Data607-Week_Two-Assignment

Shri Tripathi

09/18/2024

Week 3 Assignment

Question - 1

Provide an example of at least three dataframes in R that demonstrate normalization. The dataframes can contain any data, either real or synthetic. Although normalization is typically done in SQL and relational databases, you are expected to show this example in R, as it is our main work environment in this course.

Solution

```r
# Install and load the necessary packages in one step
# Check if the 'dplyr' and 'tidyverse' packages are installed; if not, install them
if (!requireNamespace("dplyr", quietly = TRUE)) {
    install.packages("dplyr")
}
if (!requireNamespace("tidyverse", quietly = TRUE)) {
    install.packages("tidyverse")
}

# Load the necessary libraries
library(dplyr)       # Load dplyr for data manipulation and %>%
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tidyverse)   # Load tidyverse, which includes useful packages like dplyr
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v readr     2.1.5
## v ggplot2   3.4.4     v stringr   1.5.1
## v lubridate 1.9.3     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
# Create the 'employees' dataframe, which contains employee information
employees <- data.frame(
  employee_id = c(1, 2, 3, 4),                           # Employee IDs
  employee_name = c("John Doe", "Jane Smith",             # Employee names
                    "Emily Davis", "Michael Brown"),
  hire_date = as.Date(c("2020-01-15", "2019-07-30",       # Hire dates for employees
                        "2021-02-22", "2022-06-10"))
)

# Create the 'departments' dataframe, which contains department information
departments <- data.frame(
  department_id = c(1, 2, 3),                            # Department IDs
  department_name = c("HR", "IT", "Marketing")           # Department names
)

# Create the 'employee_departments' dataframe, which links employees to departments
employee_departments <- data.frame(
  employee_id = c(1, 2, 3, 4),                           # Employee IDs linking to employees table
  department_id = c(1, 2, 2, 3)                          # Department IDs linking to departments table
)

# Print each dataframe separately
print("Employees Dataframe:")
```

```
## [1] "Employees Dataframe:"
```

```
print(employees)
```

```
##   employee_id employee_name  hire_date
## 1           1      John Doe 2020-01-15
## 2           2    Jane Smith 2019-07-30
## 3           3   Emily Davis 2021-02-22
## 4           4 Michael Brown 2022-06-10
```

```
print("Departments Dataframe:")
```

```
## [1] "Departments Dataframe:"
```

```
print(departments)
```

```
##   department_id department_name
## 1             1              HR
## 2             2              IT
## 3             3       Marketing
```

```r
print("Employee_Departments Dataframe:")
```

```
## [1] "Employee_Departments Dataframe:"
```

```r
print(employee_departments)
```

```
##   employee_id department_id
## 1           1             1
## 2           2             2
## 3           3             2
## 4           4             3
```

```r
# Join the dataframes using left_join and the pipe (%>%) operator
# First, join 'employee_departments' with 'employees' by 'employee_id'
# Then, join the result with 'departments' by 'department_id'
merged_data <- employee_departments %>%
  left_join(employees, by = "employee_id") %>%
  left_join(departments, by = "department_id")

# Print the final merged dataframe
print("Merged Dataframe:")
```

```
## [1] "Merged Dataframe:"
```

```r
print(merged_data)
```

```
##   employee_id department_id employee_name  hire_date department_name
## 1           1             1      John Doe 2020-01-15              HR
## 2           2             2    Jane Smith 2019-07-30              IT
## 3           3             2   Emily Davis 2021-02-22              IT
## 4           4             3 Michael Brown 2022-06-10       Marketing
```

Breakdown of the Normalization:

1. Employees Table:

Contains unique information about each employee (Employee ID, Name, Hire Date). Does not store department information directly to avoid redundancy.

2. Departments Table:

Contains unique information about each department (Department ID, Department Name). This table is independent of employees.

3. Employee_Departments Table:

This table creates the relationship between employees and departments, demonstrating a many-to-one relationship where each employee is assigned to a department.

Question - 2

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"

Solution

First, I imported the CSV file directly from the raw GitHub URL and stored the data in a dataframe called `majors`. Then, I loaded the necessary libraries, `dplyr` and `tidyverse`, to facilitate data manipulation. After that, I used the `filter` function to apply a regular expression that filtered out the majors containing either 'DATA' or 'STATISTICS'. The result was saved in a new dataframe named `infosci_maj`.

```r
#Importing the majors data

library(readr)

majors=read.csv(url("https://raw.githubusercontent.com/fivethirtyeight/data/master/college-majors/majors

names(majors)
```

```
## [1] "FOD1P"          "Major"          "Major_Category"
```

```r
#loading, tidyverse, dplyr
library(dplyr)
library(tidyverse)

#filtering out only DATA or STATISTICS using regex

infosci_maj = majors %>%
  filter(grepl("(?=.*DATA)|(?=.*STATISTICS)", Major, perl = TRUE))

head(infosci_maj)
```

```
##   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
```

Question - 3

Describe, in words, what these expressions will match:

1. (.)\1\1
2. "(.)(.)\\2\\1"
3. (..)\1
4. "(.).\\1.\\1"
5. "(.)(.)(.).*\\3\\2\\1"

Solution

1. This regular expression matches any character that is repeated three times in a row, such as 'zzz', '555', or 'AAAinsurance'.
2. This regular expression matches two characters followed by the same two characters in reverse order. For example, it would match 'toot', but not 'ttoo'.
3. This regular expression matches any two characters that are repeated immediately, like '2121' or 'momo'.
4. This one is a bit more complex. For instance, if you have three characters like 'e', 'm', and 'w', it will match sequences such as 'emewe'. The same pattern can apply to other letters, such as 'tmtwt' or '42484'. The key is that the first character repeats in positions 1, 3, and 5.
5. This regular expression matches any sequence of three characters followed by any characters, and then the same three characters in reverse order. For example, it would match '123…anything…321' or 'xyz…zyx'. It can be particularly useful for matching anagrams, like 'racecar…racecar'.

Question - 4

Solution

Construct regular expressions to match words that:

- Start and end with the same character.
- Contain a repeated pair of letters (e.g. "church" contains "ch" repeated twice.)
- Contain one letter repeated in at least three places (e.g. "eleven" contains three "e"s.)

Solution

To tackle this problem, I first created a sample text containing a variety of U.S. city names, ensuring there were examples that represented each condition specified, along with some additional values for extra coverage.

Next, I processed the city names by splitting the text into a vector, `city_vector`, using `str_split` to divide the cities by spaces. After the split, I removed any empty strings that may have appeared during the process.

Filtering Cities That Start and End with the Same Character

To identify cities where the first and last characters are the same, I first converted all city names in `city_vector` to lowercase. Then, I applied `str_detect` with the regular expression "^(.).*\\1$". This regex filters cities where the first and last characters are identical. Finally, I printed the results.

Filtering Cities with Repeated Pairs of Characters

Similarly, to find cities that contain a repeated pair of characters, I used `str_detect` with the regular expression "(.).*\\1". This pattern identifies cities where a pair of characters appears more than once in the string. The filtered results were printed afterward.

Filtering Cities with a Character Repeated Three Times

For the last example, I applied the regex "(.).*\\1.*\\1" using `str_detect`, which matches cities that contain a single character repeated at least three times within the string. These results were also printed for verification.

This method effectively identifies cities that meet the specified conditions through regular expressions and string manipulation functions in R.

```r
# Define a string of city names for pattern matching
cities = "NewYork Chicago Alameda LosAngeles Mississippi Houston Miami Philadelphia Elma Atlanta Dallas

# Split the city names into individual elements stored in a vector
city_vector = str_split(cities, " ")[[1]]

# Remove any empty elements from the vector
city_vector = city_vector[city_vector != ""]

# Identify cities that start and end with the same character
abba = city_vector[str_detect(tolower(city_vector), "^(.).*\\1$")]
print(paste("Cities that start and end with the same character:", paste(abba, collapse = ", ")))
```

```
## [1] "Cities that start and end with the same character: Alameda, Atlanta, Aurora"
```

```r
# Identify cities that contain a repeated pair of characters
aa = city_vector[str_detect(city_vector, "(.).*\\1")]
print(paste("Cities that contain a repeated pair of letters:", paste(aa, collapse = ", ")))
```

```
## [1] "Cities that contain a repeated pair of letters: Alameda, LosAngeles, Mississippi, Houston, Miam
```

```r
# Identify cities that contain a single character repeated three times
aaa = city_vector[str_detect(city_vector, "(.).*\\1.*\\1")]
print(paste("Cities that contain one letter repeated in at least three places:", paste(aaa, collapse =
```

```
## [1] "Cities that contain one letter repeated in at least three places: Mississippi, Tallahassee, Cin
```