DATA607 LAB3

Farhod Ibragimov

2025-02-13

Normalization

Here I'm loading necessary library:

```
library(tidyverse)
```

1. This code creates an unnormalized form data frame:

```
social_media_mess_table <- data.frame(
  user_ID = c("U001", "U002", "U001", "U003", "U002"),
  username = c("alice_b", "bob_w", "alice_b", "charlie_b", "bob_w"),
  email = c("alice@example.com", "bob@example.com", "alice@example.com", "charlie@example.com", "bob@ex
  name = c("Alice Brown", "Bob White", "Alice Brown", "Charlie Black", "Bob White"),
  hobbie = c("Love traveling", "Coffee addict", "Love traveling", "Fitness fan", "Coffee addict"),
  post_ID = c("P001", "P002", "P003", "P004", "P005"),
  post_content = c("Just visited Paris!", "Morning coffee is the best", "Paris is incredible!", "Just f
  comment_ID = c("C001", "C002", "C003", "C004", "C005"),
  comment_content = c("Looks amazing!", "I agree!", "Awesome post!", "Way to go!", "Sounds fun!"),
  like_ID = c("L001", "L002", "L003", "L004", "L005"),
  liked_post_ID = c("P001", "P002", "P003", "P004", "P005"),
  following_user_IDs = c("U002,U003", "U001,U003", "U002,U003", "U001,U002", "U001,U003"),
  follower_user_IDs = c("U004,U005", "U005,U006", "U004,U005", "U006,U007", "U005,U006")
)

print(social_media_mess_table)
```

```
##   user_ID  username               email          name         hobbie post_ID
## 1    U001   alice_b   alice@example.com   Alice Brown Love traveling    P001
## 2    U002     bob_w     bob@example.com     Bob White  Coffee addict    P002
## 3    U001   alice_b   alice@example.com   Alice Brown Love traveling    P003
## 4    U003 charlie_b charlie@example.com Charlie Black    Fitness fan    P004
## 5    U002     bob_w     bob@example.com     Bob White  Coffee addict    P005
##                    post_content comment_ID comment_content like_ID
## 1             Just visited Paris!       C001  Looks amazing!    L001
## 2    Morning coffee is the best       C002        I agree!    L002
## 3          Paris is incredible!       C003   Awesome post!    L003
## 4 Just finished a great workout       C004      Way to go!    L004
## 5           Coffee with friends       C005     Sounds fun!    L005
##   liked_post_ID following_user_IDs follower_user_IDs
## 1          P001          U002,U003         U004,U005
## 2          P002          U001,U003         U005,U006
## 3          P003          U002,U003         U004,U005
## 4          P004          U001,U002         U006,U007
## 5          P005          U001,U003         U005,U006
```

The reasons why this is unnormalized form (UNF) :

- There is no primary key defined
- "`following_user_IDs`" and "`follower_user_IDs`" columns has multi- value attributes
- It has data inconsistency. If a user changes his email or username, multiple rows needs to be updated.
- Redundancy issues: user information (`user_ID, email, username, name`) repeated several times.

2. This code cell below separates user data into its own table.

```
#users table
users_table <- social_media_mess_table |>
  select(user_ID, username,email, name) |>
  distinct()

print(users_table)
```

```
##   user_ID  username               email          name
## 1    U001   alice_b   alice@example.com   Alice Brown
## 2    U002     bob_w     bob@example.com     Bob White
## 3    U003 charlie_b charlie@example.com Charlie Black
```

3. Separates hobbies into its own table with (`user_ID -> hobbie`) linked

```
#hobbies_table
hobbies_table <- social_media_mess_table |>
  select(user_ID, hobbie)|>
  distinct()

print(hobbies_table)
```

```
##   user_ID         hobbie
## 1    U001 Love traveling
## 2    U002  Coffee addict
## 3    U003    Fitness fan
```

4. Creating table for users posts (`post_id` –> `user_ID` –> `post_content`).

```
#posts_table
posts_table <- social_media_mess_table |>
  select(post_ID, user_ID, post_content) |>
  distinct()

print(posts_table)
```

```
##   post_ID user_ID                 post_content
## 1    P001    U001           Just visited Paris!
## 2    P002    U002    Morning coffee is the best
## 3    P003    U001          Paris is incredible!
## 4    P004    U003 Just finished a great workout
## 5    P005    U002           Coffee with friends
```

5. Creating comments table (`comment_ID -> post_ID``->``user_ID --> comment_content`)

```
#comments_table
comments_table <- social_media_mess_table |>
  select(comment_ID, post_ID, user_ID, comment_content) |>
  distinct()
```

```
print(comments_table)
```

```
##   comment_ID post_ID user_ID comment_content
## 1       C001    P001    U001  Looks amazing!
## 2       C002    P002    U002        I agree!
## 3       C003    P003    U001   Awesome post!
## 4       C004    P004    U003      Way to go!
## 5       C005    P005    U002     Sounds fun!
```

6. Separate likes table (`like_ID -> user_ID -> liked_post_ID`)

```
#likes_table
likes_table <- social_media_mess_table |>
  select(like_ID, user_ID, liked_post_ID)|>
  distinct()|>
  rename(post_ID = liked_post_ID)
print(likes_table)
```

```
##   like_ID user_ID post_ID
## 1    L001    U001    P001
## 2    L002    U002    P002
## 3    L003    U001    P003
## 4    L004    U003    P004
## 5    L005    U002    P005
```

7. Creates separate table for users liked posts (`user_ID``-> liked_post_ID`)

```
liked_posts_table <- social_media_mess_table |>
  select(user_ID, liked_post_ID)|>
  distinct()|>
  rename(post_ID = liked_post_ID)
print(liked_posts_table)
```

```
##   user_ID post_ID
## 1    U001    P001
## 2    U002    P002
## 3    U001    P003
## 4    U003    P004
## 5    U002    P005
```

8. Creates a table for users following other users (`user_ID -> following_user_IDs`). Here `separate_longer_delim(following_user_IDs, delim = ',')` separate multi-value `following_user_IDs` into different cell values and assigns them to unique `user_ID`

```
#print(social_media_mess_table)
user_following_table <- social_media_mess_table |>
  select(user_ID, following_user_IDs) |>
  separate_longer_delim(following_user_IDs, delim = ',')|>
  distinct()
print(user_following_table)
```

```
##   user_ID following_user_IDs
## 1    U001               U002
## 2    U001               U003
## 3    U002               U001
## 4    U002               U003
## 5    U003               U001
```

```
## 6    U003               U002
```

9. This code creates users followers table (`user_ID -> follower_user_IDs`). Here `separate_longer_delim(follower_us delim = ",")` separate multi-value `follower_user_IDs` into different cell values and assigns them to unique `user_ID`

```
user_follower_table <- social_media_mess_table |>
  select(user_ID, follower_user_IDs) |>
  separate_longer_delim(follower_user_IDs, delim = ",")|>
  distinct()
print(user_follower_table)
```

```
##   user_ID follower_user_IDs
## 1    U001              U004
## 2    U001              U005
## 3    U002              U005
## 4    U002              U006
## 5    U003              U006
## 6    U003              U007
```

10. Joining `users_table` with `posts_table`. The `user_posts` table will have one row for each post along with the corresponding user information. It shows the user details alongside their posts.

```
#user_posts
user_posts <- left_join(users_table, posts_table, by = "user_ID")
print(user_posts)
```

```
##   user_ID  username               email          name post_ID
## 1    U001   alice_b   alice@example.com   Alice Brown    P001
## 2    U001   alice_b   alice@example.com   Alice Brown    P003
## 3    U002     bob_w     bob@example.com     Bob White    P002
## 4    U002     bob_w     bob@example.com     Bob White    P005
## 5    U003 charlie_b charlie@example.com Charlie Black    P004
##                   post_content
## 1           Just visited Paris!
## 2          Paris is incredible!
## 3    Morning coffee is the best
## 4           Coffee with friends
## 5 Just finished a great workout
```

11. Joining `posts_table` with `comments_table`. The `posts_comments` table will have information from both tables, including the post content and the associated comments. Each post will show which comment it has.

```
#posts_comments
posts_comments <- left_join(posts_table, comments_table, by = c("post_ID", "user_ID"))
print(posts_comments)
```

```
##   post_ID user_ID                 post_content comment_ID comment_content
## 1    P001    U001           Just visited Paris!       C001  Looks amazing!
## 2    P002    U002    Morning coffee is the best       C002         I agree!
## 3    P003    U001          Paris is incredible!       C003   Awesome post!
## 4    P004    U003 Just finished a great workout       C004      Way to go!
## 5    P005    U002           Coffee with friends       C005     Sounds fun!
```

12. Joining `posts_table` with `likes_table`. The `posts_likes` table will show the posts and the users who liked them.

```r
posts_likes <- left_join(posts_table, likes_table, by = c("post_ID", "user_ID"))
print(posts_likes)
```

```
##   post_ID user_ID                 post_content like_ID
## 1    P001    U001           Just visited Paris!    L001
## 2    P002    U002    Morning coffee is the best    L002
## 3    P003    U001          Paris is incredible!    L003
## 4    P004    U003 Just finished a great workout    L004
## 5    P005    U002           Coffee with friends    L005
```

13. Printing all tables:

```r
print(users_table)
```

```
##   user_ID  username               email          name
## 1    U001   alice_b   alice@example.com   Alice Brown
## 2    U002     bob_w     bob@example.com     Bob White
## 3    U003 charlie_b charlie@example.com Charlie Black
```

```r
print(hobbies_table)
```

```
##   user_ID         hobbie
## 1    U001 Love traveling
## 2    U002  Coffee addict
## 3    U003    Fitness fan
```

```r
print(posts_table)
```

```
##   post_ID user_ID                 post_content
## 1    P001    U001           Just visited Paris!
## 2    P002    U002    Morning coffee is the best
## 3    P003    U001          Paris is incredible!
## 4    P004    U003 Just finished a great workout
## 5    P005    U002           Coffee with friends
```

```r
print(comments_table)
```

```
##   comment_ID post_ID user_ID comment_content
## 1       C001    P001    U001  Looks amazing!
## 2       C002    P002    U002        I agree!
## 3       C003    P003    U001   Awesome post!
## 4       C004    P004    U003      Way to go!
## 5       C005    P005    U002     Sounds fun!
```

```r
print(likes_table)
```

```
##   like_ID user_ID post_ID
## 1    L001    U001    P001
## 2    L002    U002    P002
## 3    L003    U001    P003
## 4    L004    U003    P004
## 5    L005    U002    P005
```

```r
print(user_following_table)
```

```
##   user_ID following_user_IDs
## 1    U001               U002
## 2    U001               U003
## 3    U002               U001
```

```
## 4    U002               U003
## 5    U003               U001
## 6    U003               U002
```

```
print(user_follower_table)
```

```
##   user_ID follower_user_IDs
## 1    U001              U004
## 2    U001              U005
## 3    U002              U005
## 4    U002              U006
## 5    U003              U006
## 6    U003              U007
```

Conclusions:

- This code normalizes a messy social media dataset into structured tables, reducing redundancy and ensuring data integrity. For example, selecting the relevant columns (`post_id` –> `user_ID` –> `post_content`), this code creates a new table (`user_posts`) that stores each post linked to a unique user and the content they posted. We can track which user made which post while avoiding redundancy.
- By organizing users, posts, comments, likes, and follower relationships into separate tables, it achieves *Third Normal Form (3NF)* by eliminating partial and transitive dependencies.
- Relationships such as one-to-many (users –> posts, posts –> comments) and many-to-many (users <–> likes, users <–> followers) are structured.

Character Manipulation

1. Using the 173 majors listed in fivethirtyeight.com's College Majors dataset [https://fivethirtyeight.com/features/the-economic-guide-to-picking-a-college-major/], provide code that identifies the majors that contain either "DATA" or "STATISTICS" This code cell downloads `majors-list.csv`file from URL "https://raw.githubusercontent.com/fivethirtyeight/data/master/college-majors/majors-list.csv" and loads it into `majors_ds` dataframe:

```
url <- "https://raw.githubusercontent.com/fivethirtyeight/data/master/college-majors/majors-list.csv"
majors_ds <- read.csv(url, stringsAsFactors = FALSE)
```

This code filters the `majors_ds` dataframe to find rows where the `Major` column contains the words **"DATA"** or **"STATISTICS"** (case-insensitive). Here `str_detect(Major, regex("DATA", ignore_case = TRUE)) | str_detect(Major, regex("STATISTICS", ignore_case = TRUE))` checks if Major column contains "DATA" or (|) "STATISTICS" words. This code saves matched rows in `identify_majors` dataframe.

```
identify_majors <- majors_ds |>
  filter(
    str_detect(Major, regex("DATA", ignore_case = TRUE)) |
    str_detect(Major, regex("STATISTICS", ignore_case = TRUE))
  )
print(identify_majors)
```

```
##   FOD1P                                     Major          Major_Category
## 1  6212 MANAGEMENT INFORMATION SYSTEMS AND STATISTICS                Business
## 2  2101      COMPUTER PROGRAMMING AND DATA PROCESSING Computers & Mathematics
## 3  3702               STATISTICS AND DECISION SCIENCE Computers & Mathematics
```

2. Describe, in words, what these expressions will match:

- `(.)\1\1`

Regex `(.)\1\1` in this format is not gonna work, because of single (\) backlashes. When R sees `\1`, it interpret it as a special escape sequence, not as a back reference. It has to be double (\\) backlashes. Here we use `(.)\1\1` in a code cell below and it is not gonna work:

```
x <- c("aaa", "GGG", "aan", "???", "222", "334")
str_extract_all(x, "(.)\1\1")
```

```
## [[1]]
## character(0)
##
## [[2]]
## character(0)
##
## [[3]]
## character(0)
##
## [[4]]
## character(0)
##
## [[5]]
## character(0)
##
## [[6]]
## character(0)
```

So let me explain this correct regex format "``(.)\\1\\1"

This regex "(.)\\1\\1"looks for three identical characters in a row.

(.) creates a one capturing group, where dot . is any first character in the string.

\\1 is a back reference. It is checking if second character of the row matches to whatever was captured in first group (.)

\\1\\1 repeats the backreference twice, requiring two more identical characters.

Here is a code example of how regex (.)\\1\\1 works:

```
x <- c("aaa", "GGG", "aan", "???", "222", "334" , "aaab")
str_extract_all(x, "(.)\\1\\1")
```

```
## [[1]]
## [1] "aaa"
##
## [[2]]
## [1] "GGG"
##
## [[3]]
## character(0)
##
## [[4]]
## [1] "???"
##
## [[5]]
## [1] "222"
##
## [[6]]
## character(0)
##
## [[7]]
## [1] "aaa"
```

3. Describe, in words, what these expressions will match:

- "(.)(.)\\2\\1"

This regex "(.)(.)\\2\\1" looks for a pattern where two characters are repeated in reverse order.

(.)(.)creates two separate capturing groups from first two characters of the string. Let's call them group #1 for the first (.) and group#2 for second (.)

\\2 is a back reference for group #2 and checks if third character of the string matches the captured character in group #2.

\\1is a back reference for group #1 and checks if fourth character of the string matches the captured character in group #1.

Here is a code example of how regex "(.)(.)\\2\\1" works:

```
x <- c("azza", "FEEF", "assd", "1221", "!??!", "#@@&")
str_extract_all(x, "(.)(.)\\2\\1")
```

```
## [[1]]
## [1] "azza"
##
## [[2]]
```

```
## [1] "FEEF"
##
## [[3]]
## character(0)
##
## [[4]]
## [1] "1221"
##
## [[5]]
## [1] "!??!"
##
## [[6]]
## character(0)
```

4. Describe, in words, what these expressions will match:

- `(..)\1`

Regex `(..)\1`is not gonna work because of the single backlash.

Correct regex is `"(..)\\1"`.

`(..)` creates one capturing group from first two letter of the string.

`\\1` is a reference back to the first captured group `(..)` and checks if third and fourth characters of the string matching to the group.

Here is a code example of how regex `"(..)\1"` works:

```
x <- c("fafa", "KLKL", "fafe", "cdcdf", "1212", "1213", "@#@#", "@#@%")
str_extract_all(x, "(..)\\1")
```

```
## [[1]]
## [1] "fafa"
##
## [[2]]
## [1] "KLKL"
##
## [[3]]
## character(0)
##
## [[4]]
## [1] "cdcd"
##
## [[5]]
## [1] "1212"
##
## [[6]]
## character(0)
##
## [[7]]
## [1] "@#@#"
##
## [[8]]
## character(0)
```

5. Describe, in words, what this expression will match: `"(.).\\1.\\1"`

`"(.).\\1.\\1"` regex looks for if first character repeated in third and fifth characters of the string. Second and fourth characters can be any characters.

Here is a code example of how regex `"(.).\\1.\\1"`works:

```
x <- c("acava", "ACAVA", "acave", "13151", "13152", "!@!%!", "!@!%&")
str_extract_all(x, "(.).\\1.\\1")
```

```
## [[1]]
## [1] "acava"
##
## [[2]]
## [1] "ACAVA"
##
## [[3]]
## character(0)
##
## [[4]]
## [1] "13151"
##
## [[5]]
## character(0)
##
## [[6]]
## [1] "!@!%!"
##
## [[7]]
## character(0)
```

6. Describe, in words, what this expression will match: `"(.)(.)(.).*\\3\\2\\1"`

`"(.)(.)(.).*\\3\\2\\1"`looks for first three characters at the beginning, any characters in the middle , and then those same three characters at the end, but in reverse order.

`.*` means any amount of any characters

`\\3\\2\\1` referencing to third, second and first captured groups.

Here is a code example of how regex `"(.)(.)(.).*\\3\\2\\1"` works:

```
x <- c("abchjlcba", "DFGjhrGFD", "DFGjhrGFA", "123jhkk321", "#@%345%@#", "#@%345%@?")
str_view(x, "(.)(.)(.).*\\3\\2\\1")
```

```
## [1] | <abchjlcba>
## [2] | <DFGjhrGFD>
## [4] | <123jhkk321>
## [5] | <#@%345%@#>
```

7. Construct regular expressions to match words that:

- Start and end with the same character.

```
x <- c("abca", "MADAM", "DjhrA", "1jhkk1", "#345#", "#345?")
str_view(x, "^(.).*\\1$")
```

```
## [1] | <abca>
## [2] | <MADAM>
## [4] | <1jhkk1>
## [5] | <#345#>
```

- Contain a repeated pair of letters (e.g. “church” contains “ch” repeated twice.)

```
x <- c("church", "MAMA", "GGlkGGff", "GGlkGLff")
str_view(x, "(..).*\\1\\.*")
```

```
## [1] | <church>
## [2] | <MAMA>
## [3] | <GGlkGG>ff
```

- Contain one letter repeated in at least three places (e.g. “eleven” contains three “e”s.)

```
x <- c("abcafgajk", "AJHAIKAU", "abcath", "13615613")
str_view(x, "(.).*\\1.*\\1")
```

```
## [1] | <abcafga>jk
## [2] | <AJHAIKA>U
## [4] | <1361561>3
```