Datasets Analysis and Visualization Report

Group 9

Prepared for

Dr/ Magda Madbouly

Professor, Faculty of Computing And

Data science Alexandria university

By

Name	ID
Maryam Ibrahim	23012132
Mesk Khaled	23011537
Judy Ahmed	23011244
Asmaa Mahmoud	23011194
Basmala Moataz	23012111

May 2024

Introduction:

This report presents our findings from analyzing the Grocery (GRC) dataset. Our goal is to uncover patterns in grocery shopping behavior. By examining the details of the purchasing processes, customers, and product types; we aim to provide insights that can assist grocery stores in making informed decisions about their inventory and marketing strategies.

Problem Description:

Our project's primary goal was to simplify the Grocery (GRC) dataset, uncovering trends and patterns to inform marketing strategies and optimize product offerings. And in our way to do that, we aimed to answer the following questions:

a. What will the program do?

The program will conduct exploratory data analysis on the Grocery (GRC) dataset, revealing insights into customer behavior and purchasing patterns. It will :

1. Assess and Clean Data:

Data will be assessed and cleaned to ensure its quality and consistency for analysis.

2. Data Visualization:

In the visualization part of the project, various techniques are employed to analyze the Grocery (GRC) dataset. These include comparing cash and credit, assessing the relationship between age groups and total spending, displaying city-wise spending patterns, and visualizing the distribution of total. These visualizations aim to offer stakeholders a simplified view of the data insights for informed decision-making in the grocery retail domain.

3. Customer Segmentation:

Employ a clustering method (using k-means clustering) to split customers into groups based on their total spending and ages. Generate a table displaying each customer's name, age, total spending, and computed cluster number; and visualize them in a plot to simplify understanding of the assignments of customers to each cluster.

4. Association Rule Generation:

Utilize association rule mining algorithms (using the Apriori algorithm) to generate association rules between items. This step is crucial for developing better marketing strategies and making informed decisions regarding the businesses.

b. What the input to the program will be?

The input to the program is the Grocery (GRC) dataset, which comprises customer transaction records, including items purchased, quantities, prices, and timestamps. Additionally, user-defined parameters may be considered for specific analysis tasks such as in clustering and association rule generation processes.

c. What the output from the program will be?

The program will produce a simplified analysis of the data set provided, including descriptive statistics, graphical visualizations, and potentially predictive models. These outputs will help stakeholders understand customer preferences, identify popular products, and anticipate future trends.

Role of Each Member:

Maryam Ibrahim (Team Leader): Performed the clustering part of the analysis, aiming to identify distinct customer segments based on their purchasing behavior, and contributed to report writing.

Mesk Khaled: Applied association rules to the data, identifying patterns and relationships among items.

Judy Ahmed and Asmaa Mahmoud: Conducted data visualization tasks, utilizing various techniques that best suit the visualization's purpose to insights into the dataset.

Basmala Moataz: Executed data cleaning procedures, ensuring the dataset's quality and consistency for analysis.

Dataset Description:

A Grocery dataset, often referred to as GRC (Grocery Retail Chain), is a collection of data that records various aspects of transactions and customer interactions within a grocery retail environment. This dataset typically contains information such as:

1. Transaction Details:

Records of individual transactions, including the items purchased, quantities, and prices.

2. Customer Information:

Data about customers, such as their demographic details (age, gender, ID, etc) and purchase history.

3. Store Information:

Information about the store itself, such as its location to determine the franchise of each purchase.

4. Payment Details:

Information about the payment methods used in transactions, such as cash, credit/debit cards, or digital payments.

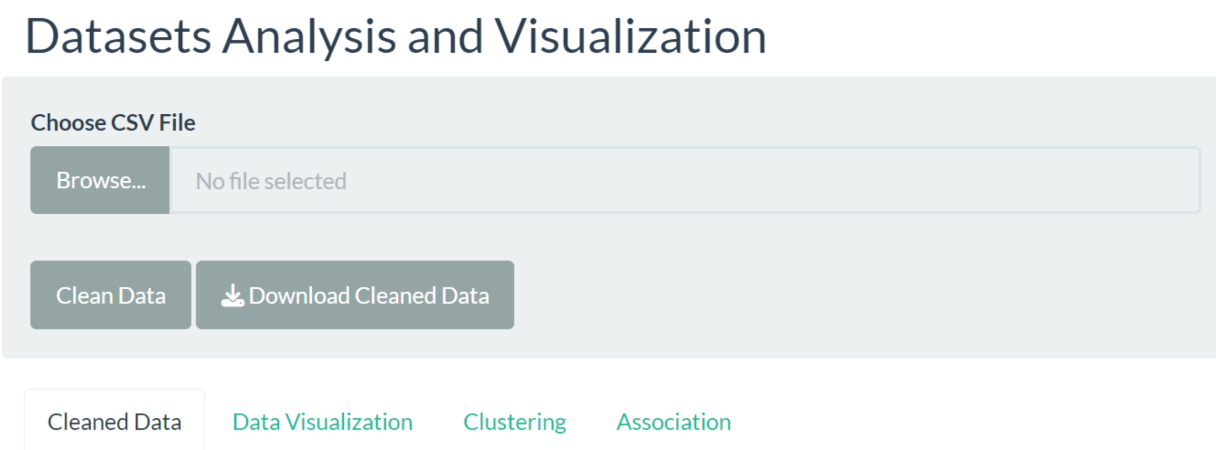
The Grocery dataset serves as a valuable resource for analyzing and understanding consumer behavior, preferences, and purchasing patterns within the grocery retail industry.

By analyzing this data, retailers can gain insights into trends, identify popular products, optimize inventory management, tailor marketing strategies, and enhance overall customer experience. Additionally, the dataset can be used for various analytical purposes, including market segmentation, trend forecasting, and recommendation systems.

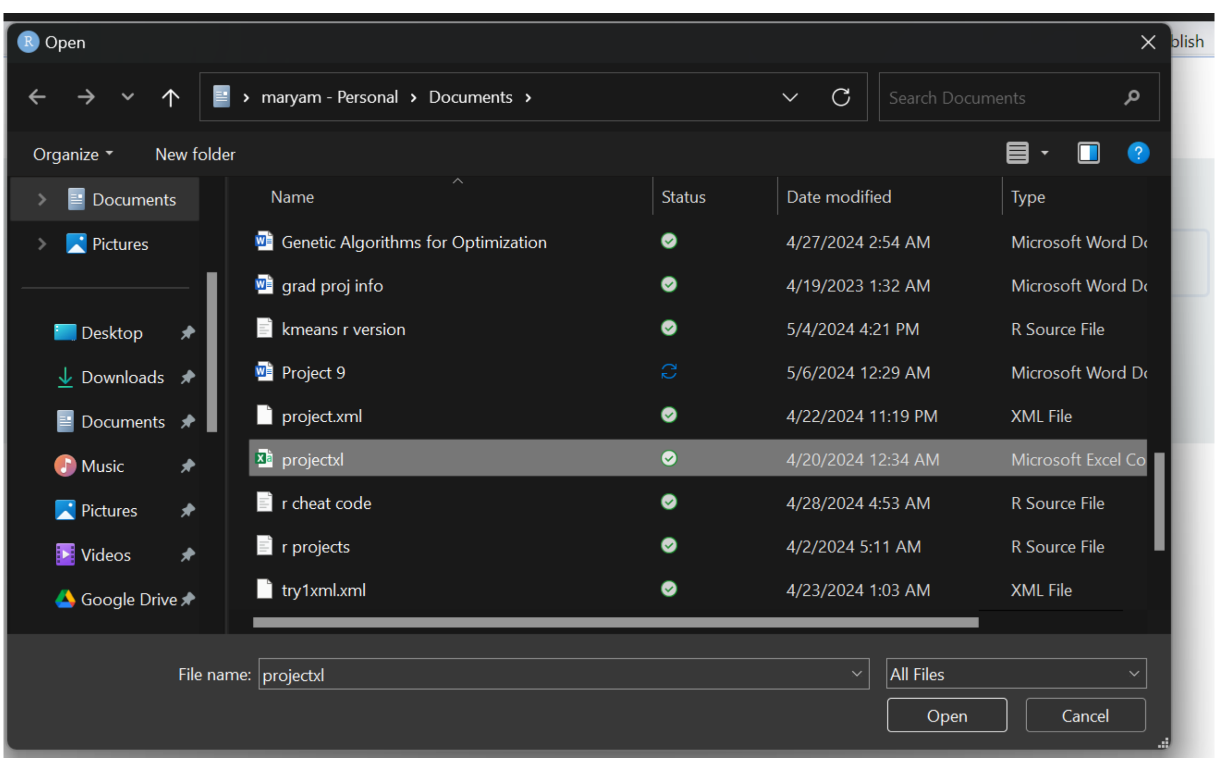
Project steps:

As illustrated before, this project consists of 4 main parts.

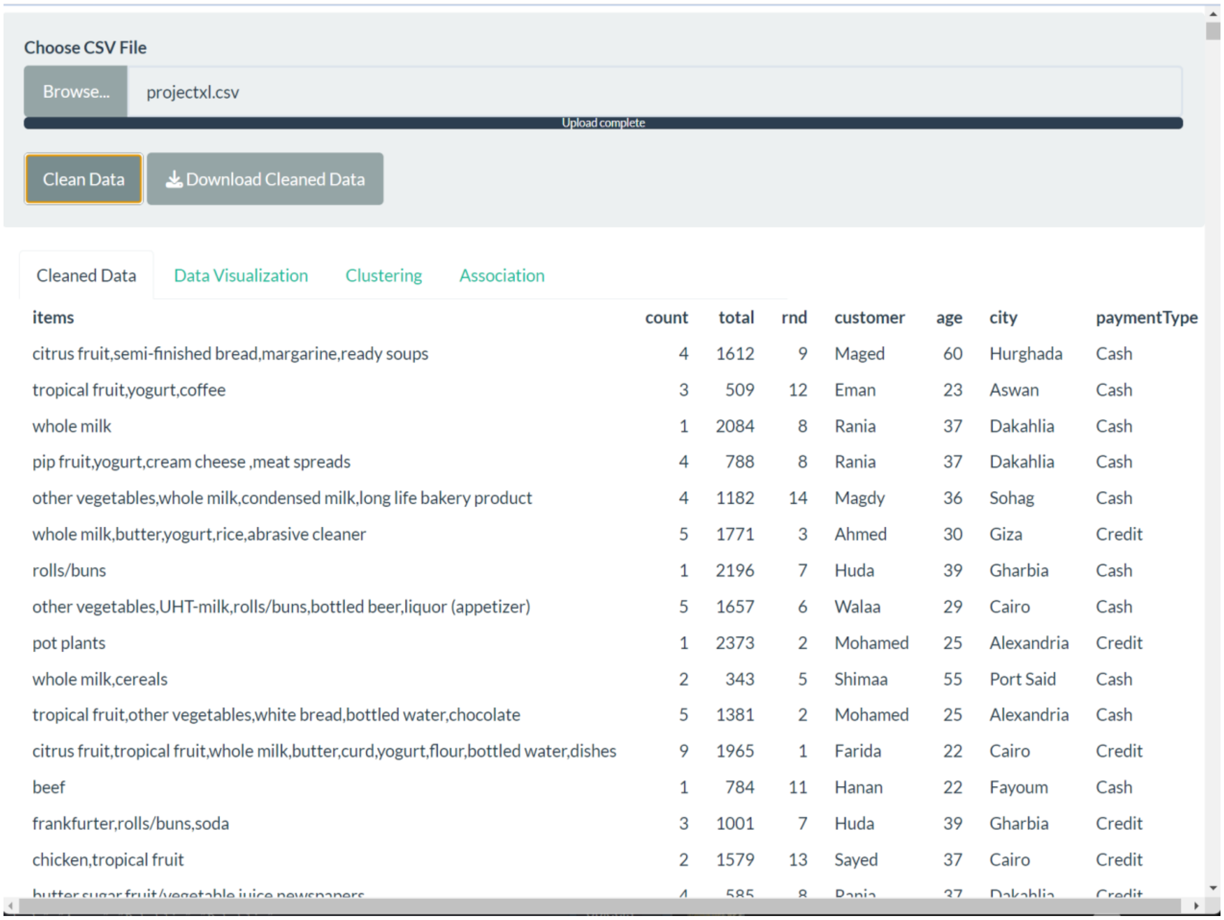
1. Data uploading and cleaning:
First of all, the user has to enter their preferred data that requires to be analyzed, this is done through the file input box at the top of the window as follows:



a) Click at the browse button, the following window will appear, select the required file:



b) After selecting the required file, click on the Clean button to start the process of data cleaning and viewing you final data in the cleaning tab panel (you can also download your cleaned data file by clicking on the download cleaned data button):



items	count	total	rnd	customer	age	city	paymentType
citrus fruit,semi-finished bread,margarine,ready soups	4	1612	9	Maged	60	Hurghada	Cash
tropical fruit,yogurt,coffee	3	509	12	Eman	23	Aswan	Cash
whole milk	1	2084	8	Rania	37	Dakahlia	Cash
pip fruit,yogurt,cream cheese ,meat spreads	4	788	8	Rania	37	Dakahlia	Cash
other vegetables,whole milk,condensed milk,long life bakery product	4	1182	14	Magdy	36	Sohag	Cash
whole milk,butter,yogurt,rice,abrasive cleaner	5	1771	3	Ahmed	30	Giza	Credit
rolls/buns	1	2196	7	Huda	39	Gharbia	Cash
other vegetables,UHT-milk,rolls/buns,bottled beer,liquor (appetizer)	5	1657	6	Walaa	29	Cairo	Cash
pot plants	1	2373	2	Mohamed	25	Alexandria	Credit
whole milk,cereals	2	343	5	Shimaa	55	Port Said	Cash
tropical fruit,other vegetables,white bread,bottled water,chocolate	5	1381	2	Mohamed	25	Alexandria	Cash
citrus fruit,tropical fruit,whole milk,butter,curd,yogurt,flour,bottled water,dishes	9	1965	1	Farida	22	Cairo	Credit
beef	1	784	11	Hanan	22	Fayoum	Cash
frankfurter,rolls/buns,soda	3	1001	7	Huda	39	Gharbia	Credit
chicken,tropical fruit	2	1579	13	Sayed	37	Cairo	Credit
butter,sugar,fruit/vegetable juice,newspapers	4	585	8	Rania	37	Dakahlia	Credit

This data will be the final data, which means that the rest of the steps results will be built upon this data version. It's important to make sure that the data cleaning process is applied correctly for the results of the rest of the project to be accurate.

2. Data Visualization according to specific requirements:

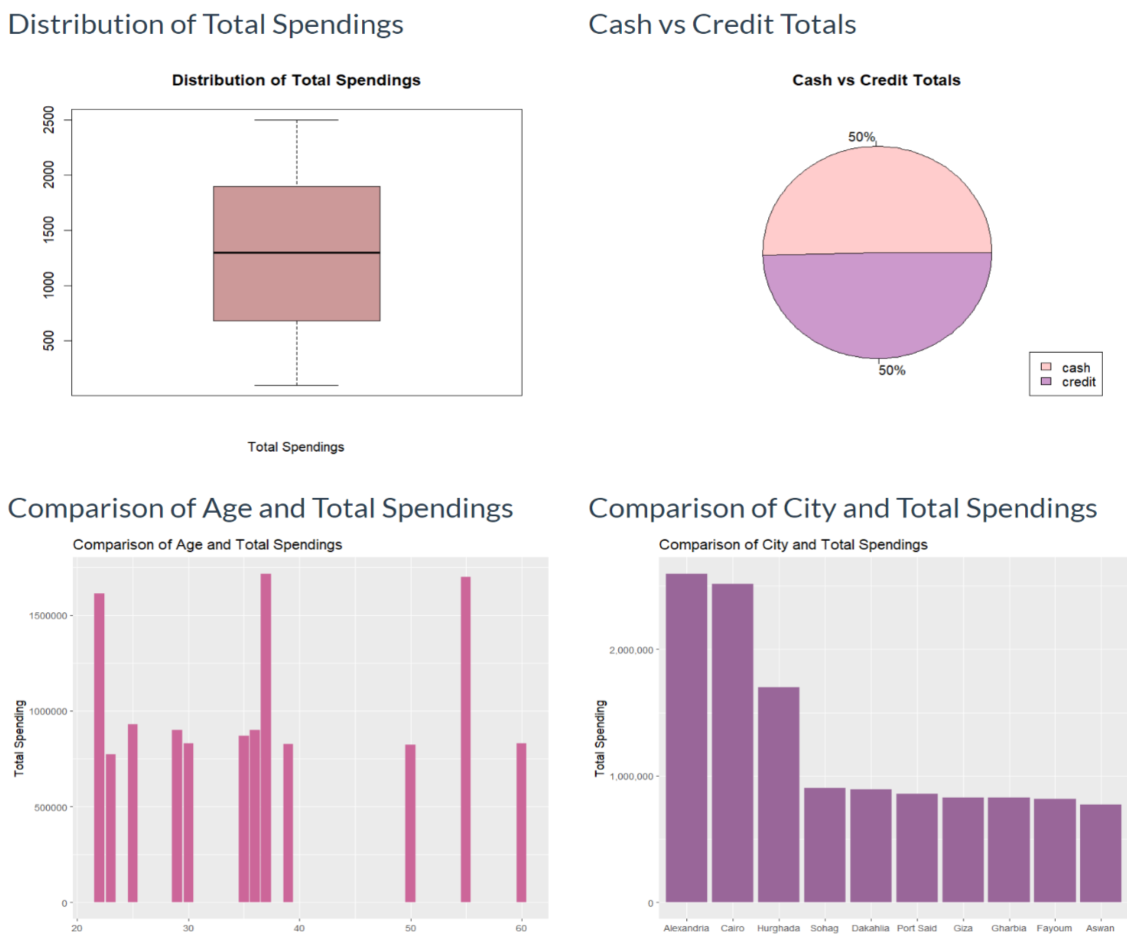
In that part, we perform 4 versions of data visualization with different parts of the data with different visualization techniques, depending on the type of data to be visualized, and then collect all the graphs in one place(dashboard).

These four versions are:

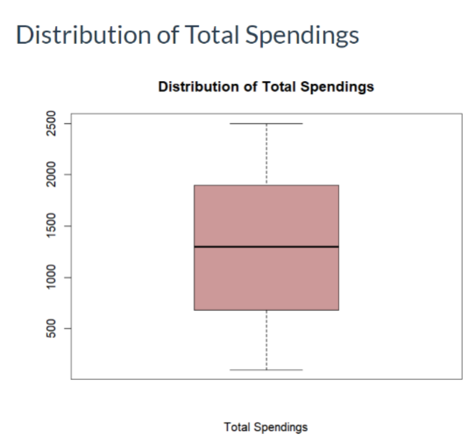
a) comparison between cash and credit total purchasing spending.
b) Comparing the total spending between each age range provided in the data.

c) Descending display of the total spending of each city.
d) Showing the distribution of the total spending.

The tab panel of this part will be displayed as the following:

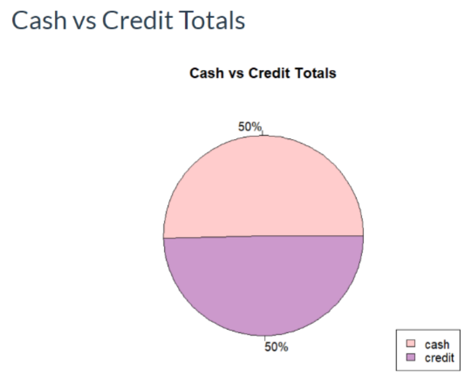


1. Distribution of total spending plot:

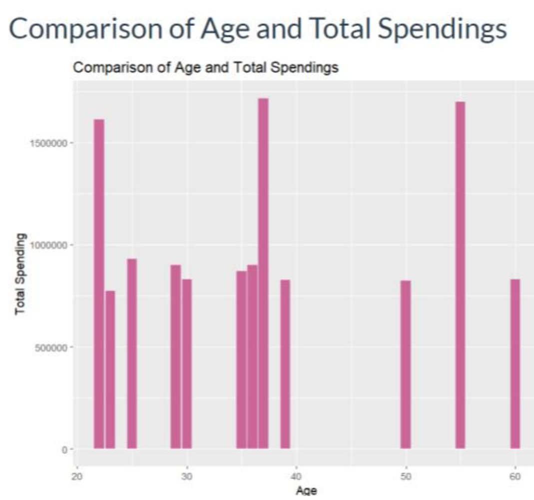


- This box plot illustrates the distribution of total spending among customers.
- The box plot displays key statistical measures such as the median, quartiles, and outliers, providing insights into the spread and central tendency of spending levels.
- By visualizing the distribution of total spending, the box plot helps identify any concentration or dispersion in spending behavior among customers.

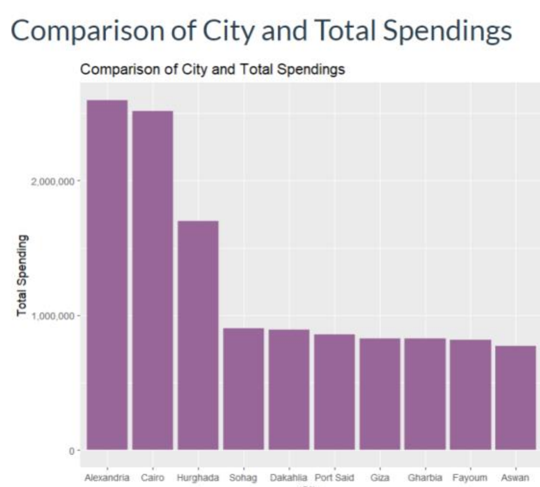
2. Cash VS credit total spending plot:
    - This pie chart illustrates the proportion of total spending attributed to transactions made using cash and credit.
    - Each slice of the pie represents a payment method (cash or credit), with the size of the slice indicating the percentage of total spending attributed to that method.
    - The pie chart provides a visual comparison of spending behavior between cash and credit transactions, allowing for easy identification of the dominant payment method.



3. Comparison of each age and total spending plot:
    - This bar plot compares the total spending across different age groups.
    - Each bar represents an age group, and the height of the bar corresponds to the total spending by customers in that age group.
    - The bar plot enables the comparison of spending patterns across different age demographics, highlighting any significant variations or trends.



4. Total spending of each city plot:
    - This bar plot displays the total spending for each city, arranged in descending order based on total spending.
    - Each bar represents a city, with the length of the bar indicating the total spending in that city.
    - The bar plot facilitates the comparison of spending levels across different cities, identifying which cities contribute the most to total sales.

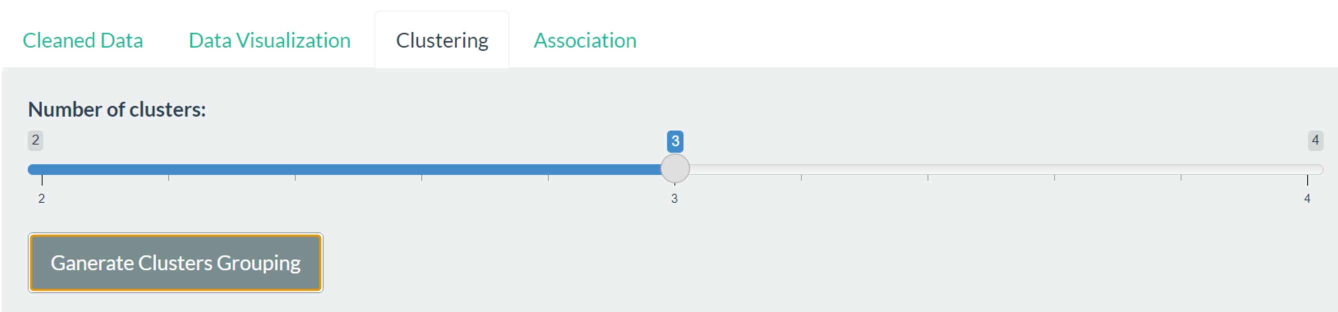


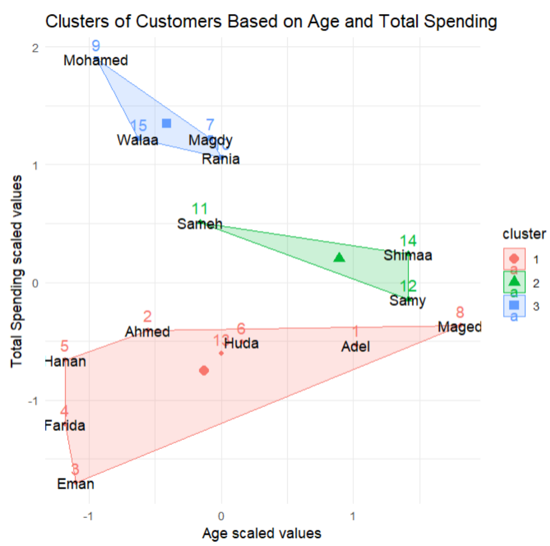
3. k-means clustering according to the age and total spending of each person:

In this part, we perform clustering (splitting the customers into groups) where each cluster contains customers with approximately equivalent ages and total spending. This is done by taking the cleaned data, extracting the needed info from it, and calculating the total spending of each customer. And since the k-means method only accepts numerical data, only the names of the customers won't be used during the calculations; instead, they will be placed as labels on each point on the graph to mark the cluster of each customer. Also, the final table will contain the customer's name, age, and total spending, in addition to the number of clusters it's assigned to.

To calculate the k-means clustering, you have to first determine the number of clusters through the slider input bar, and then click on the Generate clusters grouping button to display the table and the graph of distribution.

The following panel shows the result of the following steps:





customer	age	total_spending	cluster_number
Adel	50	824064	1
Ahmed	30	829587	1
Eman	23	772871	1
Farida	22	794570	1
Hanan	22	818543	1
Huda	39	825147	1
Magdy	36	901010	3
Maged	60	831272	1
Mohamed	25	930510	3
Rania	37	893789	3
Sameh	35	869668	2
Samy	55	841167	2
Sayed	37	820900	3
Shimaa	55	857901	2
Walaa	29	900797	3

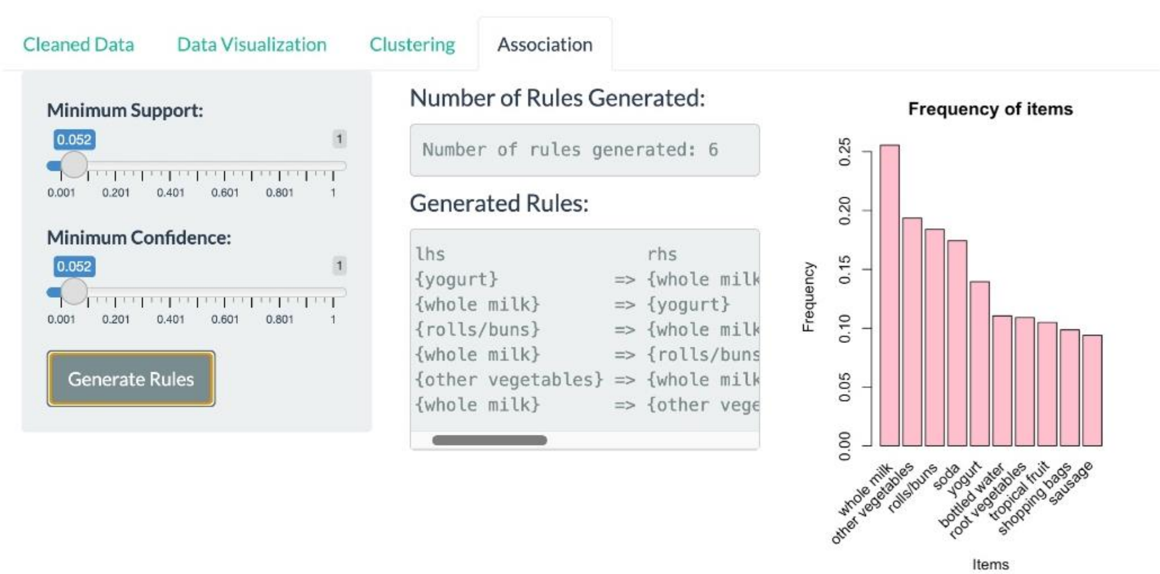
4. Association rule application:
   Association rule mining is a powerful technique used in data mining and market basket analysis to uncover relationships or patterns among items in large datasets. In the context of our project, association rules are applied to the grocery dataset to identify frequent item sets and generate meaningful insights into customer purchasing behavior.

   By analyzing transactional data from grocery purchases, association rule mining helps identify co-occurring items that are frequently purchased together. For example, if customers often buy bread and milk together, association rule mining would uncover this relationship and generate a rule reflecting this association.

   The generated association rules provide valuable insights for various business applications, including:

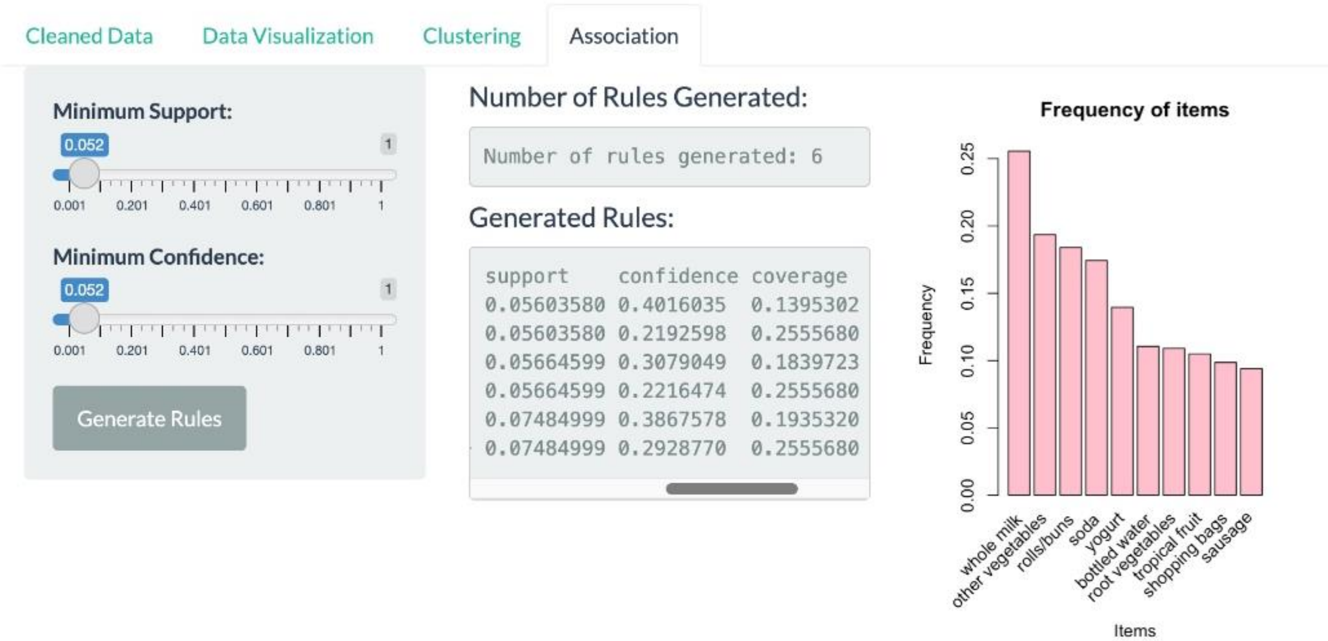
   Market basket analysis: Understanding which items are commonly purchased together, enables retailers to optimize product placement and promotions.
   Cross-selling and recommendation systems: Recommending additional products to customers based on their purchase history and preferences.
   Inventory management: Identifying related products to ensure appropriate stock levels and assortment planning.

   The following panel displays the calculation of the association rules techniques where you input the minimum support and confidence and then click the generate association rules button to display the number of rules generated, actual rules, and plot displaying the frequency of purchasing each item.



Note: In, the generated rules text box, you can move the slider to the right to view additional info about each rule such as support, confidence, etc.

This is shown in the following:



Program implementation:

In this part, we will explore the code behind our project and uncover the mechanisms driving our analysis.

Here, we provide a comprehensive explanation of the code snippets used in our analysis, the packages used, and the functionality of each line of code. By dissecting the code step by step, we aim to provide clarity to our methodology, allowing readers to understand the complexities of our data processing, analysis techniques, and visualization methods. Through this detailed examination of the code, readers can gain insights into our approach and replicate our methods in their analyses.

A. Installing and loading the necessary packages:

```r
# Install and load necessary packages

install.packages("readxl")
library(readxl)
install.packages("shiny")
library(shiny)
install.packages("dplyr")
library(dplyr)
install.packages("ggplot2")
library(ggplot2)
install.packages('factoextra')
library(factoextra)
install.packages("memoise")
library(memoise)
install.packages("arulesViz")
library(arulesViz)
install.packages("arules")
library(arules)
install.packages("shinythemes")
library(shinythemes)
install.packages("shinydashboard")
library(shinydashboard)
```

a) install.packages("readxl"): Installs the readxl package, which is used to read Excel files.
b) library(readxl): Loads the readxl package into the R environment.
c) install.packages("shiny"): Installs the shiny package, which is used to create interactive web applications in R.
d) library(shiny): Loads the shiny package into the R environment.
e) install.packages("dplyr"): Installs the dplyr package, which is used for data manipulation(in clustering).
f) library(dplyr): Loads the dplyr package into the R environment.
g) install.packages("ggplot2"): Installs the ggplot2 package, which is used for data visualization.
h) library(ggplot2): Loads the ggplot2 package into the R environment.
i) install.packages('factoextra'): Installs the factoextra package, which is used for data visualization(in clustering).
j) library(factoextra): Loads the factoextra package into the R environment.
k) install.packages("memoise"): Installs the memoise package, which is used to speed up functions by caching their results(in clustering).
l) library(memoise): Loads the memoise package into the R environment.
m) install.packages("arulesViz"): Installs the arulesViz package, which is used for visualizing association rules.
n) library(arulesViz): Loads the arulesViz package into the R environment.
o) install.packages("arules"): Installs the arules package, which is used for association rule mining.
p) library(arules): Loads the arules package into the R environment.
q) install.packages("shinythemes"): Installs the shinythemes package, which is used to customize the appearance of Shiny applications.
r) library(shinythemes): Loads the shinythemes package into the R environment.
s) install.packages("shinydashboard"): installs dashboard used in collecting the visualization graphs
t) library(shinydashboard): loads the shiny dashboard package into the R environment.

B. creating the UI fluid page (template where the processes occur) :

a) theme=shinythemes::shinytheme("flatly"): Sets the theme of the Shiny application to "flatly" using the shinythemes package.
b) titlePanel("Datasets Analysis and Visualization"): Creates a title panel with the title "Datasets Analysis and Visualization".
c) sidebarLayout: Creates a layout with a sidebar panel and a main panel.
d) sidebarPanel: Creates a sidebar panel with various input elements.

```
# Define UI (the appearance of the page)
ui <- fluidPage(
  theme=shinythemes::shinytheme("flatly"),
  titlePanel("Datasets Analysis and Visualization"),

  sidebarLayout(
    sidebarPanel(
      fileInput("file", "Choose CSV File"),
      actionButton("clean", "Clean Data"),
      downloadButton("download", "Download Cleaned Data"),
      width="auto"
    ),

    mainPanel(
      tabsetPanel(
        tabPanel("Cleaned Data", tableOutput("cleaned_table"),
                 ),

        tabPanel("Data Visualization",

                 fluidRow(
                   box(title = "Distribution of Total Spendings", plotOutput("boxplot")),
                   box(title = "Cash vs Credit Totals", plotOutput("piechart")),
                   box(title = "Comparison of Age and Total Spendings", plotOutput("agebarplot")),
                   box(title = "Comparison of City and Total Spendings", plotOutput("citybarplot"))
                 )
        ),
```

e) fileInput("file", "Choose CSV File"): Creates a file input element that allows the user to upload a CSV file.
f) actionButton("clean", "Clean Data"): Creates a button that triggers the cleaning of the data.
g) downloadButton("download", "Download Cleaned Data"): Creates a button that allows the user to download the cleaned data.
h) width="auto": Sets the width of the sidebar panel to "auto" to take the place across the whole window horizontally.
i) tabsetPanel: Creates a tabset panel with multiple tabs.
j) tabPanel("Cleaned Data", tableOutput("cleaned_table"),): Creates a tab with the title "Cleaned Data" and a table output with the ID "cleaned_table".
k) tabPanel("Data Visualization", fluidRow(...)): Creates a tab with the title "Data Visualization" and a fluid row with multiple boxes.
l) box: Creates a box with a title and a plot output.
m) plotOutput: Creates an output element for a plot.

n) tabPanel("Clustering", ...): Creates a tab with the title "Clustering" and various input and output elements.

o) sliderInput: Creates a slider input element with min. input of 2 and max. input of 4 (will be the number of clusters).

p) actionButton: Creates a button that triggers the calculation of clusters.

```
        tabPanel("Clustering",
                 sidebarPanel(
                   width = "auto",
                   sliderInput("clusters",
                             "Number of clusters:",
                             min = 2,
                             max = 4,
                             step = 1,
                             value = 2),
                   actionButton("clustersCalc", "Ganerate Clusters Grouping")),
                   mainPanel(
                     fluidRow(
                       column(width = 8, plotOutput("plot")),
                       column(width = 4, tableOutput("table"))
                     )
                 ),
             ),
```

q) mainPanel: Creates a main panel with a fluid row and two columns.

r) column: Creates a column with a plot output and a table output.

s) tabPanel("Association", ...): Creates a tab with the title "Association" and various input and output elements.

t) sliderInput: Creates a slider input element with min. and max. support and confidence values inputting.

u) actionButton: Creates a button that triggers the generation of association rules.

v) mainPanel: Creates a main panel with a fluid row and two columns.

w) column: Creates a column with various output elements.

x) verbatimTextOutput: Creates an output element for text.

```
        tabPanel("Association",
                 sidebarPanel(

                   sliderInput(
                     "minSupport",
                     "Minimum Support:",
                     min = 0.001,
                     max = 1,
                     value = 0.001,
                     step = 0.001
                   ),
                   sliderInput(
                     "minConfidence",
                     "Minimum Confidence:",
                     min = 0.001,
                     max = 1,
                     value = 0.001,
                     step = 0.001
                   ),
                   actionButton("generateButton", "Generate Rules")
                 ),
                 mainPanel(
                    fluidRow(
                  column(width = 6,  # Adjusted column width
                    h4("Number of Rules Generated:"),
                    verbatimTextOutput("numRules"),
                    h4("Generated Rules:"),
                    verbatimTextOutput("rules")
                  ),
                  column(width = 6,  # Adjusted column width
                    plotOutput("frequencyPlot")
                  )
                 )
               ),
      )
```

C. Creating the server logic that holds all the resultant actions of each component and output generating :

Tap panel1: Data cleaning

a) server <- function(input, output) {: This line defines a server function that takes two arguments: input and output. The input object contains data provided by the user through the UI, and the output object is used to send data to the UI.

b) cleaned_data <- eventReactive(input$clean, {...}): This creates a reactive expression that reads and cleans the data when the clean button is clicked in the UI. It reads a CSV file, removes duplicate rows and rows with NA values, and prints the structure of the data frame.

```r
# Define server logic (the resultant action of each component and output generating)
server <- function(input, output) {

  # Reactive function to read and clean the data
  cleaned_data <- eventReactive(input$clean, {
    req(input$file)
    data <- read.csv(input$file$datapath)
    # Remove duplicate rows
    data <- unique(data)
    # Remove rows with NA values
    data <- na.omit(data)
    # Print the structure of the data frame
    str(data)
    data
  })


  #tab panel 1:data cleaning
  # Output cleaned data as a table
  output$cleaned_table <- renderTable({
    req(is.data.frame(cleaned_data()))
    cleaned_data()
  })

  # Download handler for cleaned data CSV
  output$download <- downloadHandler(
    filename = function() {
      "cleaned_data.csv"
    },
    content = function(file) {
      write.csv(cleaned_data(), file, row.names = FALSE)
    }
  )

```

c) output$cleaned_table <- renderTable({...}): This renders a table in the UI that displays the cleaned data.

d) output$download <- downloadHandler({...}): This creates a download handler that allows the user to download the cleaned data as a CSV file.

Tap panel2: Data Visualization

```r
#tab panel 2: data visualizing
# 1)box plot
output$boxplot <- renderPlot({
  data <- cleaned_data()
  boxplot(data$total,
          main = "Distribution of Total Spendings",
          xlab = "Total Spendings",
          col = c("#CC9999"))
})

# 2)pie chart
output$piechart <- renderPlot({
  data <- cleaned_data()
  x <- table(data$paymentType)
  percentage <- paste0(round(100 * x / sum(x)), "%")
  pie(
    x ,
    labels = percentage,
    main = "Cash vs Credit Totals",
    col = c("#FFCCCC", "#CC99CC")
  )
  legend(
    "bottomright",
    legend = c("cash", "credit"),
    fill = c("#FFCCCC", "#CC99CC")
  )
})

# 3)bar plot of the total spendig of each age
output$agebarplot <- renderPlot({
  data <- cleaned_data()
  total_spending_age <- aggregate(total ~ age, data = data, FUN = sum)
  ggplot(total_spending_age, aes(x = age, y = total)) +
    geom_bar(stat = "identity", fill = "#CC6699") +
    labs(x = "Age", y = "Total Spending", title = "Comparison of Age and Total Spendings")
})

# 4)bar plot of the total spending of people from each city
output$citybarplot <- renderPlot({
  data <- cleaned_data()
  total_spending_city <- aggregate(total ~ city, data = data, FUN = sum)
  ggplot(total_spending_city, aes(x = reorder(city, -total), y = total)) +
    geom_bar(stat = "identity", fill = "#996699") +
    labs(x = "City", y = "Total Spending", title = "Comparison of City and Total Spendings") +
    scale_y_continuous(labels = scales::comma)
})
```

a) Box plot:

This code creates a box plot of the total column in the cleaned_data() . The plot displays the distribution of total spending, with the x-axis labeled as "Total Spendings" and the box plot colored in #CC9999.


```r
#tab panel 2: data visualizing
# 1)box plot
output$boxplot <- renderPlot({
  data <- cleaned_data()
  boxplot(data$total,
          main = "Distribution of Total Spendings",
          xlab = "Total Spendings",
          col = c("#CC9999"))
})
```


b) Pie chart:

This code creates a pie chart of the paymentType column in the cleaned_data() data frame. The plot displays the percentage of total spendings for cash and credit payments and creates a refernce box for contents(legend box), with the pie chart colored in #FFCCCC and #CC99CC.


```r
# 2)pie chart
output$piechart <- renderPlot({
  data <- cleaned_data()
  x <- table(data$paymentType)
  percentage <- paste0(round(100 * x / sum(x)), "%")
  pie(
    x ,
    labels = percentage,
    main = "Cash vs Credit Totals",
    col = c("#FFCCCC", "#CC99CC")
  )
  legend(
    "bottomright",
    legend = c("cash", "credit"),
    fill = c("#FFCCCC", "#CC99CC")
  )
})
```

c) Bar plot of total spending by age:

This code creates a bar plot of the total spendings by age using the ggplot2 package. The plot aggregates the total column by the age column in the cleaned_data() data frame, and displays the results as a bar plot with the x-axis labeled as "Age", the y-axis labeled as "Total Spending", and the bars colored in #CC6699.

d) Bar plot of total spending by city:

```
# 4)bar plot of the total spending of people from each city
output$citybarplot <- renderPlot({
  data <- cleaned_data()
  total_spending_city <- aggregate(total ~ city, data = data, FUN = sum)
  ggplot(total_spending_city, aes(x = reorder(city, -total), y = total)) +
    geom_bar(stat = "identity", fill = "#996699") +
    labs(x = "City", y = "Total Spending", title = "Comparison of City and Total Spendings") +
    scale_y_continuous(labels = scales::comma)
})

```

This code creates a bar plot of the total spendings by city using the ggplot2 package. The plot aggregates the total column by the city column in the cleaned_data() data frame, and displays the results as a bar plot with the x-axis labeled as "City", the y-axis labeled as "Total Spending", and the bars colored in #996699. The reorder() function is used to order the cities by their total spendings in descending order. The scale_y_continuous() function is used to format the y-axis labels with commas as thousand separators.

Note: all the colors used in this project are used in their hexadecimal values, this enabeled us to have more varity of colors choices.

3. Tab panel3: Clustering

```r
#tab panel 3: k means clustering
#1)getting the specific columns of data needed for the process
data.clust <- eventReactive(input$clustersCalc,{
  data.f<-cleaned_data()[, c("customer", "age", "total")] %>%
        group_by(customer, age) %>% summarise(total_spending = sum(total))
})

#2)getting the number of clusters according to the input of the slider input
cluster <- eventReactive(input$clustersCalc,input$clusters)

#3)calculate k-means clustering
k_means <- reactive({
  kmeans(data.clust()[, c("age", "total_spending")], centers = as.integer(cluster()))
})

#4)create the table of each customer info and cluster number
output$table <- renderTable({
  if (is.null(k_means())) {
    return("K-means clustering has not been calculated yet.")
  } else {
    final_data <- data.clust() %>% mutate(cluster_number = k_means()$cluster[match(age, data.clust()$age)])
  }
})

#5)visualizing the clustering result
output$plot <- renderPlot({
  if (is.null(k_means())) {
    return("K-means clustering has not been calculated yet.")
  } else {
    fviz_cluster(k_means(), data = data.clust()[, c("age", "total_spending")], add.cluster = TRUE) +
                  labs(title = 'Clusters of Customers Based on Age and Total Spending',
                       x = 'Age scaled values', y = 'Total Spending scaled values') +
                  theme_minimal() +
                  geom_text(label = data.clust()$customer, check_overlap = TRUE)
  }
})
```

a) data.clust <- eventReactive(input$clustersCalc, { ... }) :

This line creates a reactive expression that subsets the cleaned_data() data frame to include only the customer, age, and total columns. It then groups the data by customer and age, and calculates the total spending for each group. The resulting data frame is returned as data.clust().


```r
#tab panel 3: k means clustering
#1)getting the specific columns of data needed for the process
data.clust <- eventReactive(input$clustersCalc,{
  data.f<-cleaned_data()[, c("customer", "age", "total")] %>%
        group_by(customer, age) %>% summarise(total_spending = sum(total))
})
```

b) cluster <- eventReactive(input$clustersCalc, input$clusters): This line creates a reactive expression that returns the number of clusters specified by the user in the Shiny app.

```
#2)getting the number of clusters according to the input of the slider input
cluster <- eventReactive(input$clustersCalc,input$clusters)
```

3.k_means <- reactive({ ... }): This line creates a reactive expression that calculates the k-means clustering using the data.clust() data frame and the number of clusters specified by the user. The resulting k-means object is returned as k_means().

```
#3)calculate k-means clustering
k_means <- reactive({
  kmeans(data.clust()[, c("age", "total_spending")], centers = as.integer(cluster()))
})
```

c) output$table <- renderTable({ ... }: This line creates a table that displays the customer information and the cluster number assigned to each customer. The table is only displayed if the k-means clustering has been calculated.

```
#4)create the table of each customer info and cluster number
output$table <- renderTable({
  if (is.null(k_means())) {
    return("K-means clustering has not been calculated yet.")
  } else {
    final_data <- data.clust() %>% mutate(cluster_number = k_means()$cluster[match(age, data.clust()$age)])
  }
})
```

d) output$plot <- renderPlot({ ... }) : This line creates a scatter plot that displays the age and total spending of each customer, with different colors representing different clusters. The plot is only shown if the k-means clustering has been calculated.

The fviz_cluster() function from the factoextra package creates the plot. The add.cluster argument is set to TRUE to add the cluster centers to the plot. The labs() function is used to add a title and labels to the plot. The theme_minimal() function is used to simplify the plot theme (mainly used to set the background color to white). The geom_text() function is used to add the customer IDs to the plot. The check_overlap argument is set to TRUE to prevent overlapping text labels.

```
#5)visualizing the clustering result
output$plot <- renderPlot({
  if (is.null(k_means())) {
    return("K-means clustering has not been calculated yet.")
  } else {
    fviz_cluster(k_means(), data = data.clust()[, c("age", "total_spending")], add.cluster = TRUE) +
                  labs(title = 'Clusters of Customers Based on Age and Total Spending',
                       x = 'Age scaled values', y = 'Total Spending scaled values') +
                  theme_minimal() +
                  geom_text(label = data.clust()$customer, check_overlap = TRUE)
  }
})

```

4.Tab panel4: Associationn Rule

```r
#tab panel 4: association rules
#1) Reactive function to generate association rules based on user input
rules <- eventReactive(input$generateButton, {
  req(cleaned_data(), input$minSupport, input$minConfidence)
  data <- cleaned_data()
  minSupport <- input$minSupport
  minConfidence <- input$minConfidence
  transactions_list <- lapply(data$items, function(x)unlist(strsplit(x, ",")))
  options("max.print" = 8000000)
  items_data <- as(transactions_list, "transactions")
  rules <- apriori(
    items_data,
    parameter = list(
      support = minSupport,
      confidence = minConfidence,
      minlen = 2
    )
  )
  return(rules)
})

# 2)text output of the number of rules generated
output$numRules = renderText({
  req(rules())
  paste("Number of rules generated:", length(rules()))
})

# 3)output of the generated rules details
output$rules = renderPrint({
  req(rules())
  if (!is.null(rules())) {
    inspect(rules())
  } else {
    print("No rules generated yet")
  }
})
```

a) rules <- eventReactive(input$generateButton, { ... }): This line creates a reactive function that generates association rules based on user input. It takes the cleaned_data() data frame and the values of input$minSupport and input$minConfidence as inputs. It then creates a list of transactions from the items column of the cleaned_data() data frame, and generates association rules using the apriori() function from the arules package. The resulting rules are returned as rules().


```r
#tab panel 4: association rules
#1) Reactive function to generate association rules based on user input
rules <- eventReactive(input$generateButton, {
  req(cleaned_data(), input$minSupport, input$minConfidence)
  data <- cleaned_data()
  minSupport <- input$minSupport
  minConfidence <- input$minConfidence
  transactions_list <- lapply(data$items, function(x)unlist(strsplit(x, ",")))
  options("max.print" = 8000000)
  items_data <- as(transactions_list, "transactions")
  rules <- apriori(
    items_data,
    parameter = list(
      support = minSupport,
      confidence = minConfidence,
      minlen = 2
    )
  )
  return(rules)
})
```

b) output$numRules = renderText({ ... }): This line creates a text output that displays the number of rules generated. It takes the length of rules() as input and returns a text string with the number of rules.

```r
# 2)text output of the number of rules generated
output$numRules = renderText({
  req(rules())
  paste("Number of rules generated:", length(rules()))
})
```

c) output$rules = renderPrint({ ... }): This line creates a print output that displays the details of the generated rules. It takes rules() as input and returns the output of the inspect() function, which displays the details of the rules.

```r
# 3)output of the generated rules details
output$rules = renderPrint({
  req(rules())
  if (!is.null(rules())) {
    inspect(rules())
  } else {
    print("No rules generated yet")
  }
  width="auto"
})
```

d) output$frequencyPlot = renderPlot({ ... }): This line creates a frequency plot that displays the frequency of items in the transactions. It takes the cleaned_data() data frame and the value of input$generateButton as inputs. It then creates a list of transactions from the items column of the cleaned_data() data frame, and generates a frequency plot using the itemFrequencyPlot() function from the arules package. The resulting plot is returned as output$frequencyPlot.

```r
#  4) visualizing the output in a frequency plot
output$frequencyPlot = renderPlot({
  req(cleaned_data(), input$generateButton)
  data <- cleaned_data()
  if (!is.null(rules())) {
    transactions_list <- lapply(data$items, function(x) unlist(strsplit(x, ",")))
    items_data <- as(transactions_list, "transactions")
    itemFrequencyPlot(
      items_data,
      main = "Frequency of items",
      xlab = "Items",
      ylab = "Frequency",
      col = "#CC9999",
      topN = 10,
      type= 'absolute'
    )
  }
})
```

C. Running the application:

```
# Running the constructed application
shinyApp(ui = ui, server = server)
```

This line launches the Shiny application with the user interface (ui) and server logic (server) defined earlier in the code. The ui argument specifies the user interface components, such as input controls and output displays, that are displayed to the user. The server argument specifies the server logic that processes user input and generates output. By calling shinyApp() with these arguments, the Shiny application is initialized and executed, allowing the user to interact with the application and view the output.

Conclusion:

In conclusion, we have developed a Shiny application that provides an interactive interface for exploring and analyzing the dataset. The application includes various input controls and output displays, allowing users to easily manipulate and visualize the data. The server logic is implemented using R code, which processes user input and generates output using various R packages and functions.

The Shiny application offers several advantages over traditional static reports, including improved usability, flexibility, and interactivity. By allowing users to interact with the data and customize the analysis to their needs, the application can provide more personalized and relevant insights. Furthermore, the application can be easily updated and modified as new data becomes available, making it a valuable tool for ongoing analysis and monitoring.

Overall, the Shiny application is a powerful tool for data analysis and visualization, and it has the potential to enhance data-driven decision-making. We believe that this application can be a valuable resource for stakeholders and decision-makers, and we look forward to continuing to refine and improve it in the future.