Alexandria University
Faculty of Computer and Data Science
Department : Data Science
Course Title: Data Science 2023-2024

Name	ID	Role
Leader: Abdelrahman Khaled Mohamed Mohamed	23011315	Association
Yassin Mohamed Hassan Elsayed	23011622	Clustering and report
Selvia Nasser Bekhit	23011293	Graphic user interface
Shahd Hesham Ahmed Abdelkader	23010156	Scanning and cleaning data
Habiba Hamdy Ezzat Abdelgwad	23011250	Visualization

Introduction

This section will inform many details about the program. First of all, the program made to clean the entered dataset of customers including different details about the customers as

- Names
- City
- Payment method
- Age
- Total customer spending

- RND
- Items

Unused data will be cleared as (NA / Duplicated) values Then it will calculate, compare and group data in graphical analytics like:

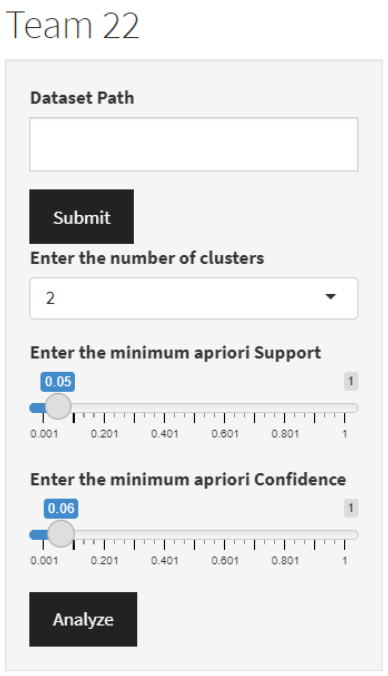
- Comparing between payment methods used
- Sum of total spending according to age
- Total spending for each city
- Distribution of total spending

(Then it will group all the calculated graphs in dashboard.)

- **The program also will cluster the data and creates a cluster table.**

- Customer total spend clusters
- Sum of total spending according to age and clusters

The only data user needs to provide is:


Team 22

Dataset Path

Submit

Enter the number of clusters

2

Enter the minimum apriori Support

0.05

Enter the minimum apriori Confidence

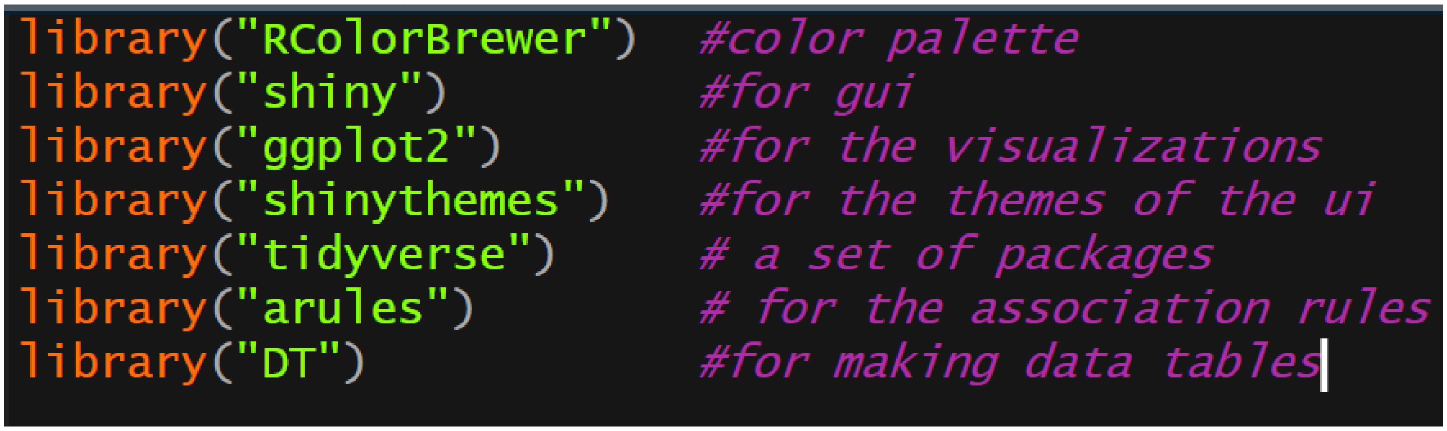
0.06

Analyze

- Data Set in CSV file.
- Cluster numbers.
- Minimum support and confident values.

Next section will display program GUI and code with detailed explanation.

Libraries



```
library("RColorBrewer")   #color palette
library("shiny")          #for gui
library("ggplot2")        #for the visualizations
library("shinythemes")    #for the themes of the ui
library("tidyverse")      # a set of packages
library("arules")         # for the association rules
library("DT")             #for making data tables
```

Program includes many libraries as (RColorBrewer - shiny - ggplot2 – shinythemes – tidyverse – arules - DT)

Each library has its own function.

RColorBrewer: is used to add gradient colors to plots

Shiny: For making Graphic User Interface (GUI)

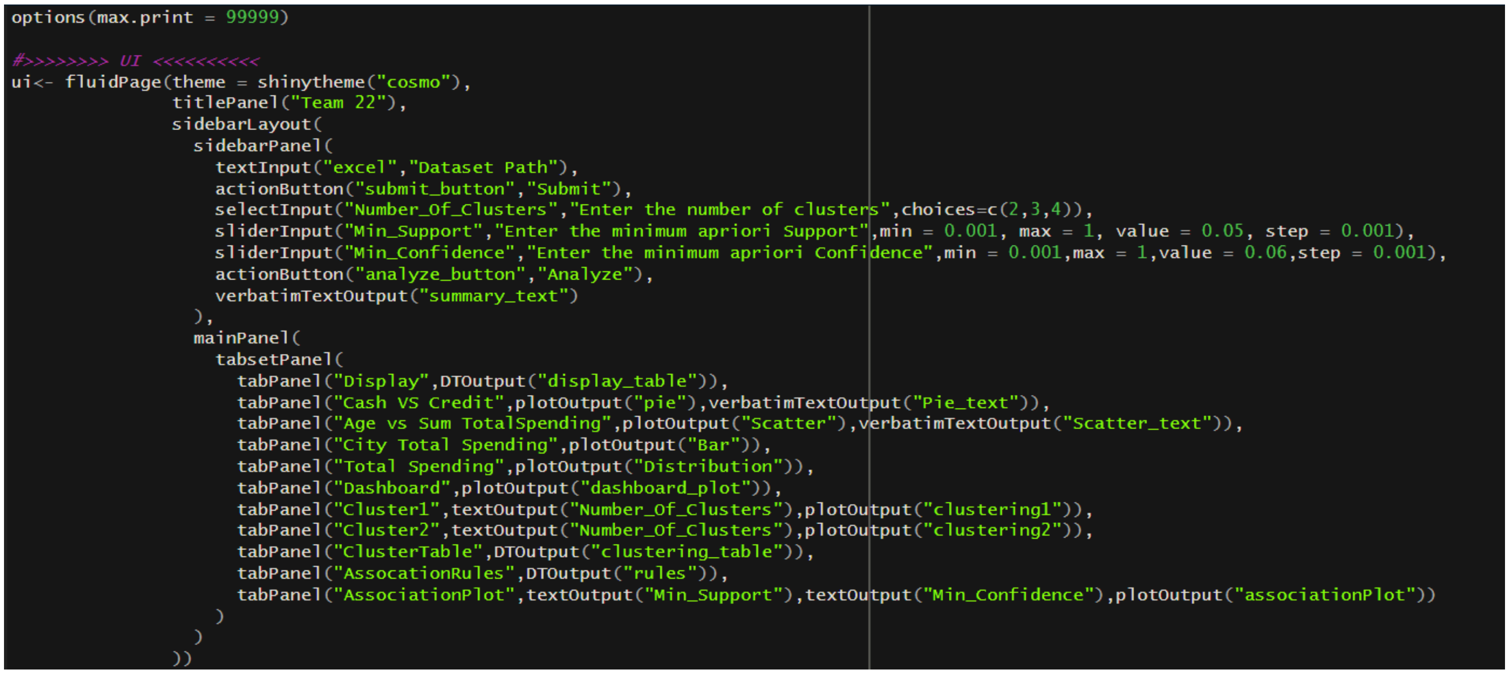
ggplot2: For visualizing the data in form of graphs and plots

tidyverse: it is a library with many packages like “dplyr” package which used in visualizing data

arules: is used for adding association rules

DT: for making data tables

Graphical user interface



```
options(max.print = 99999)

#>>>>>>>>> UI <<<<<<<<<<<
ui<- fluidPage(theme = shinytheme("cosmo"),
               titlePanel("Team 22"),
               sidebarLayout(
                 sidebarPanel(
                   textInput("excel","Dataset Path"),
                   actionButton("submit_button","Submit"),
                   selectInput("Number_Of_Clusters","Enter the number of clusters",choices=c(2,3,4)),
                   sliderInput("Min_Support","Enter the minimum apriori Support",min = 0.001, max = 1, value = 0.05, step = 0.001),
                   sliderInput("Min_Confidence","Enter the minimum apriori Confidence",min = 0.001,max = 1,value = 0.06,step = 0.001),
                   actionButton("analyze_button","Analyze"),
                   verbatimTextOutput("summary_text")
                 ),
                 mainPanel(
                   tabsetPanel(
                     tabPanel("Display",DTOutput("display_table")),
                     tabPanel("Cash VS Credit",plotOutput("pie"),verbatimTextOutput("Pie_text")),
                     tabPanel("Age vs Sum TotalSpending",plotOutput("Scatter"),verbatimTextOutput("Scatter_text")),
                     tabPanel("City Total Spending",plotOutput("Bar")),
                     tabPanel("Total Spending",plotOutput("Distribution")),
                     tabPanel("Dashboard",plotOutput("dashboard_plot")),
                     tabPanel("Cluster1",textOutput("Number_Of_Clusters"),plotOutput("clustering1")),
                     tabPanel("Cluster2",textOutput("Number_Of_Clusters"),plotOutput("clustering2")),
                     tabPanel("ClusterTable",DTOutput("clustering_table")),
                     tabPanel("AssocationRules",DTOutput("rules")),
                     tabPanel("AssociationPlot",textOutput("Min_Support"),textOutput("Min_Confidence"),plotOutput("associationPlot"))
                   )
                 )
               ))
```

Title panel is where the title displayed as it will be showed later in the graphical part of the report.

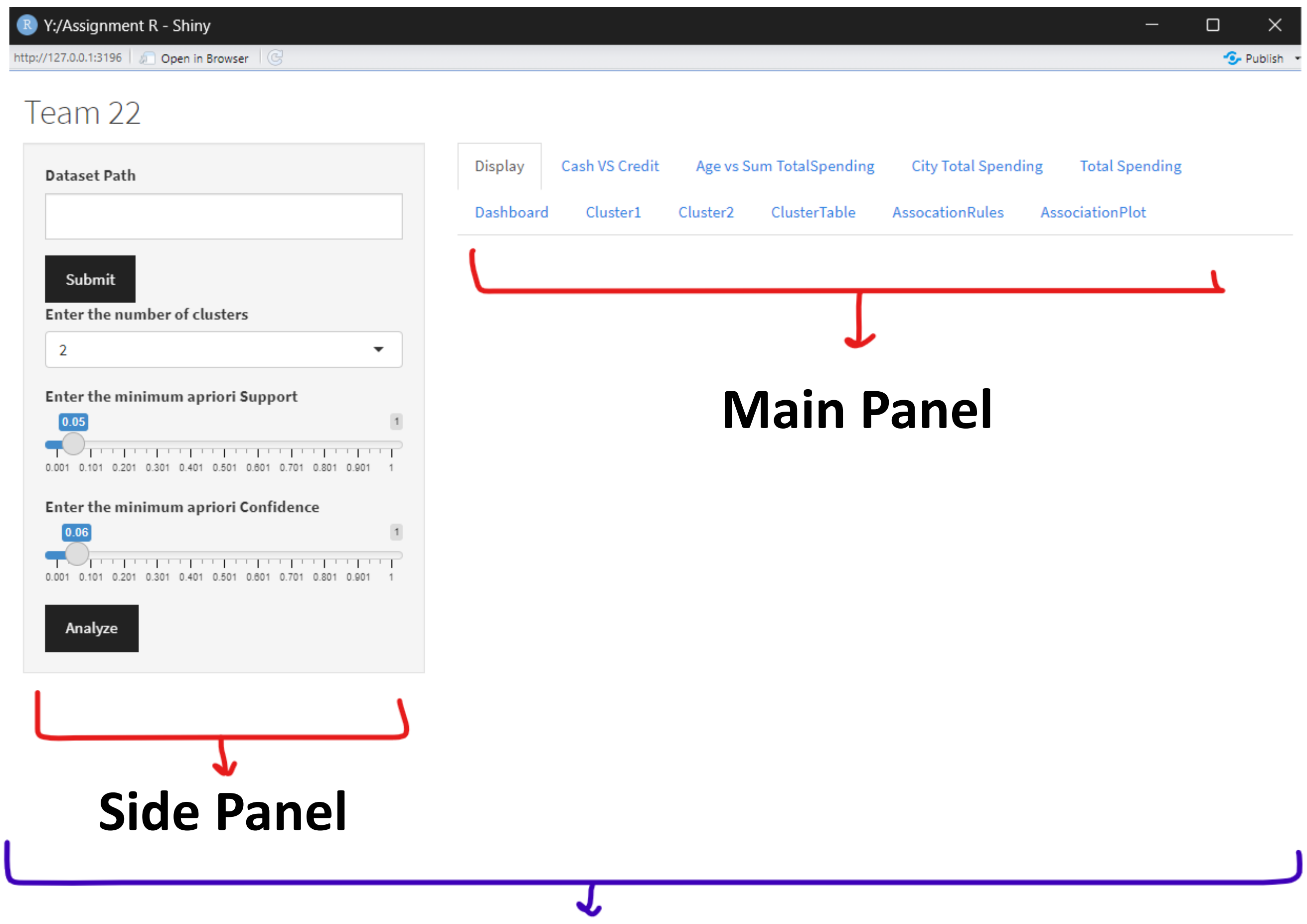
Side Bar

- SidebarLayout is where all the tabs will be displayed.
- SidebarPanel is where the data will be entered, the confident and support values set and number of clusters chose. This tabs created by internal functions in "sidebarpanel()" like:

- **textInput**: where the ID "excel" added to it and the title of the input set "Data Path".
- **actionButton**: in this program this button added to submit data entering to start calculating needed data
- **selectInput**: is an option added to allow the user to choose within many values.
- **sliderInput**: it is another way to allow the user to choose between interval of choices in form of slider.

Main Panel

Main panel is where the data calculated, visualized and entered displayed in many tabs added using internal functions inside "mainpanel()" as all tabs are added using "tabsetPanel()" function and each tab display specific data, graph and calculations by (DToutput() – plotOutput() – textOutput()) arguments as they call data from the visualization section as each plot has its own ID and the first argument set in the "tabPanel" is the ID of the panel that shows what each panel will display.



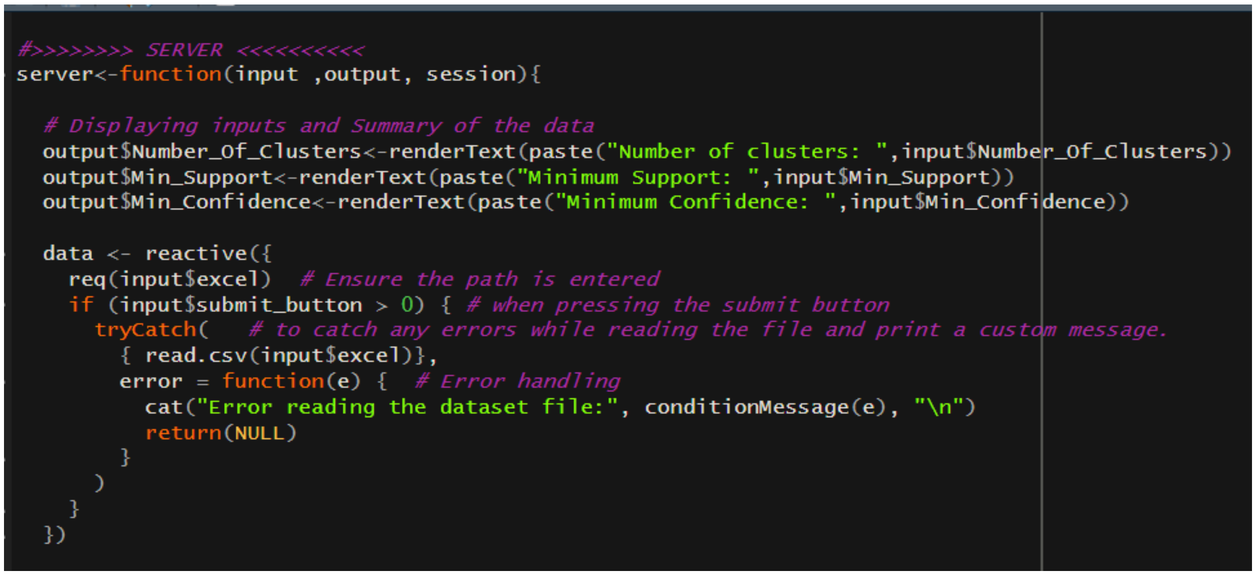
Enter the number of clusters: 2

Enter the minimum apriori Support: 0.05

Enter the minimum apriori Confidence: 0.06

Side Bar Layout (The whole layout)

Servers



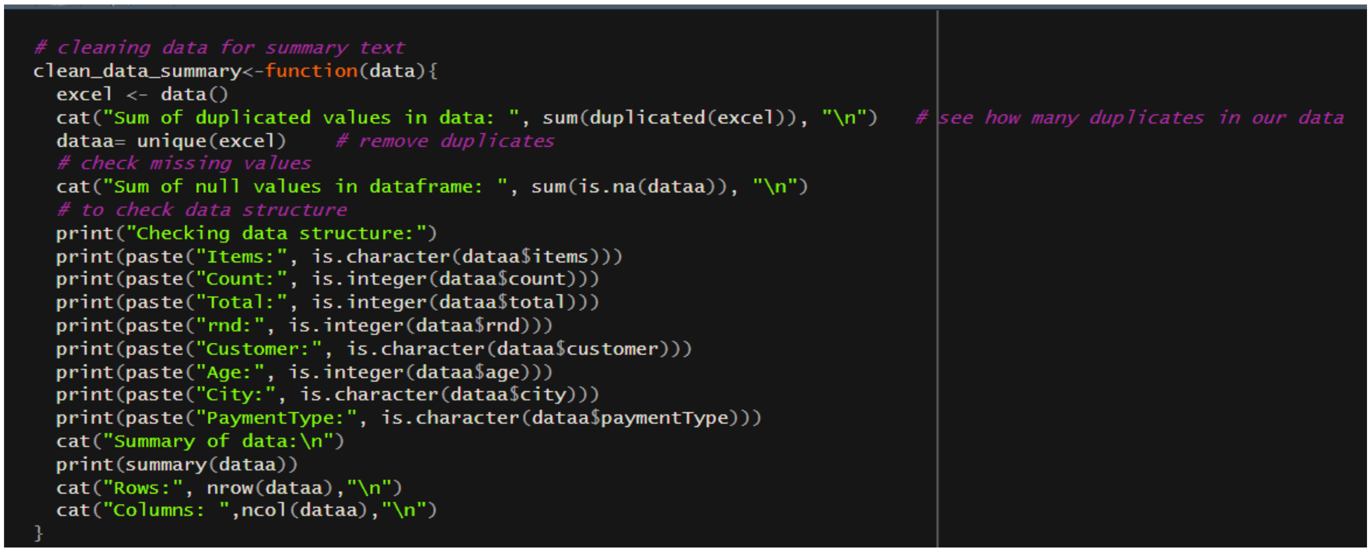
```
#>>>>>>>> SERVER <<<<<<<<<<
server<-function(input ,output, session){

  # Displaying inputs and Summary of the data
  output$Number_Of_Clusters<-renderText(paste("Number of clusters: ",input$Number_Of_Clusters))
  output$Min_Support<-renderText(paste("Minimum Support: ",input$Min_Support))
  output$Min_Confidence<-renderText(paste("Minimum Confidence: ",input$Min_Confidence))

  data <- reactive({
    req(input$excel)  # Ensure the path is entered
    if (input$submit_button > 0) { # when pressing the submit button
      tryCatch(   # to catch any errors while reading the file and print a custom message.
        { read.csv(input$excel)},
        error = function(e) {  # Error handling
          cat("Error reading the dataset file:", conditionMessage(e), "\n")
          return(NULL)
        }
      )
    }
  })
```

Server is used to display the inputs and summary text in the main panel by assigning each variable to the arguments in the function (input , output , sessions) to reuse the input data and display it in the output.

“req()” Is used to request the path added to the input to ensure that the path is entered correctly.

If condition added to handle errors if it occurred as soon as the submit button clicked at least once the “tryCatch()” function will play its role in catching errors while reading the file and print custom message using “cat()” function.

Data Cleaning



```
# cleaning data for summary text
clean_data_summary<-function(data){
  excel <- data()
  cat("Sum of duplicated values in data: ", sum(duplicated(excel)), "\n")   # see how many duplicates in our data
  dataa= unique(excel)    # remove duplicates
  # check missing values
  cat("Sum of null values in dataframe: ", sum(is.na(dataa)), "\n")
  # to check data structure
  print("Checking data structure:")
  print(paste("Items:", is.character(dataa$items)))
  print(paste("Count:", is.integer(dataa$count)))
  print(paste("Total:", is.integer(dataa$total)))
  print(paste("rnd:", is.integer(dataa$rnd)))
  print(paste("Customer:", is.character(dataa$customer)))
  print(paste("Age:", is.integer(dataa$age)))
  print(paste("City:", is.character(dataa$city)))
  print(paste("PaymentType:", is.character(dataa$paymentType)))
  cat("Summary of data:\n")
  print(summary(dataa))
  cat("Rows:", nrow(dataa),"\n")
  cat("Columns: ",ncol(dataa),"\n")
}
```

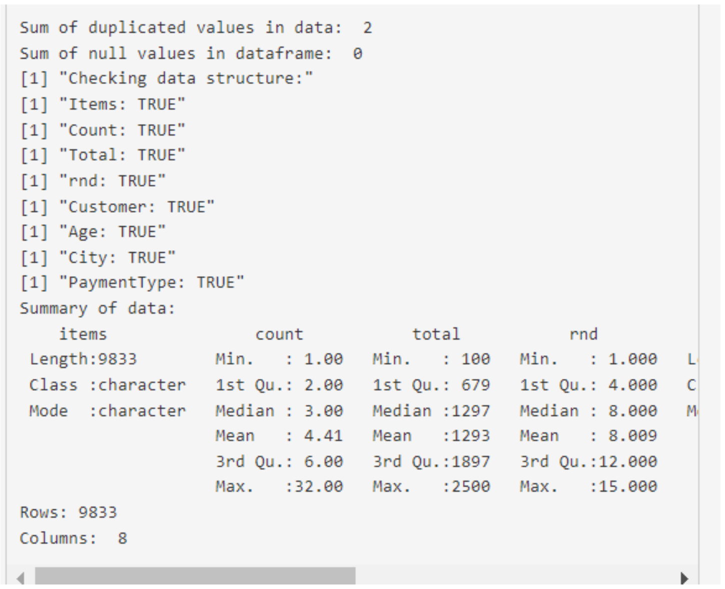
Data cleaning is one of the main features of the program as it is the process where the dataset will be read and the program will clean the unused data like duplicated data or NA values to prepare data to be displayed in the summary text.

- First data will be stored in a variable named "excel" then the excel file entered will be scanned and in the summary text the sum of the duplicated data will be displayed in the summary text using "cat()" function and "sum(duplicated(dataEntred))" function and arguments then the same process will be repeated for the missing values "NA" and display the sum of them in the summary text using the same method "cat()" and "sum(is.na(dataEntred))" where sum function used to calculate the sum of the entered arguments and the (is.na) is used to specify the data "Missing Values" the program is looking for.

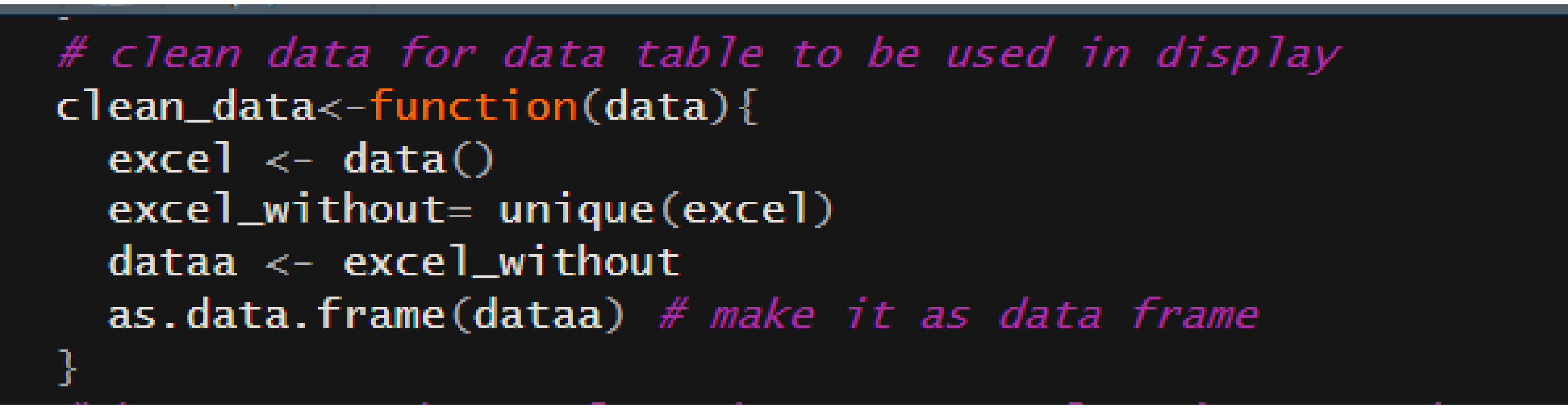
- Then the program will delete the duplicated unnecessary data using "unique()" function then store the new cleaned data in another variable.
- The data structure will be checked and displayed in the summary text using "is.(dataType)" data type can be [character – integer – numeric] then after checking the data structure it will be displayed by "print()" function.

Summary text will include:

- Items
- Count
- Total
- Rnd
- Age
- Customer names
- City
- Payment method
- Rows and columns

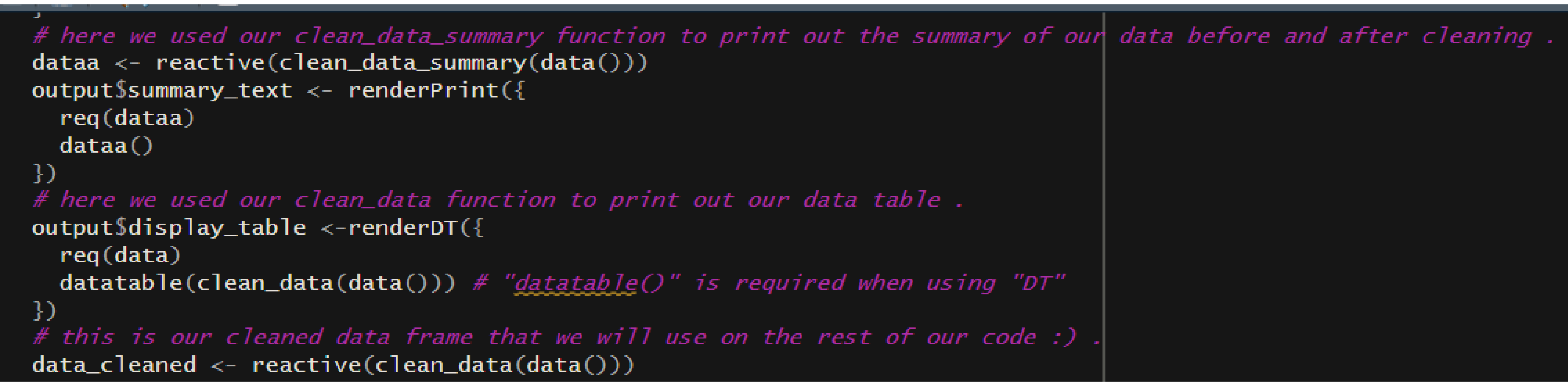


```
Sum of duplicated values in data:  2
Sum of null values in dataframe:  0
[1] "Checking data structure:"
[1] "Items: TRUE"
[1] "Count: TRUE"
[1] "Total: TRUE"
[1] "rnd: TRUE"
[1] "Customer: TRUE"
[1] "Age: TRUE"
[1] "City: TRUE"
[1] "PaymentType: TRUE"
Summary of data:
    items               count           total            rnd          
 Length:9833        Min.   : 1.00   Min.   : 100   Min.   : 1.000   L
 Class :character   1st Qu.: 2.00   1st Qu.: 679   1st Qu.: 4.000   C
 Mode  :character   Median : 3.00   Median :1297   Median : 8.000   M
                    Mean   : 4.41   Mean   :1293   Mean   : 8.009
                    3rd Qu.: 6.00   3rd Qu.:1897   3rd Qu.:12.000
                    Max.   :32.00   Max.   :2500   Max.   :15.000
Rows: 9833
Columns:  8
```



```
# clean data for data table to be used in display
clean_data<-function(data){
  excel <- data()
  excel_without= unique(excel)
  dataa <- excel_without
  as.data.frame(dataa) # make it as data frame
}
```

Here in this section data will be prepared to be displayed as data table by getting the cleaned data stored in a variable then converting it by "as.data.frame(data)" function to data frame.



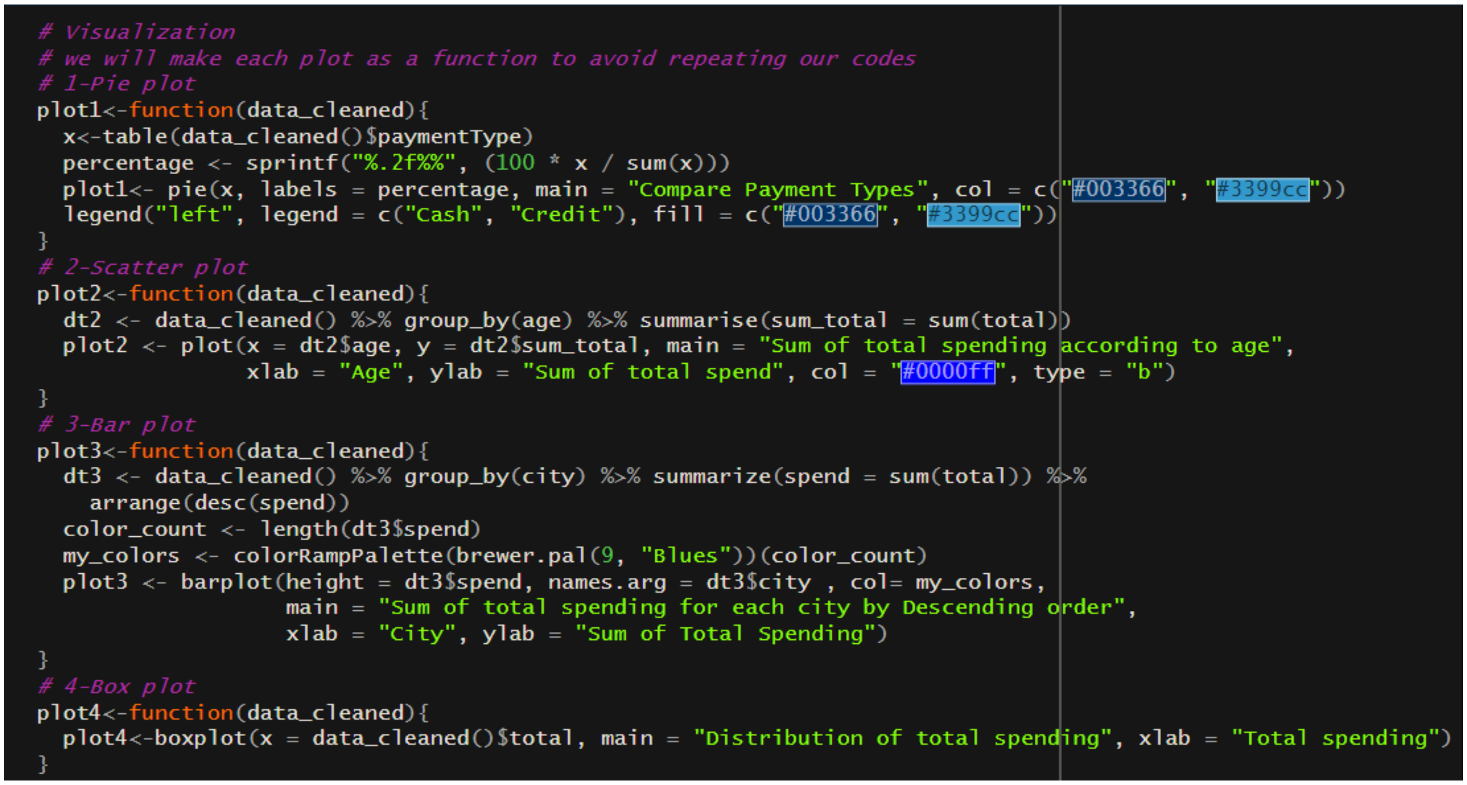
```
# here we used our clean_data_summary function to print out the summary of our data before and after cleaning .
dataa <- reactive(clean_data_summary(data()))
output$summary_text <- renderPrint({
  req(dataa)
  dataa()
})
# here we used our clean_data function to print out our data table .
output$display_table <-renderDT({
  req(data)
  datatable(clean_data(data())) # "datatable()" is required when using "DT"
})
# this is our cleaned data frame that we will use on the rest of our code :) .
data_cleaned <- reactive(clean_data(data()))
```

Another Comparison the program will make and display In the summary text where it is the data before and after cleaning using the "output$(requestedData)" to be displayed.

Using “DT” library the program will create a data table to display the data frame that made before using “renderDT()” function then “dataTable()” function.

Visualization

Visualization is the process where the data visualized in form of graphs to be easy to read and comparison.



```
# Visualization
# we will make each plot as a function to avoid repeating our codes
# 1-Pie plot
plot1<-function(data_cleaned){
  x<-table(data_cleaned()$paymentType)
  percentage <- sprintf("%.2f%%", (100 * x / sum(x)))
  plot1<- pie(x, labels = percentage, main = "Compare Payment Types", col = c("#003366", "#3399cc"))
  legend("left", legend = c("Cash", "Credit"), fill = c("#003366", "#3399cc"))
}
# 2-Scatter plot
plot2<-function(data_cleaned){
  dt2 <- data_cleaned() %>% group_by(age) %>% summarise(sum_total = sum(total))
  plot2 <- plot(x = dt2$age, y = dt2$sum_total, main = "Sum of total spending according to age",
               xlab = "Age", ylab = "Sum of total spend", col = "#0000ff", type = "b")
}
# 3-Bar plot
plot3<-function(data_cleaned){
  dt3 <- data_cleaned() %>% group_by(city) %>% summarize(spend = sum(total)) %>%
    arrange(desc(spend))
  color_count <- length(dt3$spend)
  my_colors <- colorRampPalette(brewer.pal(9, "Blues"))(color_count)
  plot3 <- barplot(height = dt3$spend, names.arg = dt3$city , col= my_colors,
                 main = "Sum of total spending for each city by Descending order",
                 xlab = "City", ylab = "Sum of Total Spending")
}
# 4-Box plot
plot4<-function(data_cleaned){
  plot4<-boxplot(x = data_cleaned()$total, main = "Distribution of total spending", xlab = "Total spending")
}
```

- The program makes four types of visualizations as functions to make us able to use it anywhere without writing the whole code again:

1. Pie Plot
2. Scatter Plot
3. Bar Plots
4. Box plots

Pie Plot: is the common method used to compare the percentage of

usage or consuming between

two or more things are in the

same category but with

different usage or function ex:

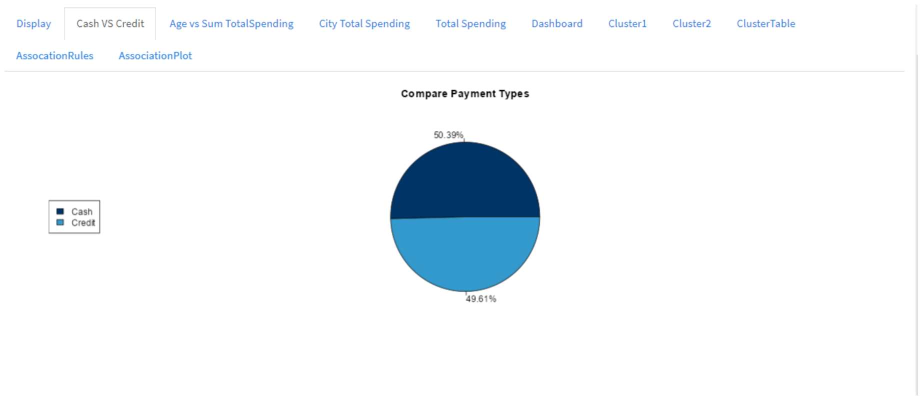
Credit and Cash.

payment methods have two

main types which are

(cash - credit) the pie plot will

calculate the payment method multiplied by hundred and divided by



the sum of total payment methods used as shown in the code to get

the percentage of each payment method and compare between

them in Pie diagram shape, Pie diagram is created using “Pie()”

syntax then entering arguments as label to define the way pie works,

“main()” to add a title, “legend()” to define position of the labels .

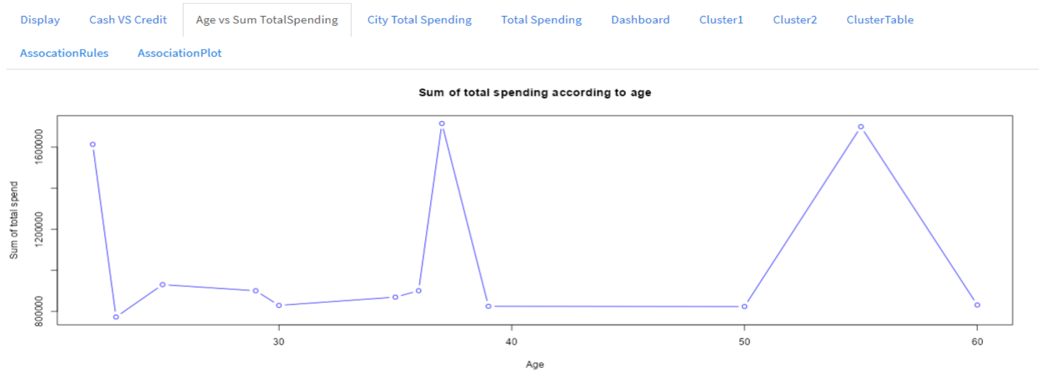
Scatter Plot: is a common way to get statistics of various data type

connected with the same action. Ex: Spending depending on age

Firstly, common data values will be grouped by “dplyr” library and

“group_by()” function as shown in code and the total spending for each age will be calculated using the “summarise ()” function and “sum()” argument then the new grouped and summarized data is stored in a variable that will be used in the scatter plot. Secondly the plot will be created using “plot syntax” and many

arguments will be inserted as “x” and “y” to display called data for each axis, “xlab and ylab” to add labels for each axis



Bar Plot: Common way to compare between unique data that share

common data. Ex: Cities and there total spendings as each city is

unique by its name but all cities share the spending data type

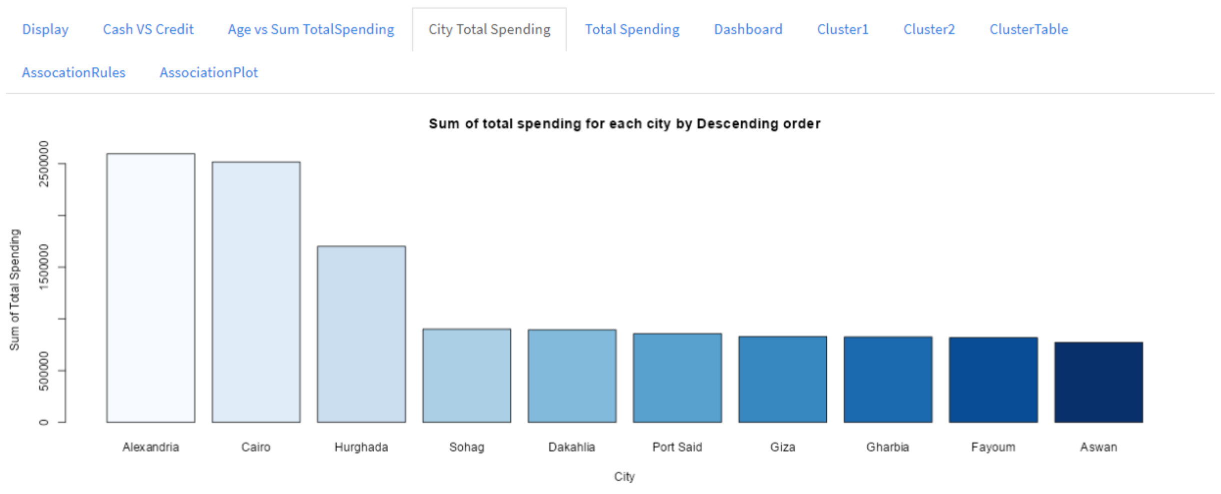
The program will do the same grouping method that is used in the

previous plot which is “group_by()” to group each customer’s city

and to calculate the sum of spending using “summarise()” function

and “sum()” arguments. Sort bars descending feature is added using “arrange(desc(spend))” function and arguments. Using

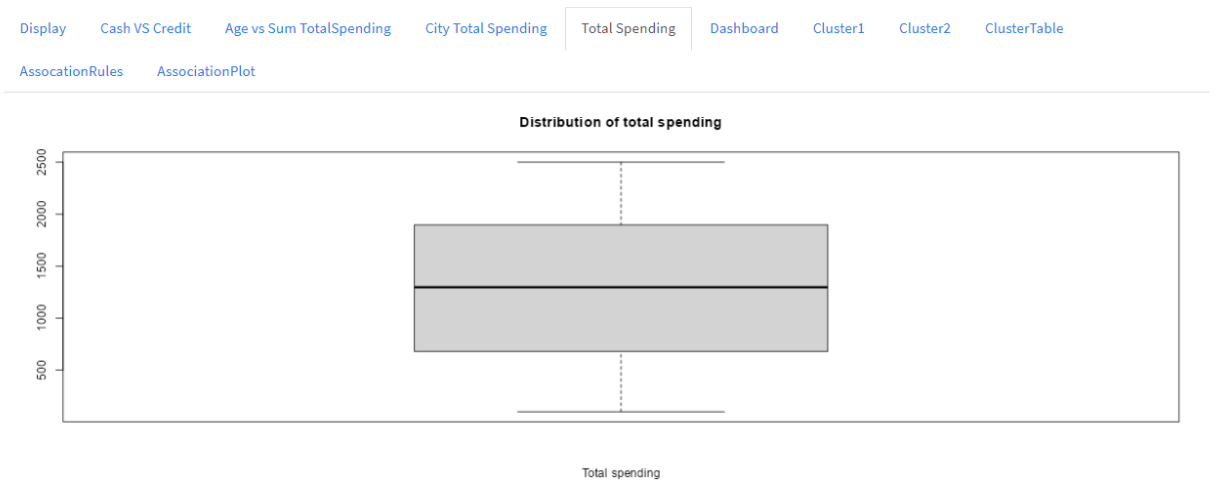
“RColorBrewer” library a special gradient color was added to make the bars more readable and attractive.



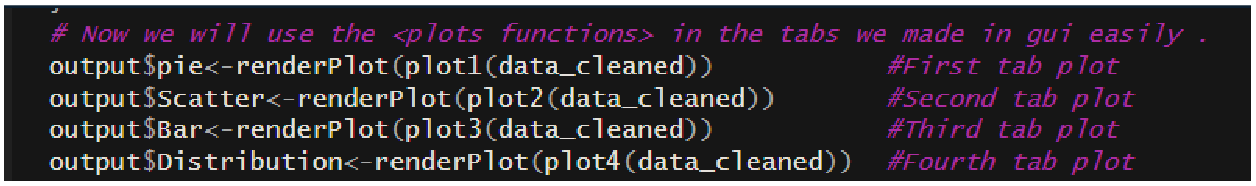
Box plot: is a way to show the distribution of data.

Boxplot made using "boxplot()" function and arguments as "x = data" to add specific data to x axis, "main" for adding a title and

"xlab" to add label to x axis.

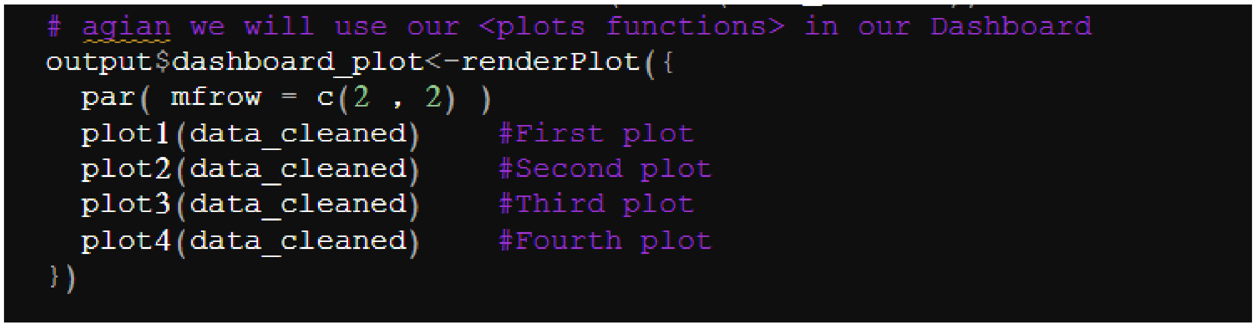


After giving each plot an ID it will be easy to display them in the Graphical User Interface.



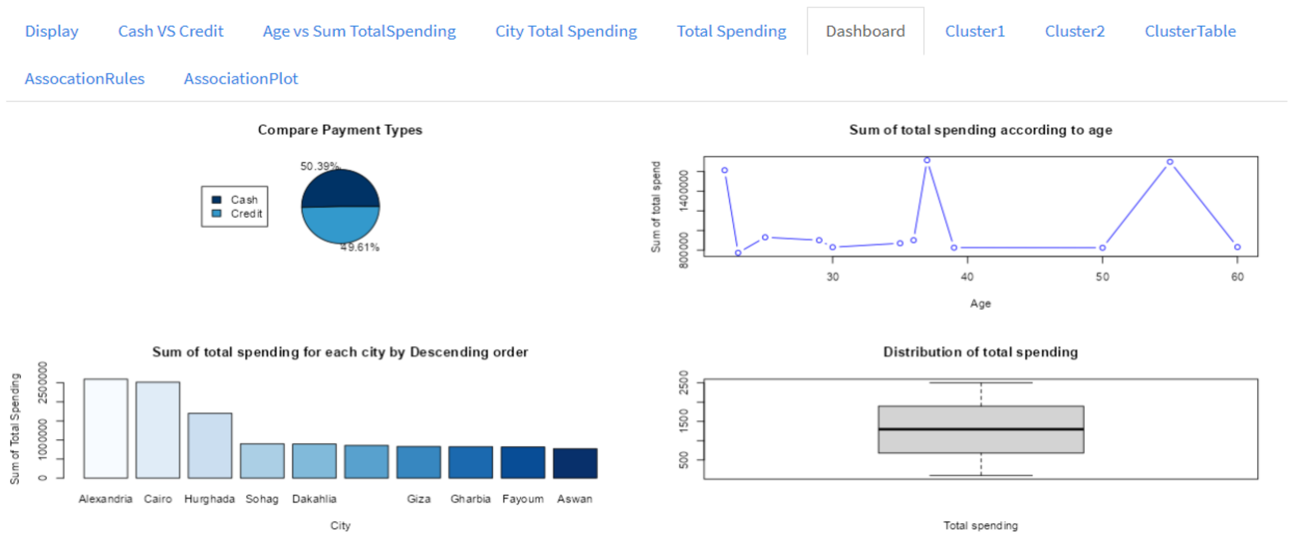
```
# Now we will use the <plots functions> in the tabs we made in gui easily .
output$pie<-renderPlot(plot1(data_cleaned))          #First tab plot
output$Scatter<-renderPlot(plot2(data_cleaned))      #Second tab plot
output$Bar<-renderPlot(plot3(data_cleaned))          #Third tab plot
output$Distribution<-renderPlot(plot4(data_cleaned)) #Fourth tab plot
```

Using the "Output$(plotOutput ID)" and entering the final cleaned data which is ready to be calculated in form of plots as an argument inside "plotID()"

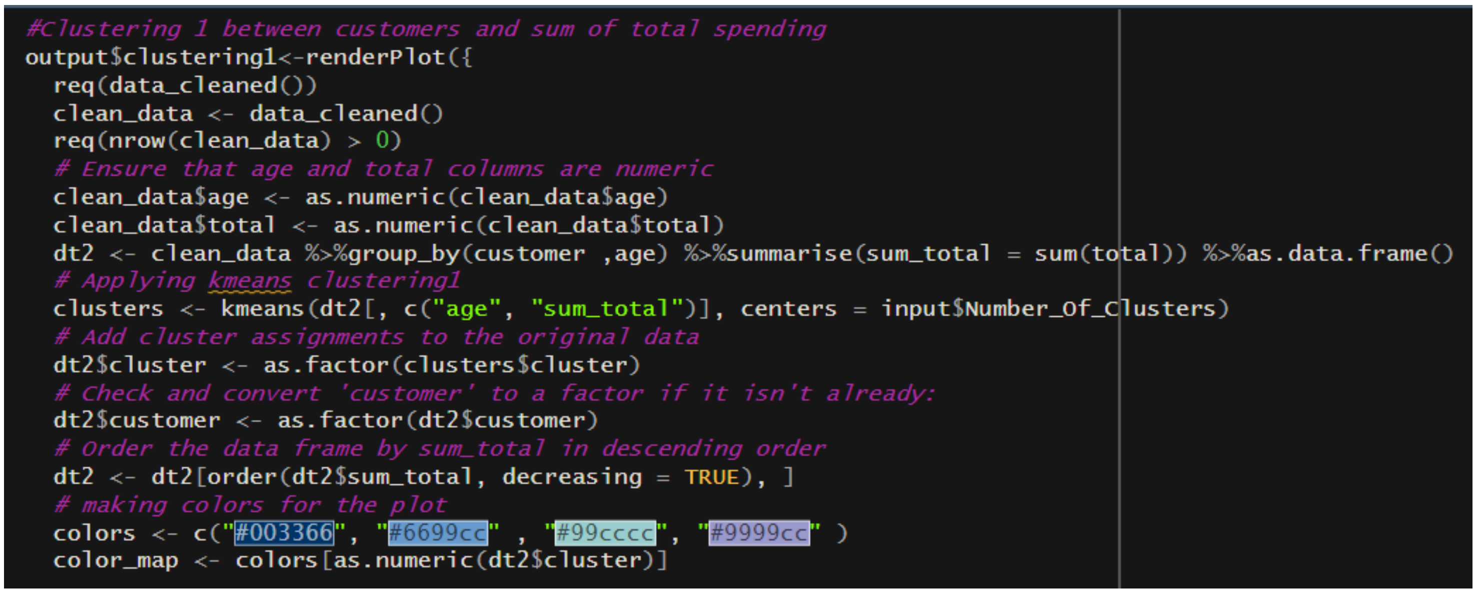


```
# agian we will use our <plots functions> in our Dashboard
output$dashboard_plot<-renderPlot({
  par( mfrow = c(2 , 2) )
  plot1(data_cleaned)    #First plot
  plot2(data_cleaned)    #Second plot
  plot3(data_cleaned)    #Third plot
  plot4(data_cleaned)    #Fourth plot
})
```

Dashboard will be easily created using the same "output$dashboard" method to be displayed in its tab. It will be separated to 4 equal parts each part will display a plot using the *"par(mfrow = c(2 , 2))".*



Clusters



```
#Clustering 1 between customers and sum of total spending
output$clustering1<-renderPlot({
  req(data_cleaned())
  clean_data <- data_cleaned()
  req(nrow(clean_data) > 0)
  # Ensure that age and total columns are numeric
  clean_data$age <- as.numeric(clean_data$age)
  clean_data$total <- as.numeric(clean_data$total)
  dt2 <- clean_data %>%group_by(customer ,age) %>%summarise(sum_total = sum(total)) %>%as.data.frame()
  # Applying kmeans clustering1
  clusters <- kmeans(dt2[, c("age", "sum_total")], centers = input$Number_Of_Clusters)
  # Add cluster assignments to the original data
  dt2$cluster <- as.factor(clusters$cluster)
  # Check and convert 'customer' to a factor if it isn't already:
  dt2$customer <- as.factor(dt2$customer)
  # Order the data frame by sum_total in descending order
  dt2 <- dt2[order(dt2$sum_total, decreasing = TRUE), ]
  # making colors for the plot
  colors <- c("#003366", "#6699cc" , "#99cccc", "#9999cc" )
  color_map <- colors[as.numeric(dt2$cluster)]
```

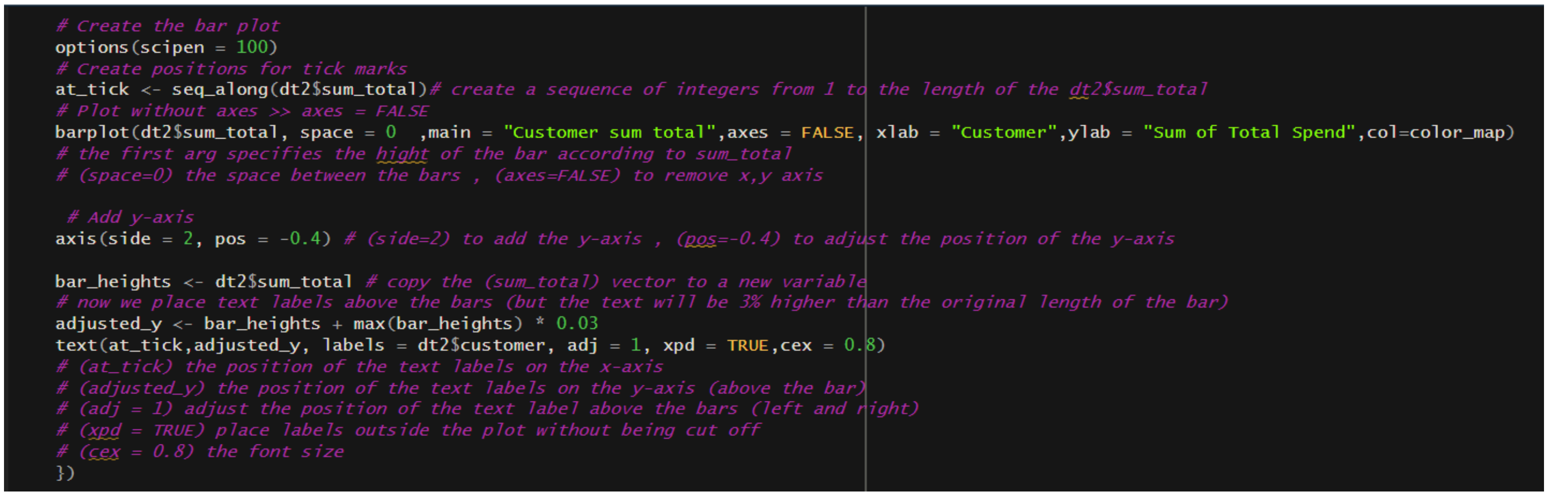
Program will make the first clustering between customers and the sum of total spendings. This process will happen after checking and converting the used data to the suitable structure using “as.numeric” function to convert data to numeric data type then the common names and ages will be grouped using “group_by()” function and entering column names that will be grouped as an argument. “Summarise()” function used to summarize and calculate many things using its unique arguments like: (sum() – mean() – mode()). In this block of code “Summarise()” function will calculate the sum of total spending using “sum()” arguments. The K-means will be applied using “Kmeans” function then entering the needed data to be clustered as an arguments, then select from the graphic user interface how many centers needed for the calculations from the “centers” argument.

The original (uncleaned data) will be assigned and added to the cluster using “as.factor()” function.

The same method will be applied again to another column (Customer) to convert it to factor.

The data frame made will be ordered using “order()” function and entering the key of sorting as argument.

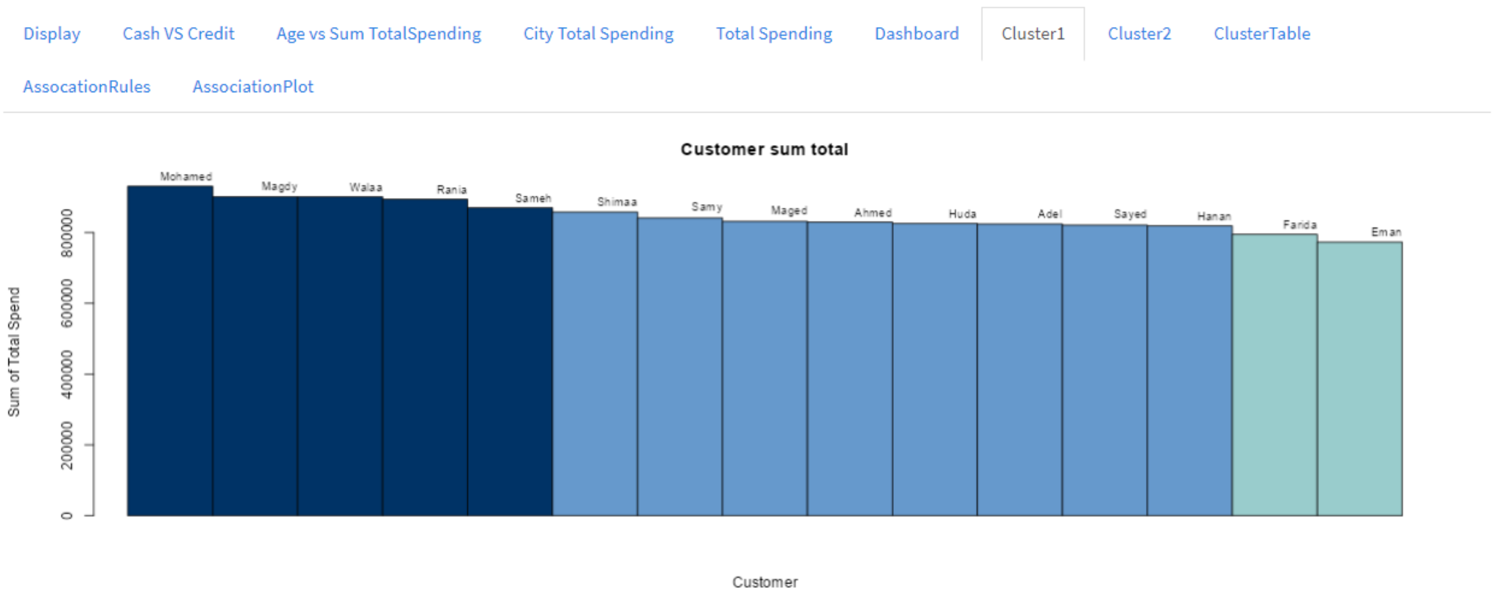
The program add some colorful gradient to the plots by “RColorBrewer” library that mentioned before in the libraries section. A color vector will be created then assigned to a variable named color to be used in coloring the cluster plots.



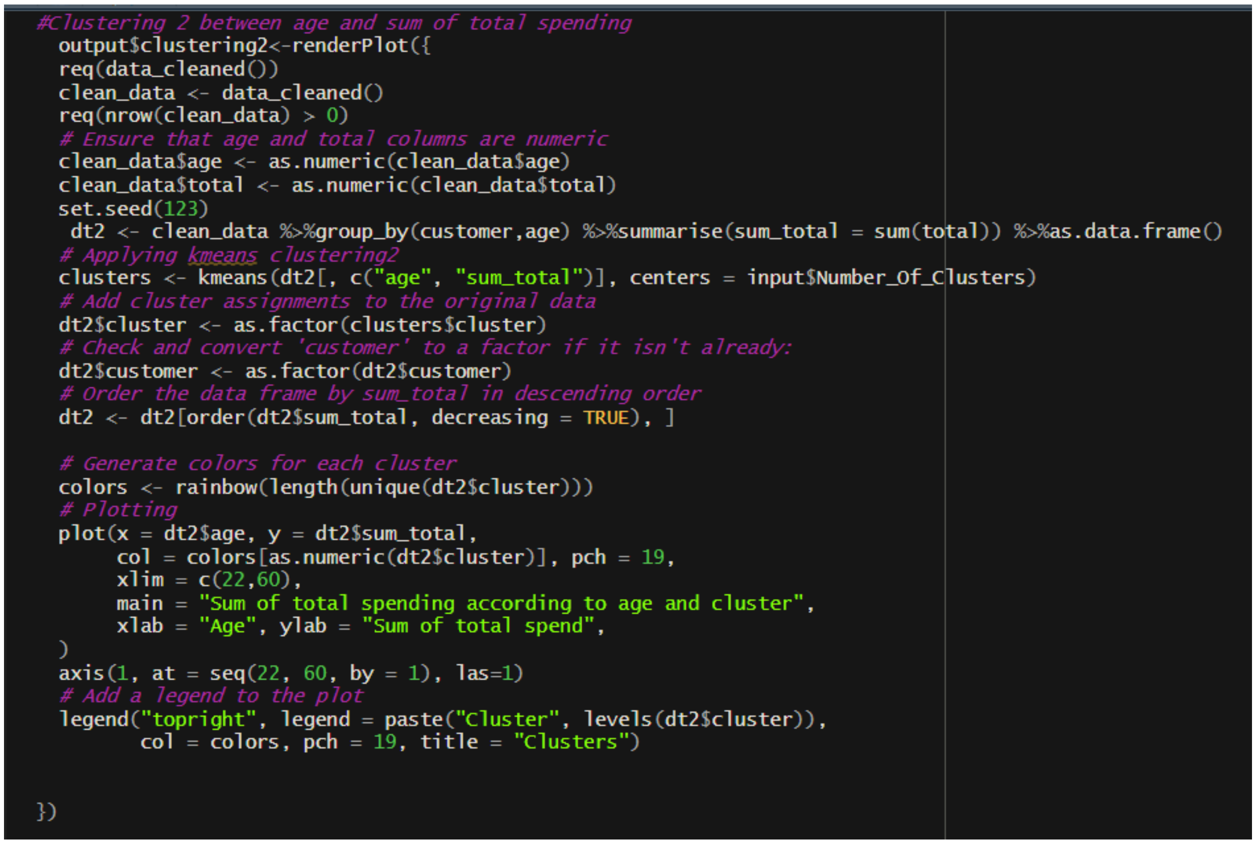
```
# Create the bar plot
options(scipen = 100)
# Create positions for tick marks
at_tick <- seq_along(dt2$sum_total)# create a sequence of integers from 1 to the length of the dt2$sum_total
# Plot without axes >> axes = FALSE
barplot(dt2$sum_total, space = 0  ,main = "Customer sum total",axes = FALSE, xlab = "Customer",ylab = "Sum of Total Spend",col=color_map)
# the first arg specifies the hight of the bar according to sum_total
# (space=0) the space between the bars , (axes=FALSE) to remove x,y axis

 # Add y-axis
axis(side = 2, pos = -0.4) # (side=2) to add the y-axis , (pos=-0.4) to adjust the position of the y-axis

bar_heights <- dt2$sum_total # copy the (sum_total) vector to a new variable
# now we place text labels above the bars (but the text will be 3% higher than the original length of the bar)
adjusted_y <- bar_heights + max(bar_heights) * 0.03
text(at_tick,adjusted_y, labels = dt2$customer, adj = 1, xpd = TRUE,cex = 0.8)
# (at_tick) the position of the text labels on the x-axis
# (adjusted_y) the position of the text labels on the y-axis (above the bar)
# (adj = 1) adjust the position of the text label above the bars (left and right)
# (xpd = TRUE) place labels outside the plot without being cut off
# (cex = 0.8) the font size
})
```

Bar plots will be created. First a position will be created for the tick marks by assigning the "seq_along(lengthKey)" function to a variable, this function create a sequence of integers from one to the length of the key entered in the function as an argument.

A plot without x and y axis created as "axes" argument set to false as the axis will be added manually. The first argument set the height of the bars according to the entered column. "Space" argument set to zero to keep no distance between bars. The axis will be added using "axis()" function, "side" argument set to two to add y-axis and the "pos" argument is used to set the position of the axis. By copying the bar blots height by assigning the height key to a variable named "bar_heights" it will be easy to add the height in equations as above each bar a text will appear to label each bar using "text()" function the first argument is the set tick map, the second argument is the height and position of the text as the text will be higher than the plot by three percent as calculated by the equation: (bar heights) + (maximum bar heights)*0.03. the "adj" argument adjust the position of the text label above the bars, the "xpd" argument set to true to place labels outside the plot without being cut off, "cex" argument is for setting font size.



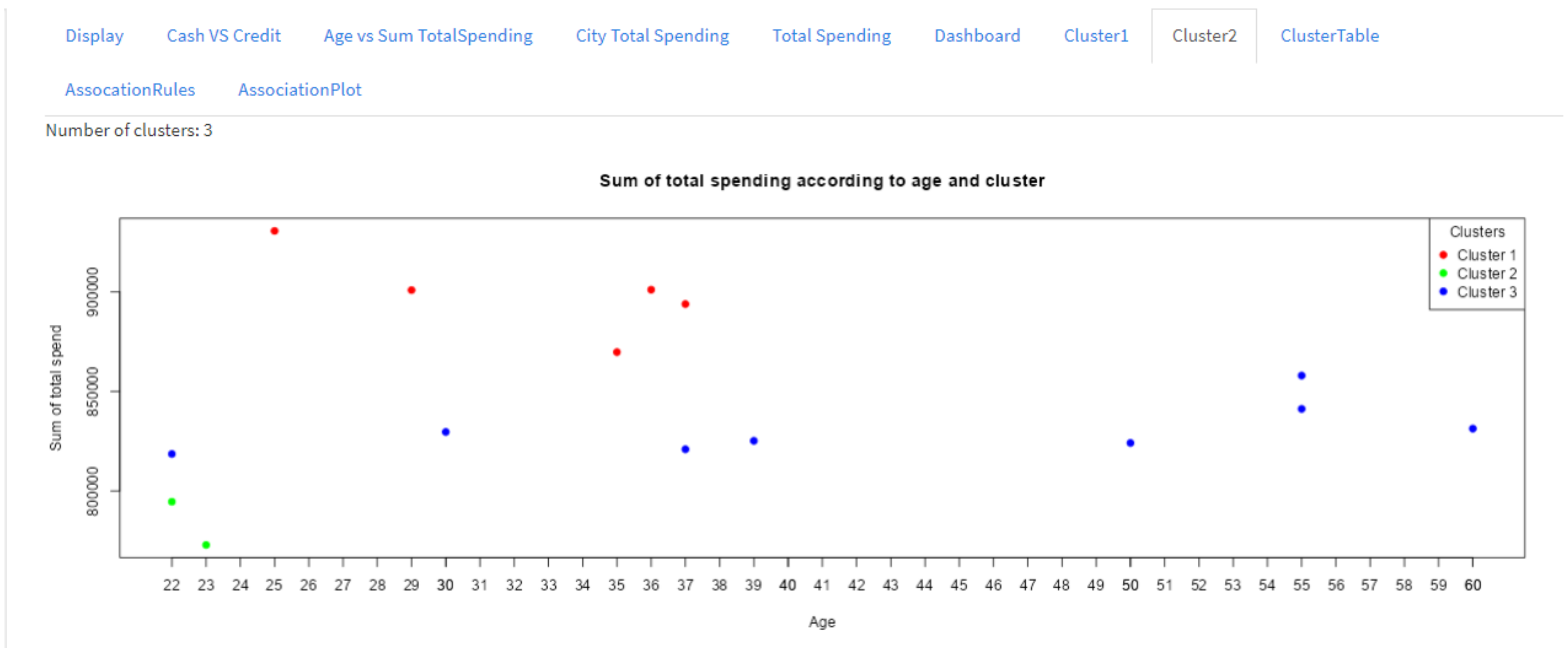
In the next part the second clustering “Cluster 2” will be explained.



```
#Clustering 2 between age and sum of total spending
  output$clustering2<-renderPlot({
  req(data_cleaned())
  clean_data <- data_cleaned()
  req(nrow(clean_data) > 0)
  # Ensure that age and total columns are numeric
  clean_data$age <- as.numeric(clean_data$age)
  clean_data$total <- as.numeric(clean_data$total)
  set.seed(123)
   dt2 <- clean_data %>%group_by(customer,age) %>%summarise(sum_total = sum(total)) %>%as.data.frame()
  # Applying kmeans clustering2
  clusters <- kmeans(dt2[, c("age", "sum_total")], centers = input$Number_Of_Clusters)
  # Add cluster assignments to the original data
  dt2$cluster <- as.factor(clusters$cluster)
  # Check and convert 'customer' to a factor if it isn't already:
  dt2$customer <- as.factor(dt2$customer)
  # Order the data frame by sum_total in descending order
  dt2 <- dt2[order(dt2$sum_total, decreasing = TRUE), ]

  # Generate colors for each cluster
  colors <- rainbow(length(unique(dt2$cluster)))
  # Plotting
  plot(x = dt2$age, y = dt2$sum_total,
       col = colors[as.numeric(dt2$cluster)], pch = 19,
       xlim = c(22,60),
       main = "Sum of total spending according to age and cluster",
       xlab = "Age", ylab = "Sum of total spend",
  )
  axis(1, at = seq(22, 60, by = 1), las=1)
  # Add a legend to the plot
  legend("topright", legend = paste("Cluster", levels(dt2$cluster)),
         col = colors, pch = 19, title = "Clusters")


})
```

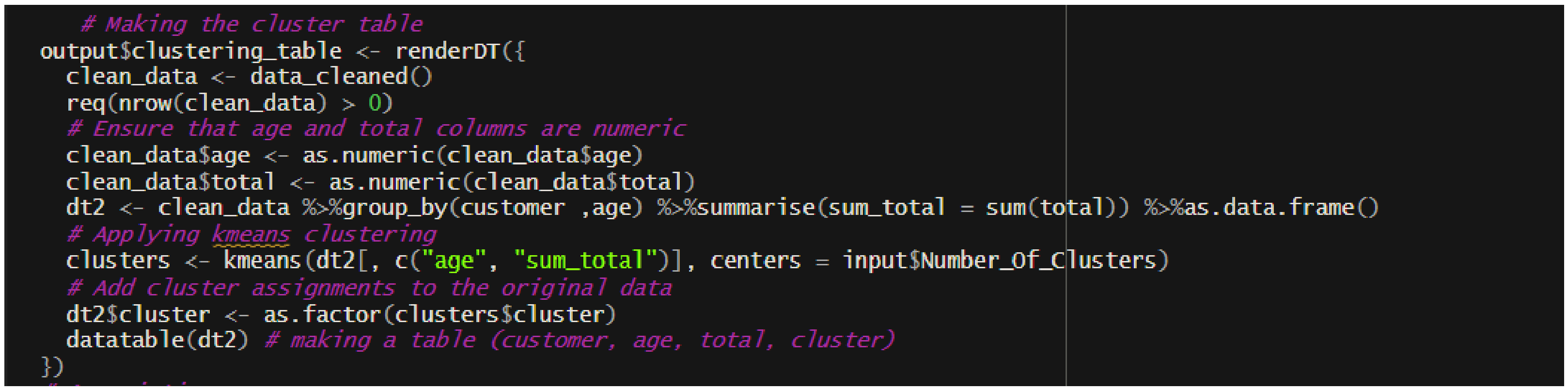
There is common methods already explained in the previous clustering. The difference between them is the plotting method as appeared here in this block of code, a “Plot()” function typed to plot the data between age and sum of total spending. The “x and y” arguments is for displaying the data along x and y axis.

“xlim” argument is to add limit to the x-axis and “xlab” argument to give x-axis a label. Legend added to the plot using the “legend()” function.



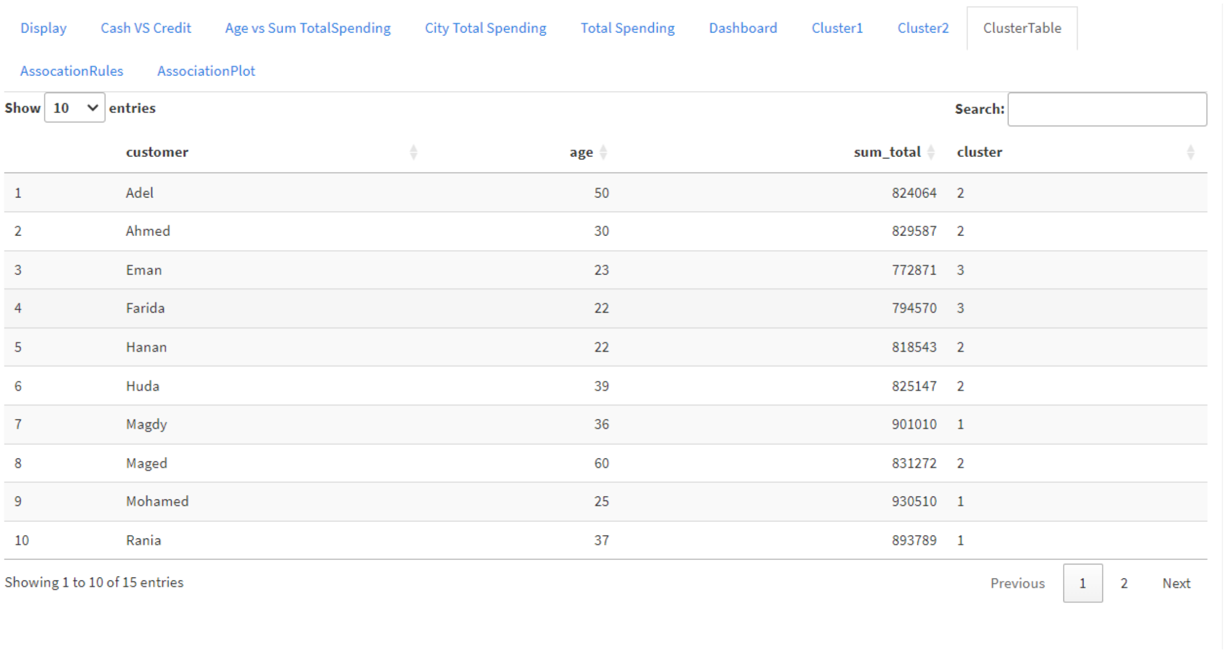
Number of clusters: 3

Cluster Table



```
   # Making the cluster table
output$clustering_table <- renderDT({
  clean_data <- data_cleaned()
  req(nrow(clean_data) > 0)
  # Ensure that age and total columns are numeric
  clean_data$age <- as.numeric(clean_data$age)
  clean_data$total <- as.numeric(clean_data$total)
  dt2 <- clean_data %>%group_by(customer ,age) %>%summarise(sum_total = sum(total)) %>%as.data.frame()
  # Applying kmeans clustering
  clusters <- kmeans(dt2[, c("age", "sum_total")], centers = input$Number_Of_Clusters)
  # Add cluster assignments to the original data
  dt2$cluster <- as.factor(clusters$cluster)
  datatable(dt2) # making a table (customer, age, total, cluster)
})
```

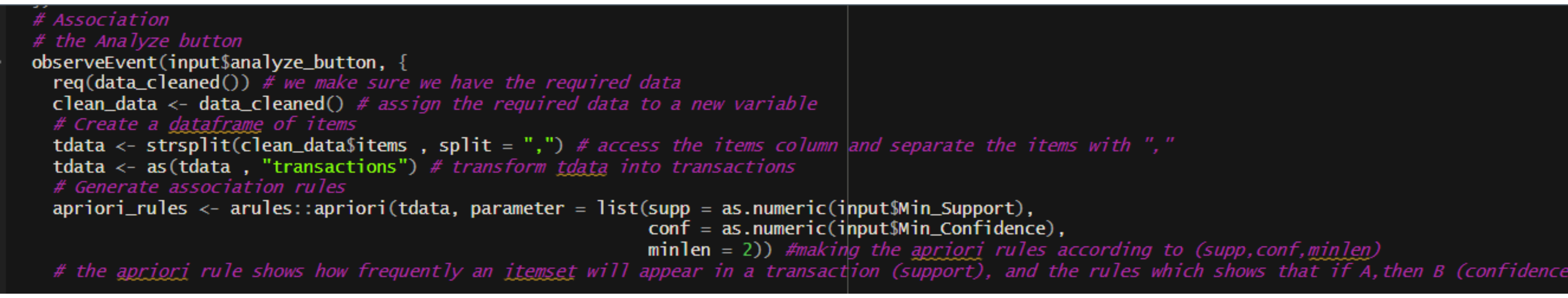
Cluster table will be created. First the data that will be displayed in the table will be converted to numeric using the “as.numeric()” function. Then the K-means will be applied to the data using “kmeans()” function. Then the original data assigned to the added cluster. Finally the data will be added to the “datatable()” function to make a table of (Customer , age , total, cluster).



	customer	age	sum_total	cluster
1	Adel	50	824064	2
2	Ahmed	30	829587	2
3	Eman	23	772871	3
4	Farida	22	794570	3
5	Hanan	22	818543	2
6	Huda	39	825147	2
7	Magdy	36	901010	1
8	Maged	60	831272	2
9	Mohamed	25	930510	1
10	Rania	37	893789	1

Showing 1 to 10 of 15 entries

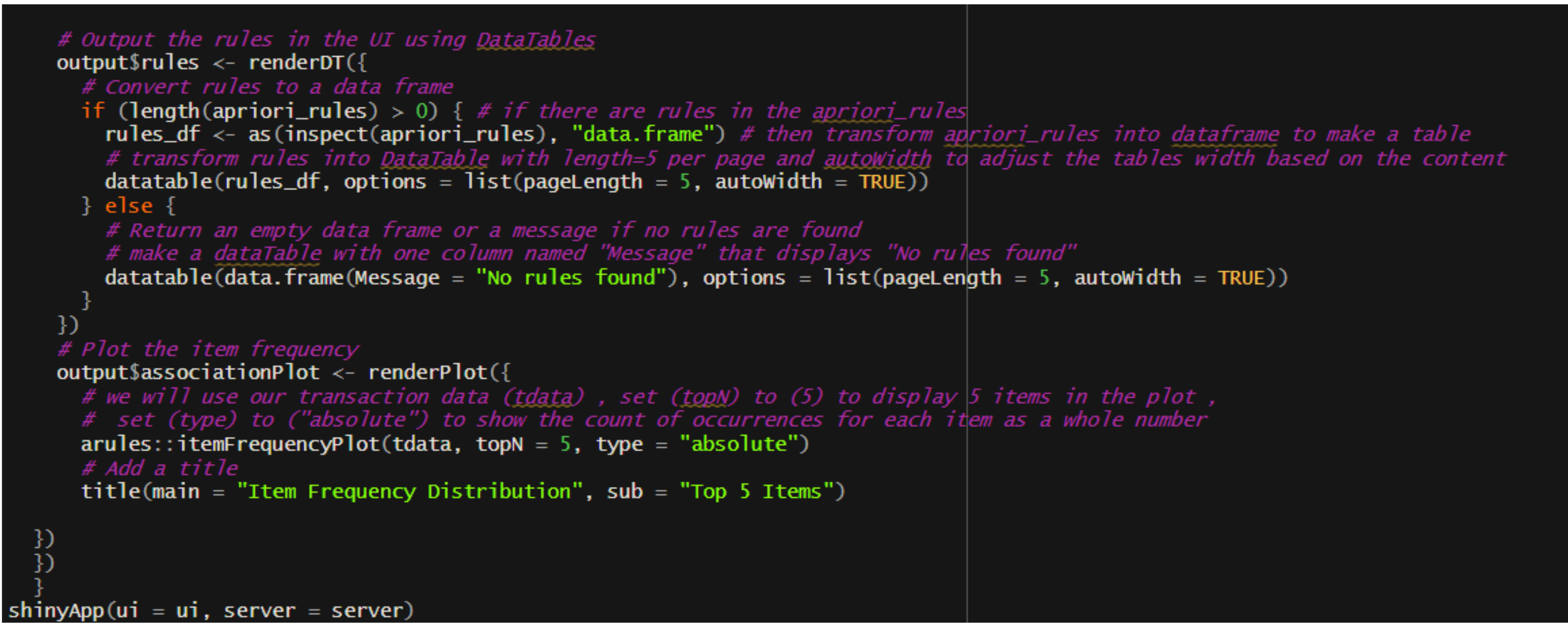
When the analyze button clicked the association rules will be calculated and applied.



```
# Association
# the Analyze button
observeEvent(input$analyze_button, {
  req(data_cleaned()) # we make sure we have the required data
  clean_data <- data_cleaned() # assign the required data to a new variable
  # Create a dataframe of items
  tdata <- strsplit(clean_data$items , split = ",") # access the items column and separate the items with ","
  tdata <- as(tdata , "transactions") # transform tdata into transactions
  # Generate association rules
  apriori_rules <- arules::apriori(tdata, parameter = list(supp = as.numeric(input$Min_Support),
                                                           conf = as.numeric(input$Min_Confidence),
                                                           minlen = 2)) #making the apriori rules according to (supp,conf,minlen)
  # the apriori rule shows how frequently an itemset will appear in a transaction (support), and the rules which shows that if A,then B (confidence
```

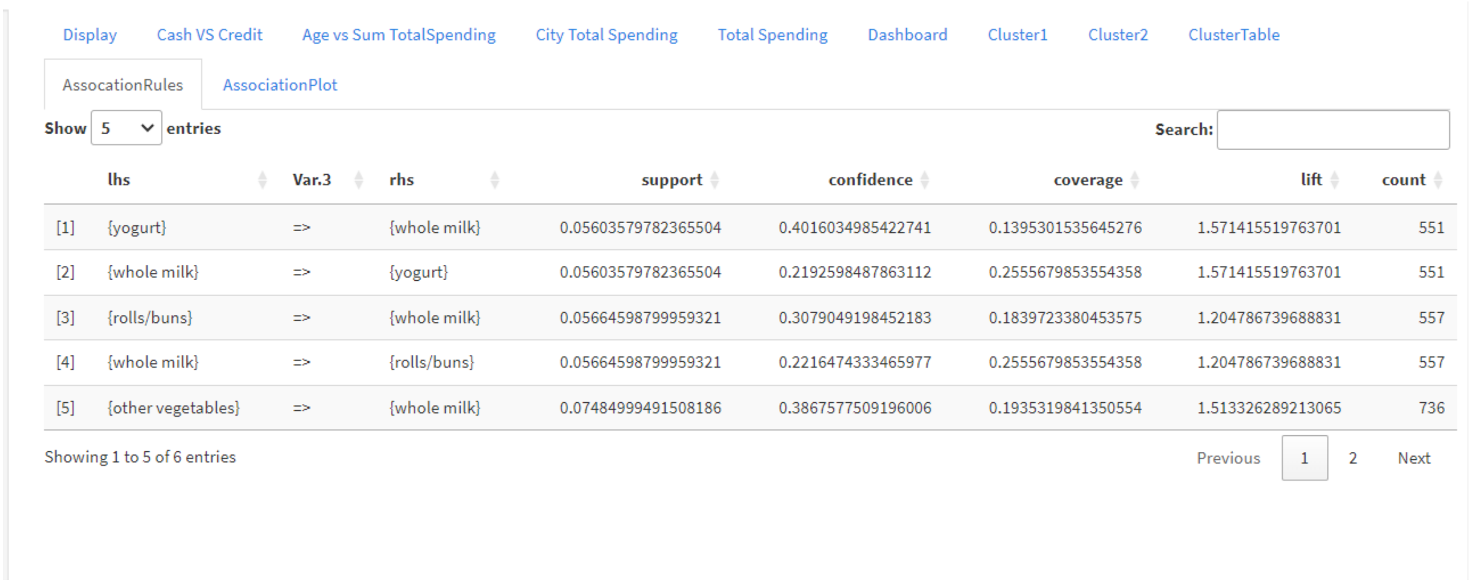
A data frame of items will be created using the “Strsplit()” to split between items using a separator “,” by the “split” argument. Then the data will be converted to transaction data using “as()” function, the first argument is the data needed to be converted to transaction data, the second argument is for the conversion type.

Apriori rules shows how frequently an itemset will appear in a transaction (support).Apriori rules will be applied using the “arules :: apriori()” function the first argument is the data the rules will be applied on, the rest of the arguments will be a list contain the (support – confident – minlen) to set how the rules work and depend on.
The rules will be displayed using the “output()” function.



```r
  # Output the rules in the UI using DataTables
  output$rules <- renderDT({
   # Convert rules to a data frame
   if (length(apriori_rules) > 0) { # if there are rules in the apriori_rules
    rules_df <- as(inspect(apriori_rules), "data.frame") # then transform apriori_rules into dataframe to make a table
    # transform rules into DataTable with length=5 per page and autoWidth to adjust the tables width based on the content
    datatable(rules_df, options = list(pageLength = 5, autoWidth = TRUE))
   } else {
    # Return an empty data frame or a message if no rules are found
    # make a dataTable with one column named "Message" that displays "No rules found"
    datatable(data.frame(Message = "No rules found"), options = list(pageLength = 5, autoWidth = TRUE))
   }
  })
  # Plot the item frequency
  output$associationPlot <- renderPlot({
   # we will use our transaction data (tdata) , set (topN) to (5) to display 5 items in the plot ,
   #  set (type) to ("absolute") to show the count of occurrences for each item as a whole number
   arules::itemFrequencyPlot(tdata, topN = 5, type = "absolute")
   # Add a title
   title(main = "Item Frequency Distribution", sub = "Top 5 Items")

 })
 })
 }
shinyApp(ui = ui, server = server)
```

The rules will be converted to data frames by “as()” function. Data table will be created by “datatables()” function the first argument is the data frames of the rules, the length per page will be set using the “options” argument and setting the length value.
In case the analyze button not clicked the “else” function will apply to return an empty set and warning message.

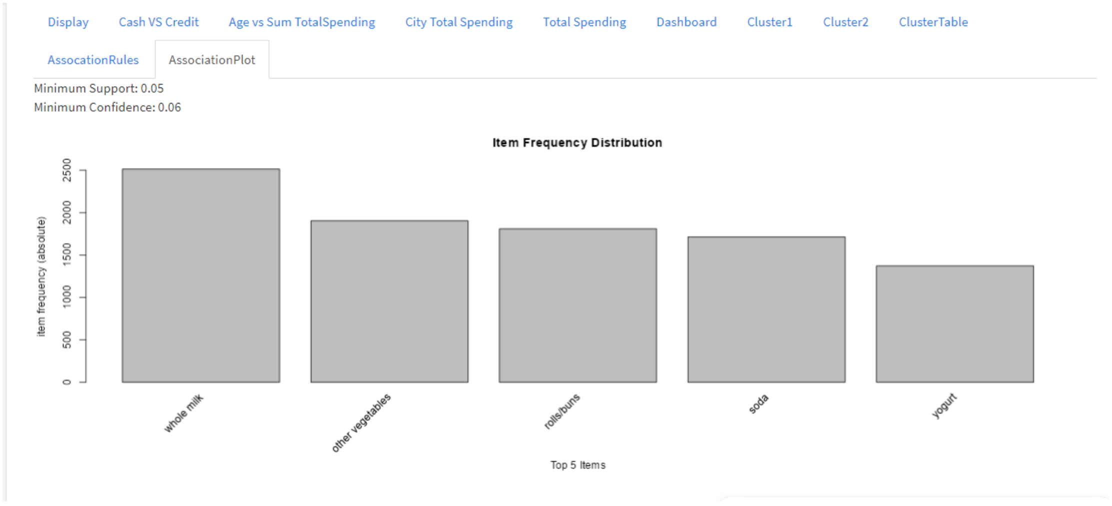


	lhs	Var.3	rhs	support	confidence	coverage	lift	count
[1]	{yogurt}	=>	{whole milk}	0.05603579782365504	0.4016034985422741	0.1395301535645276	1.571415519763701	551
[2]	{whole milk}	=>	{yogurt}	0.05603579782365504	0.2192598487863112	0.2555679853554358	1.571415519763701	551
[3]	{rolls/buns}	=>	{whole milk}	0.05664598799959321	0.3079049198452183	0.1839723380453575	1.204786739688831	557
[4]	{whole milk}	=>	{rolls/buns}	0.05664598799959321	0.2216474333465977	0.2555679853554358	1.204786739688831	557
[5]	{other vegetables}	=>	{whole milk}	0.07484999491508186	0.3867577509196006	0.1935319841350554	1.513326289213065	736

Showing 1 to 5 of 6 entries

Plotting item frequency

From the transaction data, the program will plot top five items frequency by "itemFrequency()" function and the argument "topN" to how many plot will be displayed then the "type" argument set to absolute to show the count of occurrences for each item as a whole number. A simple title added using "title()" function.



Minimum Support: 0.05
Minimum Confidence: 0.06