Week 2A Pricilla Nakyazze Assignment – SQL and R

2025-09-03

#{r setup, include=FALSE} #knitr::opts_chunk$set(echo = TRUE) #

```
library(RSQLite)
library(DBI)
library(sqldf)
```

```
## Loading required package: gsubfn
```

```
## Loading required package: proto
```

```
library(RODBC)
library(odbc)
library(crayon)
library(dplyr)
```

```
##
## 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
```

```
library(RPostgres)
```

Choose six recent popular movies. Ask at least five imaginary friends to rate each of these movies that they have seen on a scale of 1 to 5. Take the results (observations) and store them in a SQL database of your choosing.Load the information from the SQL database into an R dataframe. I downloaded the PostgreSQL Ansi driver, and using ODBC data source Administrator created the User DSN Post to access the PGadmin Tables.

```
# Connect to the database
con <- dbConnect(odbc::odbc(), "Post")


# Run your query
movies2 <- dbGetQuery(con, "SELECT * FROM movietable;")

movies3 <- dbGetQuery(con, "SELECT * FROM MovieID;")
```

```
dfMovies <- movies2
dfMovies
```

```
##   id   name elio how_to_train_your_dragon f1_the_movie superman
## 1  1 Daniel    2                        4            5        5
## 2  2 Eleana    5                        5            3        5
## 3  3  Susan    4                        3         <NA>        5
## 4  4 Winnie    3                        1         <NA>        5
## 5  5  Aiden <NA>                        4            5        5
##   mission_impossible
## 1                  3
## 2                  4
## 3                  5
## 4                  2
## 5                  1
```

Load the information from the SQL database into an R dataframe.

```
Movie_detail <- movies3
Movie_detail
```

```
##     id               movietitle released1
## 1 1143                     Elio      2025
## 2 2945 How to Train your Dragon      2025
## 3 4983             F1 the movie      2025
## 4 8100                 Superman      2025
## 5 5233       Mission Impossible      2025
```

Handling missing data is a foundational skill when working with SQL or R. To receive full credit, you should demonstrate a reasonable approach for handling missing data. After all, how likely is it that all five of your friends have seen all six movies?

To calculate the mean of a vector in R that contains missing values (represented as NA), you need to handle them explicitly, or R will return NA by default. You can do this using the na.rm argument.

Using na.rm NA demonstration. By Calculating Susan's mean rating.

The mean calculation for 'Susan' will result in NA because of the missing value in the 'f1__the__movie' column.

```
mean_susan_with_na <- mean(c(dfMovies$elio[3], dfMovies$how_to_train_your_dragon[3], dfMovies$f1_the_mo
```

```
## Warning in mean.default(c(dfMovies$elio[3],
## dfMovies$how_to_train_your_dragon[3], : argument is not numeric or logical:
## returning NA
```

```
print(mean_susan_with_na)
```

```
## [1] NA
```

```r
dfMovies <- dfMovies %>%
  mutate(across(everything(), ~na_if(., "NA")))
```

Correctly calculate the mean using na.rm = TRUE # The `na.rm = TRUE` argument tells R to remove NA values before computing the mean. ‘

```r
mean_susan_correct <- mean(
  as.numeric(c(
    dfMovies$elio[3],
    dfMovies$how_to_train_your_dragon[3],
    dfMovies$f1_the_movie[3],
    dfMovies$superman[3]
  )),
  na.rm = TRUE
)

print(mean_susan_correct)
```

```
## [1] 4
```

Is there any benefit in standardizing ratings? How might you approach this? Z-score standardization (mean = 0, sd = 1)

Its Useful when you want to normalize values so they follow a standard normal distribution. This process, often called Z score normalization, transforms data to have a mean of 0 and a standard deviation of 1. Normalization can help with outlier detection. Values that are significantly far from the mean (e.g., Z-scores above 3 or below -3) are more easily identified as potential outliers.

The superman column has a scaled:scale of 0 and all NaN values. This happened because the original data had no variation. All the ratings were the exact same. Since the standard deviation was zero, the scale() function couldn’t perform the division, resulting in NaN (Not a Number) values.

```r
# Convert "NA" strings to actual NA values
dfMovies[dfMovies == "NA"] <- NA

#  Identify numeric columns
numeric_cols <- sapply(dfMovies, is.numeric)

#  Standardize numeric columns only
dfMovies_scaled <- dfMovies  # Make a copy

dfMovies_scaled[numeric_cols] <- lapply(dfMovies[numeric_cols], scale)
```

```r
str(dfMovies_scaled)
```

```
## 'data.frame':    5 obs. of  7 variables:
##  $ id                     : chr  "1" "2" "3" "4" ...
##  $ name                   : chr  "Daniel" "Eleana" "Susan" "Winnie" ...
##  $ elio                   : chr  "2" "5" "4" "3" ...
##  $ how_to_train_your_dragon: chr  "4" "5" "3" "1" ...
##  $ f1_the_movie           : chr  "5" "3" NA NA ...
##  $ superman               : chr  "5" "5" "5" "5" ...
##  $ mission_impossible     : chr  "3" "4" "5" "2" ...
```