date channel         message
## 1  2023-08-02 Twitter    Team meeting
## 2  2023-08-02 Twitter  Exciting news!
## 3  2023-08-02 Twitter  Exciting news!
## 4  2023-08-02 Twitter  Exciting news!
## 5  2023-08-02 Twitter  Exciting news!
## 6  2023-08-02 Twitter    Team meeting
## 7  2023-08-02 Twitter     Great work!
## 8  2023-08-02 Twitter Hello everyone!
## 9  2023-08-02 Twitter Hello everyone!
## 10 2023-08-02 Twitter Need assistance
## 11 2023-08-02 Twitter Need assistance
## 12 2023-08-02 Twitter Need assistance
## 13 2023-08-02 Twitter  Exciting news!
## 14 2023-08-02 Twitter Need assistance
## 15 2023-08-02 Twitter Need assistance
```

Question-3: Chronological Order

Utilizing the arrange command, arrange the "comm_data" dataframe in ascending order based on the "date" column.

Solution:

```
# Enter code here
comm_data %>% arrange(comm_data, date)
```

Question-4: Distinct Discovery

Apply the distinct command to find the unique senders in the "comm_data" dataframe.

Solution:

```
# Enter code here
comm_data %>% distinct(sender)
```

```
##         sender
## 1  dave@example
## 2   @bob_tweets
## 3   @frank_chat
## 4   @erin_tweets
## 5 alice@example
## 6   carol_slack
```

Question-5: Sender Stats

Employ the count and group_by commands to generate a summary table that shows the count of messages sent by each sender in the "comm_data" dataframe.

Solution:

```
# Enter code here
comm_data %>%
  group_by(sender) %>%
  summarise(count=n())
```

```
## # A tibble: 6 × 2
##   sender         count
##   <chr>          <int>
## 1 @bob_tweets      179
## 2 @erin_tweets     171
## 3 @frank_chat      174
## 4 alice@example    180
## 5 carol_slack      141
## 6 dave@example     155
```

Question-6: Channel Chatter Insights

Using the group_by and count commands, create a summary table that displays the count of messages sent through each communication channel in the "comm_data" dataframe.

Solution:

```
# Enter code here
comm_data %>%
  group_by(channel) %>%
  summarise(count=n())
```

```
## # A tibble: 3 × 2
##   channel count
##   <chr>   <int>
## 1 Email     331
## 2 Slack     320
## 3 Twitter   349
```

Question-7: Positive Pioneers

Utilize the filter, select, and arrange commands to identify the top three senders with the highest average positive sentiment scores. Display their usernames and corresponding sentiment averages.

Solution:

```
# Enter code here
comm_data %>%
  group_by(sender) %>%
  summarise(average_positive_sentiment_scores = mean(sentiment)) %>%
  filter(average_positive_sentiment_scores>0) %>%
  arrange(desc(average_positive_sentiment_scores)) %>%
  select(sender,average_positive_sentiment_scores) %>%
  slice(1:3)
```

```
## # A tibble: 3 × 2
##   sender         average_positive_sentiment_scores
##   <chr>                                      <dbl>
## 1 carol_slack                               0.118
## 2 alice@example                             0.0570
## 3 dave@example                              0.00687
```

Question-8: Message Mood Over Time

With the group_by, summarise, and arrange commands, calculate the average sentiment score for each day in the "comm_data" dataframe.

Solution:

```
# Enter code here
comm_data %>%
  group_by(date) %>%
  arrange(date) %>%
  summarise(average_sentiment_score = mean(sentiment))
```

```
## # A tibble: 20 × 2
##    date       average_sentiment_score
##    <chr>                        <dbl>
##  1 2023-08-01                -0.0616
##  2 2023-08-02                 0.136
##  3 2023-08-03                 0.107
##  4 2023-08-04                -0.0510
##  5 2023-08-05                 0.193
##  6 2023-08-06                -0.0144
##  7 2023-08-07                 0.0364
##  8 2023-08-08                 0.0666
##  9 2023-08-09                 0.0997
## 10 2023-08-10                -0.0254
## 11 2023-08-11                -0.0340
## 12 2023-08-12                 0.0668
## 13 2023-08-13                -0.0604
## 14 2023-08-14                -0.0692
## 15 2023-08-15                 0.0617
## 16 2023-08-16                -0.0220
## 17 2023-08-17                -0.0191
## 18 2023-08-18                -0.0760
## 19 2023-08-19                 0.0551
## 20 2023-08-20                 0.0608
```

Question-9: Selective Sentiments

Use the filter and select commands to extract messages with a negative sentiment score (less than 0) and create a new dataframe.

Solution:

```
# Enter code here
comm_data %>%
  filter(sentiment < 0) %>%
  select(date,channel,sender,message,sentiment)
```

Question-10: Enhancing Engagement

Apply the mutate command to add a new column to the "comm_data" dataframe, representing a sentiment label: "Positive," "Neutral," or "Negative," based on the sentiment score.

Solution:

```
# Enter code here
comm_data %>%
  mutate(sentiment_label = case_when(
    sentiment > 0 ~ "Positive",
    sentiment == 0 ~ "Neutral",
    sentiment < 0 ~ "Negative"))
```

Question-11: Message Impact

Create a new dataframe using the mutate and arrange commands that calculates the product of the sentiment score and the length of each message. Arrange the results in descending order.

Solution:

```
# Enter code here
comm_data %>%
  mutate(sentiment_product = sentiment * nchar(message)) %>%
  arrange(desc(sentiment_product))
```

Question-12: Daily Message Challenge

Use the group_by, summarise, and arrange commands to find the day with the highest total number of characters sent across all messages in the "comm_data" dataframe.

Solution:

```
# Enter code here
comm_data %>%
  group_by(date) %>%
  summarise(total_characters = sum(nchar(message))) %>%
  arrange(desc(total_characters)) %>%
    slice(1)
```

```
## # A tibble: 1 × 2
##   date       total_characters
##   <chr>                 <int>
## 1 2023-08-10              875
```

Question-13: Untidy data

Can you list at least two reasons why the dataset illustrated in slide 10 is non-tidy? How can it be made Tidy?

Solution: Firstly, variables are not organised properly and each observation does not form a row. For example, "Unemployed", "Unemployment Rate" and "Employed" all exist as rows instead of columns in the table. Secondly, there are blanks in the rows, which makes the table harder to read and comprehend. Reasons why the dataset is non-tidy: Age classification is included as sub-headings in the data-set, so is the employment under each age category. Instead, we could have had age and employment as two separate columns or variables. That would make the data-set conform to Tidy data structure.