Challenge-4

Elise Wong

2023-09-04

Questions

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

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

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ─
─
## ✔ dplyr     1.1.0     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.4.3     ✔ tibble    3.1.8
## ✔ lubridate 1.9.2     ✔ tidyr     1.3.0
## ✔ purrr     1.0.1
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ─
─
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the ]8;;http://conflicted.r-lib.org/conflicted package]8;; to force a
ll conflicts to become errors
```

```
glimpse(comm_data)
```

```
## Rows: 1,000
## Columns: 5
## $ date      <chr> "2023-08-11", "2023-08-11", "2023-08-11", "2023-08-18", "202…
## $ channel   <chr> "Twitter", "Email", "Slack", "Email", "Slack", "Email", "Twi…
## $ sender    <chr> "dave@example", "@bob_tweets", "@frank_chat", "@frank_chat",…
## $ message   <chr> "Fun weekend!", "Hello everyone!", "Hello everyone!", "Fun w…
## $ sentiment <dbl> 0.8240997, 0.6624869, -0.1434508, 0.3801966, 0.1879540, -0.1…
```

```
df1 <- select(comm_data,
       date, channel, message)
head(df1)
```

```
##         date channel         message
## 1 2023-08-11 Twitter    Fun weekend!
## 2 2023-08-11   Email Hello everyone!
## 3 2023-08-11   Slack Hello everyone!
## 4 2023-08-18   Email    Fun weekend!
## 5 2023-08-14   Slack Need assistance
## 6 2023-08-04   Email Need assistance
```

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(channel == "Twitter", date == "2023-08-02") %>%
    select(channel, message, date)
```

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

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
df3 <- arrange(comm_data, date)
head(df3)
```

```
##         date channel        sender         message  sentiment
## 1 2023-08-01 Twitter alice@example Need assistance  0.6767770
## 2 2023-08-01 Twitter   @bob_tweets Need assistance  0.1483952
## 3 2023-08-01 Twitter   @frank_chat Need assistance  0.5990454
## 4 2023-08-01 Twitter   @frank_chat  Exciting news! -0.8227803
## 5 2023-08-01   Slack   @frank_chat    Team meeting -0.2020947
## 6 2023-08-01   Slack   @bob_tweets  Exciting news!  0.1463969
```

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(mean_sentiment = mean(sentiment)) %>%
    arrange(desc(mean_sentiment)) %>%
    slice(1:3)
```

```
## # A tibble: 3 × 2
##   sender        mean_sentiment
##   <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) %>%
  summarise(mean_sentiment = mean(sentiment))
```

```
## # A tibble: 20 × 2
##    date       mean_sentiment
##    <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
df9 <- comm_data %>%
  filter(sentiment <= 0) %>%
    select(message, sentiment)
head(df9)
```

```
##           message  sentiment
## 1 Hello everyone! -0.1434508
## 2 Need assistance -0.1083762
## 3 Hello everyone! -0.7408555
## 4 Hello everyone! -0.1879179
## 5 Hello everyone! -0.9325254
## 6 Need assistance -0.8794133
```

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
df10 <- comm_data %>%
  mutate(sentiment = case_when(
         sentiment > 0 ~ "Positive",
         sentiment == 0 ~ "Neutral",
         sentiment < 0 ~ "Negative"
         ))
head(df10)
```

```
##         date channel       sender         message sentiment
## 1 2023-08-11 Twitter dave@example    Fun weekend!  Positive
## 2 2023-08-11   Email  @bob_tweets Hello everyone!  Positive
## 3 2023-08-11   Slack  @frank_chat Hello everyone!  Negative
## 4 2023-08-18   Email  @frank_chat    Fun weekend!  Positive
## 5 2023-08-14   Slack  @frank_chat Need assistance  Positive
## 6 2023-08-04   Email @erin_tweets Need assistance  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
df11 <- comm_data %>%
  mutate(length = nchar(message),
         product = sentiment * length) %>%
  arrange(desc(product)) %>%
  select(date, sender, message, length, product)
head(df11)
```

```
##         date       sender         message length  product
## 1 2023-08-16  @frank_chat Hello everyone!     15 14.96403
## 2 2023-08-14 @erin_tweets Hello everyone!     15 14.81748
## 3 2023-08-18 dave@example Hello everyone!     15 14.67330
## 4 2023-08-17 dave@example Hello everyone!     15 14.65342
## 5 2023-08-07  carol_slack Hello everyone!     15 14.60145
## 6 2023-08-06 dave@example Hello everyone!     15 14.52123
```

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 %>%
  mutate(length = nchar(message)) %>%
  group_by(date) %>%
    summarise(total_characters = sum(length)) %>%
    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: First, a single column in the dataset contains multiple variables, e.g. under ‘Percent’, there are percentages and raw numbers.

Second, there are multiple variables in columns, e.g. Employment Status > Age > etc..

To make the dataset Tidy, I would first tidy the numeric data by cleaning the relevant columns, and standardising the unit of measurement throughout. Additionally, I would further segregate the ‘subject’ column into clear variables e.g. Age, Labour Force/Armed Force, etc..