Data Science Project

Problem Description

The task includes developing a data analysis and visualization program using R.

Question (1): What will the program do?

The program should allow users to upload a dataset in its raw form to undergo the various processes of applied data analytics and visualization along with associative calculations to later display results interactively and explicitly through a web application. Each team member has specific roles in the development process to ensure the success of the project.

Question (2): What the input to the program will be?

A file provided by the user in the form of a **.csv** file containing customer names, their individual ages, the total amounts spent by each customer, their city of residence, and their preferred payment type. The user inputs the number of clusters they want to be displayed, the minimum support, and the minimum confidence to be calculated.

Question (3): What the output of the program will be?

The output shall be a clean, organized version of the dataset's contents, meaning all duplicated and missing data are removed from the set. The results should be displayed in three different forms of visualized graphs for easy comprehension of the data. The k-means clustering result should be displayed in a sorted table. Similarly, the outcomes of the association rules will be correctly calculated and displayed.

Team member roles:

- **Cleaning data:**

  This role is responsible for identifying and taking note of any issues in the raw dataset. This may involve tasks such as removing duplicates, handling missing values, and correcting errors. The responsible team member will use various R packages and functions to perform data cleaning tasks efficiently. The goal is to prepare the data for further analysis by ensuring its quality.

- **Data visualization:**

  The role of a team member responsible for data visualization is essential for translating the attained data analysis findings into insights that effectively communicate through engaging and informative visualizations.

- **GUI code:**

  The team member responsible for GUI code plays a significant role in creating interfaces that enhance user engagement and satisfaction in software applications. Their work connects the gap between technical functions and user experience which is a major contribution to the success of the project.

- **K-means clustering:**

  The team members work closely with data scientists and domain experts to define the problem that K-means clustering aims to solve.

They identify relevant variables and preprocess the data as needed to prepare it for clustering analysis.

- **Association rules:**

  The team member responsible for Association Rules analyzes transactional data to identify frequent item sets and generate association rules.
  They use algorithms such as **Apriori**.

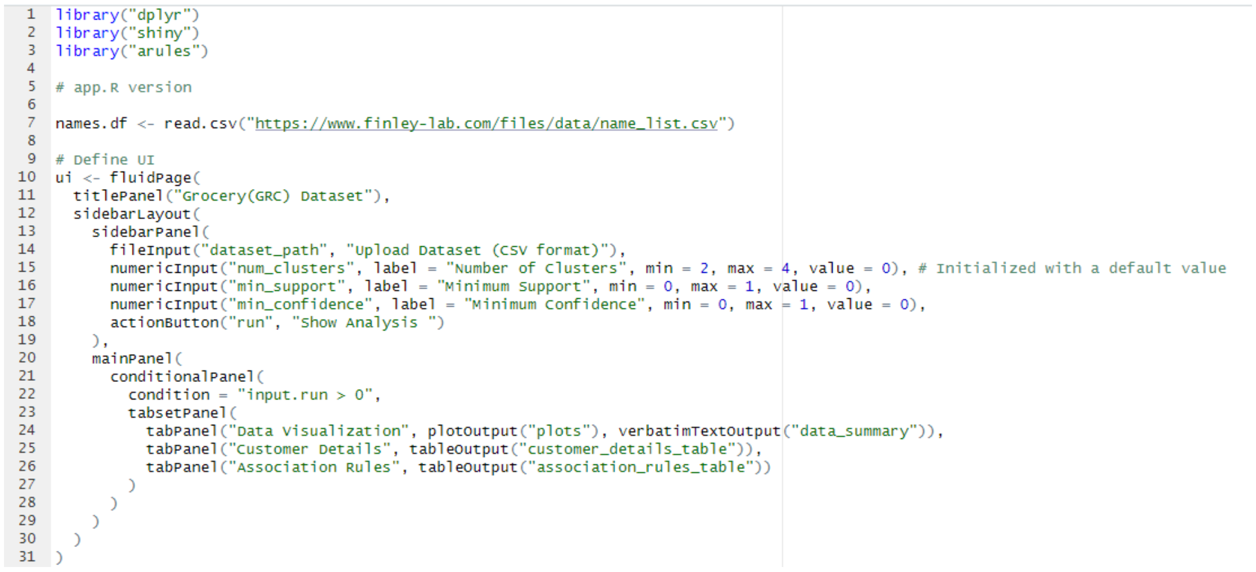
- **Report:**

  The team member responsible for reporting creates clear, concise, and visually appealing reports or presentations that contain key insights about the entire project process and gives functional recommendations.

Dataset description:

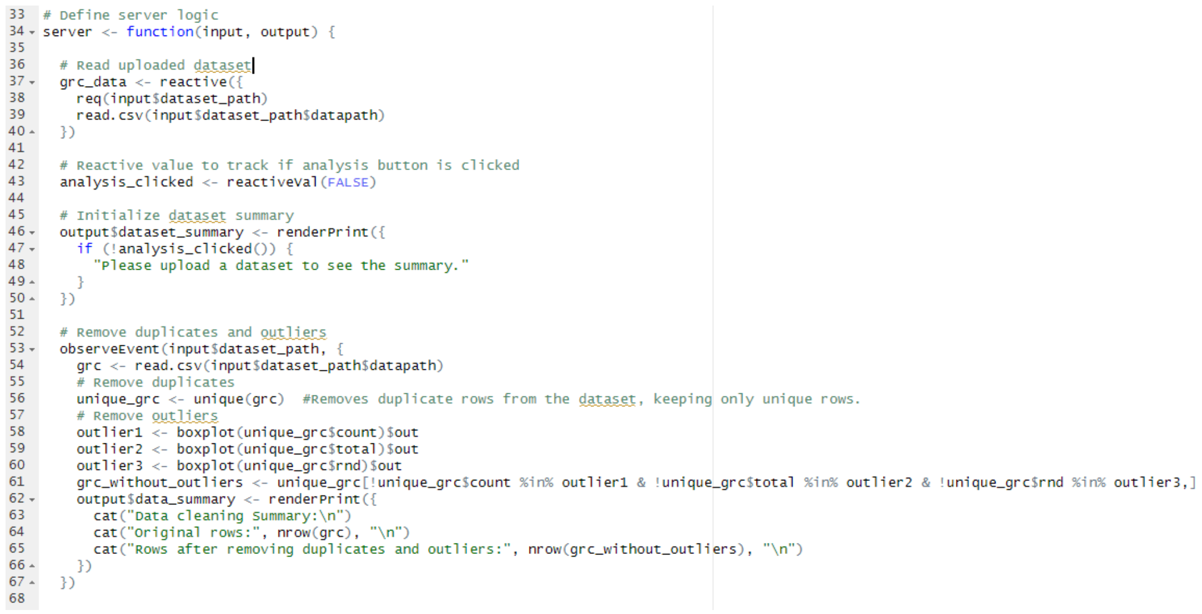
The dataset used in the project, named **"grc.csv",** contains information about transactions, including customer details such as age, city, payment method, and total amounts spent. The dataset will be used for various analyses and visualizations to understand patterns and relationships within the data. It will be used to display the purchase details of customers and the rates of their choices in payment methods and the average of payments, along with their choice of products. It is also used for k-means clustering and calculating the association rules (support and confidence) based on the data.

Screenshots from project steps:



```r
library("dplyr")
library("shiny")
library("arules")

# app.R version

names.df <- read.csv("https://www.finley-lab.com/files/data/name_list.csv")

# Define UI
ui <- fluidPage(
  titlePanel("Grocery(GRC) Dataset"),
  sidebarLayout(
    sidebarPanel(
      fileInput("dataset_path", "Upload Dataset (CSV format)"),
      numericInput("num_clusters", label = "Number of Clusters", min = 2, max = 4, value = 0), # Initialized with a default value
      numericInput("min_support", label = "Minimum Support", min = 0, max = 1, value = 0),
      numericInput("min_confidence", label = "Minimum Confidence", min = 0, max = 1, value = 0),
      actionButton("run", "Show Analysis ")
    ),
    mainPanel(
      conditionalPanel(
        condition = "input.run > 0",
        tabsetPanel(
          tabPanel("Data Visualization", plotOutput("plots"), verbatimTextOutput("data_summary")),
          tabPanel("Customer Details", tableOutput("customer_details_table")),
          tabPanel("Association Rules", tableOutput("association_rules_table"))
        )
      )
    )
  )
)
```

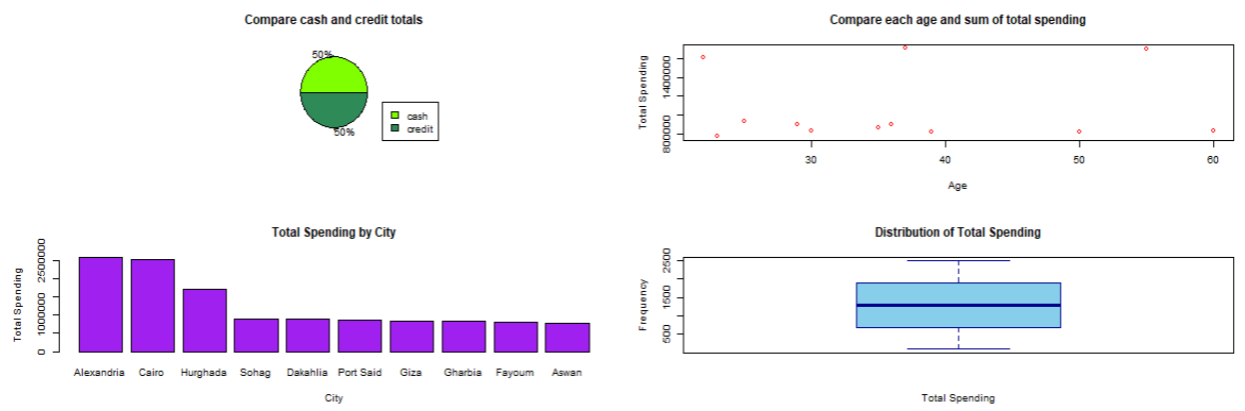
- We initiated the code by downloading several R packages: dplyr, shiny and arules.
- We applied the **read.csv ()** function to read a CSV file named "name_list.csv" from a specified URL into a data frame named names.df.
- Then, the UI is defined using the **fluidPage ()** function, which creates a responsive layout for the web application. Inside the UI, there is a title panel **("Grocery (GRC) Dataset")** and a sidebar layout with sidebar panels for file input **(fileInput ()),** numeric inputs **(numericInput ()),** and an action button **(actionButton ()).**
- The main panel contains **(tabsetPanel ())** with three-tab panels: "Data Visualization", "Customer Details", and "Association Rules".
- Each tab panel contains several types of output elements such as plots, tables, and text outputs.



```r
# Define server logic
server <- function(input, output) {

  # Read uploaded dataset
  grc_data <- reactive({
    req(input$dataset_path)
    read.csv(input$dataset_path$datapath)
  })

  # Reactive value to track if analysis button is clicked
  analysis_clicked <- reactiveVal(FALSE)

  # Initialize dataset summary
  output$dataset_summary <- renderPrint({
    if (!analysis_clicked()) {
      "Please upload a dataset to see the summary."
    }
  })

  # Remove duplicates and outliers
  observeEvent(input$dataset_path, {
    grc <- read.csv(input$dataset_path$datapath)
    # Remove duplicates
    unique_grc <- unique(grc)  #Removes duplicate rows from the dataset, keeping only unique rows.
    # Remove outliers
    outlier1 <- boxplot(unique_grc$count)$out
    outlier2 <- boxplot(unique_grc$total)$out
    outlier3 <- boxplot(unique_grc$rnd)$out
    grc_without_outliers <- unique_grc[!unique_grc$count %in% outlier1 & !unique_grc$total %in% outlier2 & !unique_grc$rnd %in% outlier3,]
    output$data_summary <- renderPrint({
      cat("Data cleaning Summary:\n")
      cat("Original rows:", nrow(grc), "\n")
      cat("Rows after removing duplicates and outliers:", nrow(grc_without_outliers), "\n")
    })
  })

```

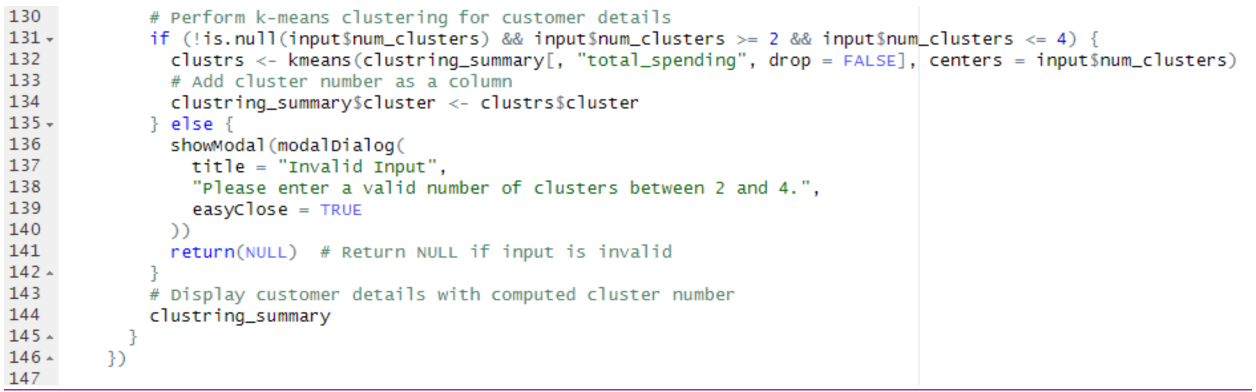
- The expression **(grc_data)** reads the uploaded dataset whenever the **input$dataset_path** changes. It uses the **req ()** function to ensure that the dataset path is not null before reading the data.
- A reactive value **(analysis_clicked) is initiated to 'FALSE'** using **'reactiveVal () '** .this track whether analysis button is clicked or not.
- The **output$dataset_summary** is rendered using the **renderPrint ()** function. It displays a message that asks the user to upload a dataset.
- Duplicates are removed from the dataset using the **unique ()** function and outliers are removed separately for each column (count, total, and rnd) using the **boxplot ()** function.
- The summary information about the original and cleaned dataset is rendered using **'renderPrint ()'** in the **'output$data_summary.'**
- Overall, this snippet ensures that the dataset is properly accessed, cleaned, and summarized before further analysis.

Explaining the results and insights:



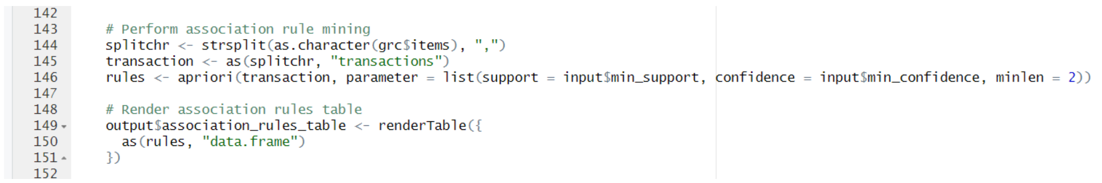
- **Distribution of Total Spending:**
  The box plot shows the distribution of total spending, indicating potential outliers and the spread of spending between customers.
- **Comparison of Cash and Credit Totals:**
  The pie chart compares cash and credit totals, displaying the proportion of each payment type in transactions.
- **Spending Patterns by Age and City:**
  The scatter plot explains the relationship between age and total spending, while the bar plot shows total spending by city, providing insights into geographical spending patterns.
- **Clustering Analysis:**
  The table shows clustering results, identifying customer segments based on total spending.
- **Association Rule**:
  The table presents association rules which illustrate frequent item sets and interconnected patterns in transaction data.

Discussing code parts



```r
      # Perform k-means clustering for customer details
      if (!is.null(input$num_clusters) && input$num_clusters >= 2 && input$num_clusters <= 4) {
        clustrs <- kmeans(clustring_summary[, "total_spending", drop = FALSE], centers = input$num_clusters)
        # Add cluster number as a column
        clustring_summary$cluster <- clustrs$cluster
      } else {
        showModal(modalDialog(
          title = "Invalid Input",
          "Please enter a valid number of clusters between 2 and 4.",
          easyClose = TRUE
        ))
        return(NULL)  # Return NULL if input is invalid
      }
      # Display customer details with computed cluster number
      clustring_summary
    }
  })
```

- The **kmeans()** function is used to perform k-means clustering. It takes the 'age' and 'total' columns from the grc dataset as input. The **centers** parameter specifies the number of clusters to create.
- It checks if the **num_clusters** value is not null and is between 2 and 4. If the input is valid, it performs k-means clustering on the 'total' column. It adds a new column 'cluster'.
- It returns NULL if the input is invalid.



```r
# Perform association rule mining
splitchr <- strsplit(as.character(grc$items), ",")
transaction <- as(splitchr, "transactions")
rules <- apriori(transaction, parameter = list(support = input$min_support, confidence = input$min_confidence, minlen = 2))

# Render association rules table
output$association_rules_table <- renderTable({
  as(rules, "data.frame")
})
```

- The **strsplit()** function splits each list of items into individual items. The **as()** function converts the split items into a format for association rule mining.
- The **apriori()** function performs association rule mining on the transactions.

- The **support** parameter specifies the minimum support for an itemset to be considered frequent. The **confidence** parameter specifies the minimum confidence. The **minlen** parameter specifies the minimum length of the rules.
- The association rules taken from the **apriori()** function are converted into a data frame using **as(rules, "data.frame").**
- The resulting data frame is rendered as a table using **renderTable().**