TEAM 36

DATA SCIENCE

```
information=data. Frame(
    names=c("Abdelrahman Ahmed Abdelmonem", "Ahmed Mohamed Abdullatif", "Adham Ibrahim Farouk","Khaled Nabil Fathy", "Manar Mohamed Abdelkarim”, Lamia Araby Abozaid Ahmed"),
    id=c(23011311,23011214,23011219,23011259,23011560,23011127)
  )
```

describing the role of each member

- Abdelrahman Ahmed Abdelmonem
  (the team leader and the Responsible for GUI and report writing)

- Ahmed Mohamed Abdullatif
  (Supervisor of creating the graphical user interface in addition to displaying the normal distribution of total spending)

- Adham Ibrahim Farouk
  (Responsible for cleaning data in addition to creating apriori algorithm)

- Khaled Nabil Fathy
  (Responsible for creating a comparison between cash and credit spending And create a dashboard) to place plots inside

- Manar Mohamed Abdelkarim
  (In partnership with Lamia, she used the kmeans algorithm to divide customers into groups, in addition to comparing ages and total spending)

- Lamia Araby Abozaid
  (In collaboration with Manar, they created the kmeans algorithm, in addition to comparing each city and its total spending)

- What is the input ? :

  The grc.csv file , the number of clusters , the min supp and the min conf

- What is the output ? :

  The 4 plots and kmeans table and association rules

problem description :

We have a large grocery store and our mission is to create an accurate database to increase the productivity of this store

First, we must take into account the presence of some corrupted data that may affect the accuracy of the results

In addition, there are some duplicate purchases and outliers

Through the approach that we created, we were able to limit these problems by cleaning the data and removing obstacles, and (Adham) gave an outstanding performance in it!

In the second stage, we were expected to create 4 visual graphs, which in turn would determine the natural and accurate distribution of the data

The first plot was a comparison between the total spending in cash and credit, and (Khaled) dealt with it using pie.

The second plot was to create a comparison between age and total spending, and (Manar) dealt with it using the scatterplot, which showed accuracy in the output data.

The third plot was to display the total expenditures for each Egyptian governorate and arrange them in descending order from the most to the least. We found that:

(Alexandria, Cairo, and Hurghada are the three governorates in terms of total spending)

(Lamia) used bar plot, which greatly clarified the data

The fourth plot, which was carried out by (Ahmed), is to display the normal distribution of spending data and display the maximum and highest value of spending and the average spending for all ages.

(Abdulrahman) then presented all of his team's data in a tab called (about us)

(Khaled) then put all the previous data into a dashboard to display them in a 2*2 format

(Manar) and (Lamia) cooperated in creating Cummins databases, which aim to divide customers into groups according to their total spending according to their age, in addition to obtaining the number of centers through user input.

(Adham) created dynamic tables to display purchasing transactions to predict the best product and the relationships of other products to each other, including the min value of support and the min value of confidence.

In the end, (Abdulrahman) and (Ahmed) created the Graphical user interface for you, have a happy journey in the world of Team 36

The Code :

```
library(shiny)

plot1 <- function(input){
  req(input$file_path)  # Require a file path input(call the file path)

  # Read the CSV file
  data <- read.csv(input$file_path, stringsAsFactors = FALSE)

  #compare age and sum of total spending
  library("dplyr")
  age_total <- data %>% group_by(age) %>% summarise(Total = sum(total))
  plot(
    age_total,
    ylab = "sum of total spending",xlab = "age",
    main = "compare age and sum of total spending",
    type = "b",
    col = "#144000",
    lwd = 2
  )
}

plot2 <- function(input){
  req(input$file_path)
  data=read.csv(input$file_path, stringsAsFactors = FALSE)

  groupedPayment <- data %>% group_by(paymentType) %>% summarise(Total = sum(total))
  pie(
    groupedPayment$Total,
    main = "Comparison between cash and credit totals",
    #getting the persentage of each section
    labels = paste(groupedPayment$paymentType, ": ", round(groupedPayment$Total /
sum(groupedPayment$Total) * 100, 2), "%"),
    col = c("#00ff72", "#7500ff"),  # Colors for cash and credit
  )
  legend("topright", legend = groupedPayment$paymentType,  fill =c("#00ff72", "#7500ff"))
}

plot3 <- function(input){
  req(input$file_path)
  data=read.csv(input$file_path, stringsAsFactors = FALSE)

  groupedCity <- data %>% group_by(city) %>% summarise(Total = sum(total))
  # Assuming your data frame is named 'groupedCity'
  sorted_groupedCity <- groupedCity[order(groupedCity$Total, decreasing = TRUE), ]
```

```r
  colors <-
c('#00ff72','#00ffaf','#59ffc0','#00fffd','#00cdff','#0087ff','#004aff','#0d00ff','#5200ff','
#7500ff')#nice Color gradation😍
  barplot(main = "Comparison between each city and its total sum",
          height = sorted_groupedCity$Total,
          name = sorted_groupedCity$city,
          xlab = "City",
          ylab = "Total Sum",
          col = colors
  )


}

# 144000
plot4 <- function(input){
  req(input$file_path)
  data=read.csv(input$file_path, stringsAsFactors = FALSE)
  # Assuming you have already created your box plot
  boxplot(x = data$total,
          main = "Distribution of Total Spending",
          xlab = "Total Spending",
          col = "#7500ff")

  # Calculate the statistics
  max_value <- max(data$total)
  min_value <- min(data$total)
  median_value <- median(data$total)

  # Add a legend
  legend("topright",  # Position of the legend
         legend = c(paste("Max:", max_value), paste("Min:", min_value), paste("Median:",
median_value)),  # Legend labels
         fill = c("#7500ff", "#7500ff", "#7500ff"),  # Colors (same as boxplot color)
         title = "Statistics",  # Legend title
         bty = "n",  # Remove the legend box
         text.col = "black",  # Text color
         cex = 0.8)  # Legend size


}
#collect all the visualization usage in a one dashboard
plot5 <- function(input)
{
  par(mfrow=c(2,2))
  plot1(input)
  plot2(input)
  plot3(input)
  plot4(input)
}
```

```r
# Define UI for application
# changing some colors in the window using the CSS or HTML
ui <- fluidPage(
  tags$head(
    tags$style(
      HTML("
        /* Change background color of the entire window */
        body {
            background: linear-gradient(to right, #59C173 , #a17fe0 ,#5D26C1); /* Adjust
colors as needed */        }
    /* Define custom CSS for slider handle background color */
    .irs-single, .irs-handle {
      background-color: #5D26C1 !important; /* Change to your desired color */
    }
    div{color : black;}
    /* Change tab name color */
      .nav-tabs > li > a {
        color: white; /* You can use color names or hex values */
      }
      ")
    )
  ),
  div(id = "Team 36",
      h2("Impossible is not the title of Team 36’s story"),
      p("Thank you from the bottom of my heart for this wonderful effort😍")
  ),
  # Application title
  titlePanel("Team #36"),

  # Sidebar layout with input and output
  sidebarLayout(
    sidebarPanel(
      # Text input for file path
      textInput("file_path", "Enter File Path:"),

      # Slider inputs
      sliderInput("clus", "Select Number of Clusters :",
                  min = 2, max = 4, value = 3),
      sliderInput("supp", "Select The Minimum Support :",
                  min = 0.001, max = 1, value = 0.05),
      sliderInput("conf", "Select The Minimum Confidence :",
                  min = 0.001, max = 1, value = 0.3)
    ),
    mainPanel(
      textOutput("result"),
      # Output for file path input
      verbatimTextOutput("file_path_output"),
      #create a right area to display your input
      verbatimTextOutput("clus_output"),
```

```
      verbatimTextOutput("supp_output"),
      verbatimTextOutput("conf_output"),

      tabsetPanel(
        #tabPanel("the text that will appear on the buttom",plotOutput("the plot name"))
        tabPanel("Age Vs Total Spending📈", plotOutput("plot1")),
        tabPanel("Cash Vs Credit💸", plotOutput("plot2")),
        tabPanel("City Total Spending📊", plotOutput("plot3")),
        tabPanel("Distrib of Total T.spending🧩", plotOutput("plot4")),
        tabPanel("Dashboard📋", plotOutput("plot5")),
        tabPanel("Association rules 👀" , dataTableOutput("plot6")),
        tabPanel("Kmeans 🍃", dataTableOutput("table1")),
        tabPanel("About Us 🤷", dataTableOutput("about"))
      )
    )
  )
)

# Define server logic
server <- function(input, output, session) {
  # Output for file path input
  output$file_path_output <- renderPrint({

    input$file_path
  })

  # Output for clusters
  output$clus_output <- renderPrint({
    paste("Number of Clusters = ", input$clus)

  })

  # Output for support
  output$supp_output <- renderPrint({
    paste("Minimum Support = ", input$supp)
  })

  # Output for confidence
  output$conf_output <- renderPrint({
    paste("Minimum Confidence = ", input$conf)
  })

  #plot1💏
  output$plot1 <- renderPlot({
    plot1(input)
  })
  #plot2🤯
  output$plot2 <-renderPlot({
    plot2(input)
  })
```

```
#plot3🤩
output$plot3 <-renderPlot({
  plot3(input)
})

#plot4🤟
output$plot4 <-renderPlot({
  plot4(input)
})
#plot5🥶
output$plot5 <-renderPlot({
  plot5(input)
})

output$table1 <- renderDataTable({
  req(input$clus)
  library("dplyr")
  req(input$file_path)
  data=read.csv(input$file_path, stringsAsFactors = FALSE)
  data_summary<- data %>% group_by(customer,age) %>% summarise(Total = sum(total))
  #get the summary of the data without the customer colomn
  without_customer<-data_summary[,-c(1)]
  set.seed(1)
  #create the kmeans-clusterig
  k_means<-kmeans(without_customer,centers =input$clus)
  #show the clusters
  k_means

  #adding the cluster vector to the data summary
  data_summaryy<-cbind(data_summary,k_means$cluster)
  #rename the forth colomn to "group" instade of default name
  colnames(data_summaryy)[4]="group"
  data_summaryy
  #convert the table to data frame
  LEVA=as.data.frame(data_summaryy)

})

output$plot6 <- renderDataTable({
  req(input$file_path)
  #read the csv path as a string without adding double qoutations
  data=read.csv(input$file_path, stringsAsFactors = FALSE)
  data=clean_data_KOL(data)
  #converting the first column into transaction data for the apriori algorithm
  trans <- unlist(data[, "items"])
  # to write the data into a text file
  writeLines(trans, "transactions.txt")
  #text connection allow you to convert the colomn to text file
  trans = read.transactions("transactions.txt", sep = ',')
```

```
    #using apriori
    apororo = apriori(trans,parameter = list(support= input$supp, confidence=input$conf
,minlen=2))
    #display the assosiation rules in the output area

    #convert to dataframe to use it in table (this function to convert the arules to
df(as.dataframe()doesn't work))
    df=DATAFRAME(apororo)

  })
  #our team information
  output$about <- renderDataTable({
    df=data.frame(
      names=c("Abdelrahmaen Ahmed Abdelmonem","Ahmed Mohamed Abdullatif","Adham Ibrahim
Farouk","Khaled Nabil Fathy","Manar Mohamed Abdelkarim","Lamia Araby AboZaid Ahmed"),
      id=c(23011311,23011214,23011219,23011259,23011560,23011127)
    )
  })


}
#clean data without removing the outliers
clean_data_KOL <- function(data) {
  sum(is.na(data)) # 0
  data <- na.omit(data) # no need
  sum(duplicated(data)) # 2
  data <- unique(data) # must
  return(data)
}

# Run the application
shinyApp(ui = ui, server = server)
```

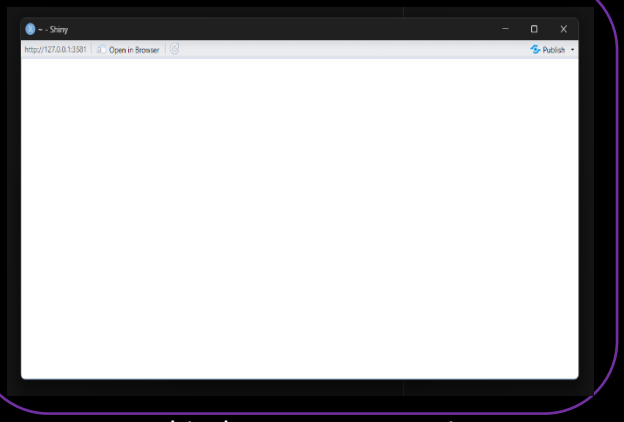
let's go to The main components of Shiny GUI 🤩

(Abdelrahman Ahmed & Ahmed Abdullatif)

```
library(shiny)
```

⚠️ this library is the god father of the Graphical User Interface in R & allows You to create all the GUI components

```
ui <- fluidPage(…)

server <- function(input, output, session) {…}

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



fluid page is the parameter that generate the Interactive window to put your graphical components on it

⚠️ server function reserves a place for you in the internal R servers and give you an URL to represent the GUI window in your Chrome or any Internet Browser

1- input in server function stands for the User interactions (textbox , sliders)

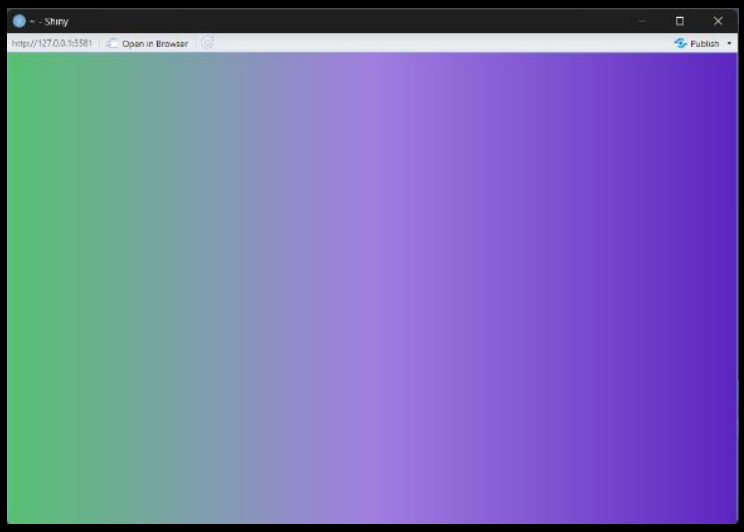
2- output in server function stands for The reactions of the program (tables , plots)

3- session in the server function stands for the current window that the program displays

Then Let's Go to the Ui fluid Page :

```
ui <- fluidPage(
  tags$head(
    tags$style(
      HTML("
        /* Change background color of the entire window */
        body {
            background: linear-gradient(to right, #59C173 , #a17fe0 ,#5D26C1); /* Adjust
colors as needed */        }
    /* Define custom CSS for slider handle background color */
    .irs-single, .irs-handle {
      background-color: #5D26C1 !important; /* Change to your desired color */
    }
    div{color : black;}
```

```
    /* Change tab name color */
      .nav-tabs > li > a {
        color: white; /* You can use color names or hex values */
      }
      ")
    )
  ),
)
```



⚠ This CSS & HTML codes changes the

1. back ground of the program

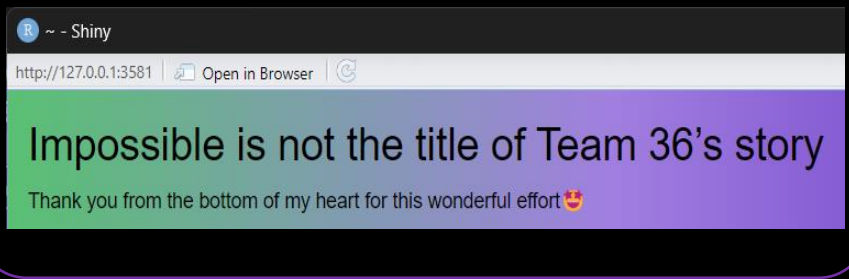
   in linear gradient mode
2. text color
3. slider hand color
4. tab name color

(all colors in hexadecimal code expect the tab name)

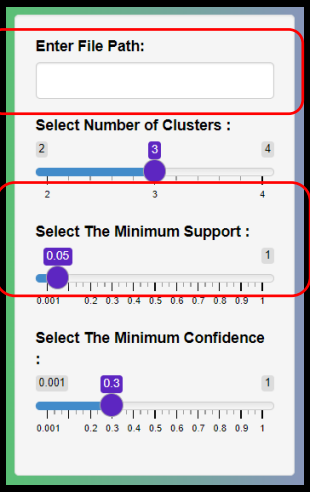
```
# Application title
  div(id = "Team 36",
      h2("Impossible is not the title of Team 36's story"),
      p("Thank you from the bottom of my heart for this wonderful effort😍")
  ),
```

⚠ Generate the following :

- label name
- heading level 2 element which typically represents the title.
- paragraph element which used for containing a subtext.



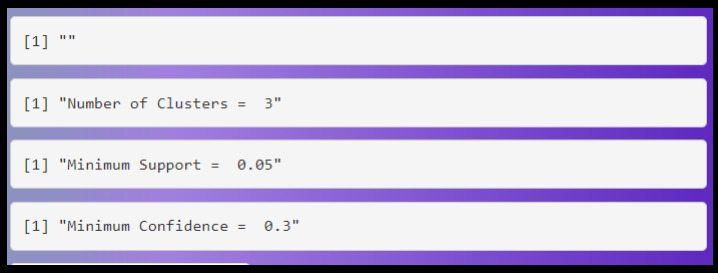
```
sidebarLayout(
    sidebarPanel(
      # Text input for file path
      textInput("file_path", "Enter File Path:"),

      # Slider inputs
      sliderInput("supp", "Select the minimum support:",
                  min = 0.001, max = 1, value = 0.05 )
    ),
```



⚠️ Generate an input tools like the text input field and the slider input

Main Panel :

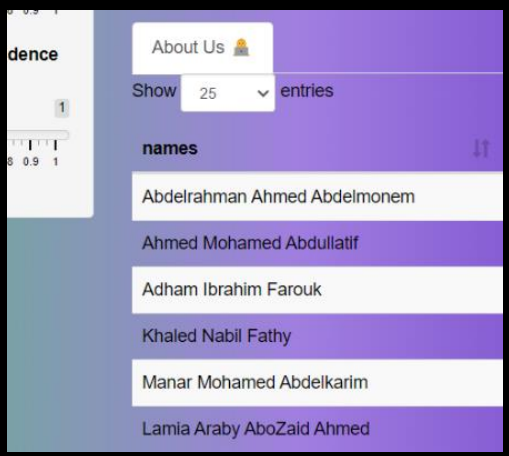
```
mainPanel(
      # Output for file path input
      verbatimTextOutput("file_path_output")
),
```

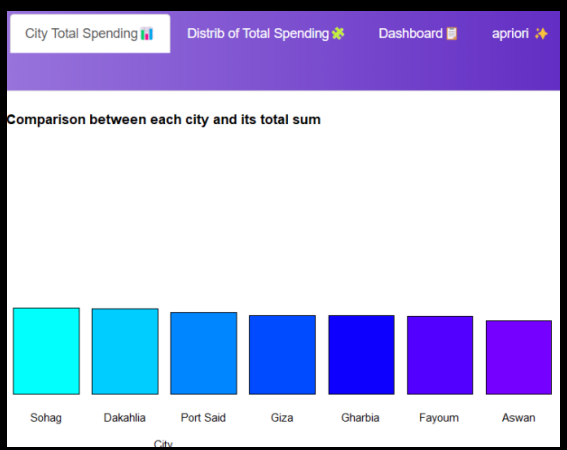


⚠️ books a place to display your input on the window as an output

```
tabsetPanel(
       tabPanel("Age Vs Total Spending📈", plotOutput("plot1")),
       tabPanel("apriori ✨" , uiOutput("plot6")),
       tabPanel("Kmeans 🍃", dataTableOutput("table1"))
     )
```

⚠️ Use to create a tab that you can put your data like (plots or tables)





In server function :

```
# Define server logic
server <- function(input, output, session) {
  # Output for clusters
  output$clus_output <- renderPrint({
    paste("Number of Clusters = ", input$clus)
  })
```

⚠️ These four functions allow you to display the inputs in the `verbatimTextOutput` on the main panel

```
output$plot1 <- renderPlot({
  plot1(input)
})

plot1 <- function(input){.....}
```

⚠️ This code is used for display the plots in the tabs

```
output$plot1 <- renderPlot({
      plot1(input)
    })
```

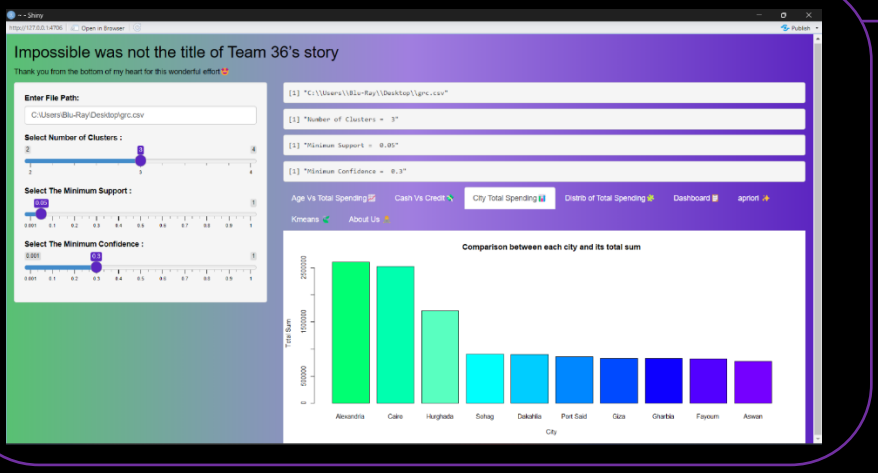
⚠️ this code repeated 5 times and this makes you able to generate the visualizations in a function and calling in in her tab

```
output$table1 <- renderDataTable({.....)}
```

⚠️ this code made to put the static tables on it like the plots (for kmeans only)

```
output$plot6 <- renderUI({.....})
```

⚠️ this code made to put the dynamic tables on it like the plots(for association rules only)



1 - Cleaning the data 🧹 (Adham Ibrahim)

We've discovered that there are two duplicated rows in our dataset by executing the following code in the console.

```
sum(duplicated(data))
```

```
We've discovered that the duplicated rows are 7025 and 9003.
           items count total rnd customer age       city paymentType
7025 canned beer     1   688  11    Hanan  22     Fayoum        Cash
9003        soda     1  1740   2  Mohamed  25 Alexandria        Cash
```

by using the following line of code:

```
print(data[which(duplicated(data)),])
```

We found no missing values in our data using the following line of code, which outputs 0:

```
sum(is.na(data))
```

Finally, We've ensured that we removed both the duplicated rows and any missing values from our data using the following code:

```
#clean data without removing the outliers
clean_data_KOL <- function(data) {
  sum(is.na(data)) # 0
  data <- na.omit(data) # no need
  sum(duplicated(data)) # 2
  data <- unique(data) # must
  return(data)
}
```

2 - Compare between Cash & Credit totals 💸

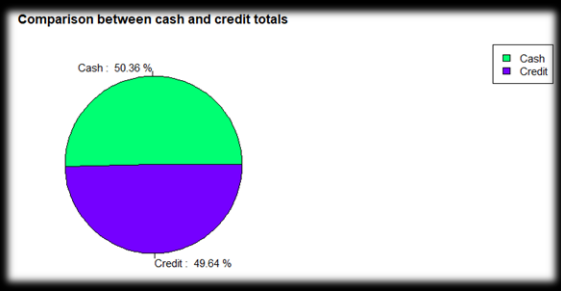
(Khaled Nabil)

First, we've divided the 'total' vector into 2 groups based on the 'paymentType' vector by using the function '`group_by`' and also applied the 'sum' function to the two groups:

```
groupedPayment <- data %>% group_by(paymentType) %>% summarise(Total = sum(total))
```

then we've created a pie chart to visualize this data:

```
pie(
    groupedPayment$Total,
    main = "Comparison between cash and credit totals",
    #getting the persentage of each section
    labels = paste(groupedPayment$paymentType, ": ", round(groupedPayment$Total /
sum(groupedPayment$Total) * 100, 2), "%"),
    col = c("#00ff72", "#7500ff
  )
  legend("topright", legend = groupedPayment$paymentType,
fill =c("#00ff72", "#7500ff"))
```



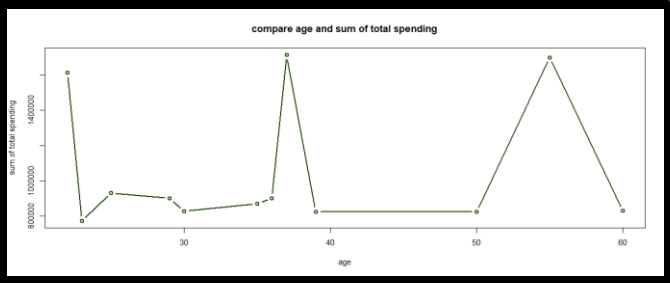
3 - Compare each age & sum of total spending(`library("dplyr")`) 📈 (Manar Mohamed)

Using the function "group_by," we first split the "total" vector into two groups according to the "age" vector. We then applied the "sum" function to the two groups:

```
age_total <- data %>% group_by(age) %>% summarise(Total = sum(total))
```

With the data represented by dots and lines, this scatter code plots the relationship between age (on the x-axis) and the total amount spent (on the y-axis). The plot employs a certain color and line width for the plotted items, and it features unique labels for the axes and a title.

```
plot(
    age_total,
    ylab = "sum of total spending",xlab = "age",
    main = "compare age and sum of total spending",
    type = "b",
    col = "#144000",
    lwd = 2
  )
```



4 - Comparison between each city and its total sum 📊 (Lamia Araby)

Initially, I saved the data in a variable named groupedCity after classifying it according to each city's total spending.

```
groupedCity <- data %>% group_by(city) %>% summarise(Total = sum(total))
```

I then saved the results of my descending order sorting the cities based on total spending in a variable named sorted_groupedCity.

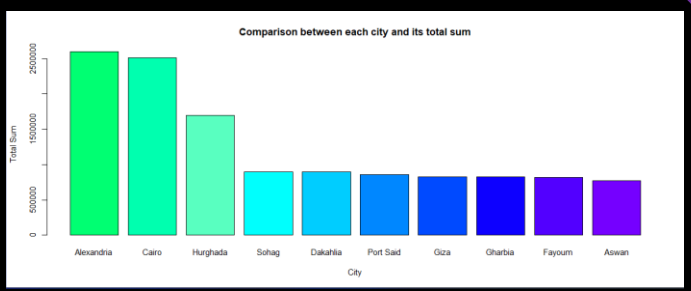
```
sorted_groupedCity <- groupedCity[order(groupedCity$Total, decreasing = TRUE), ]
```

Next, I saved the colors in a variable named colors and gradually assigned darker to lighter colors.

```
colors <-
c('#00ff72','#00ffaf','#59ffc0','#00fffd','#00cdff','#0087ff','#004aff','#0d00ff','#5200ff','
#7500ff')#nice Color gradation😎
```

- "Comparison between each city and its total sum" is the title I gave my bar plot after that. The horizontal axis is called "City" and represents the cities, while the vertical axis, representing the total spending, is labeled "Total sum". The bars' color and the colors variable match.

```
barplot(main = "Comparison between each city and its total sum",

          height = sorted_groupedCity$Total,
          name = sorted_groupedCity$city,
          xlab = "City",
          ylab = "Total Sum",
          col = colors)
```



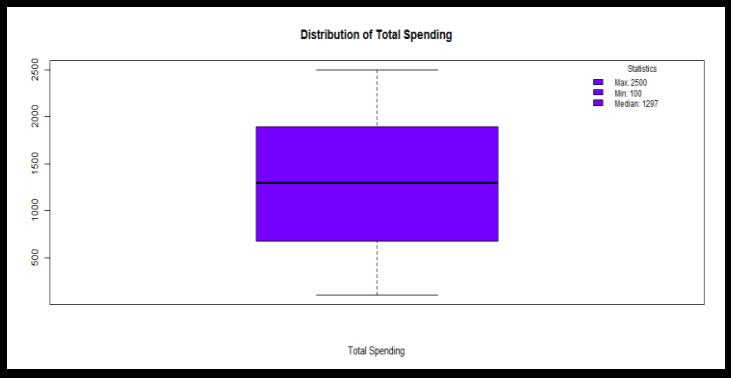
5 – Show the distribution Of total Spending 🧩

(Ahmed Abdullatif)

To show the distribution of total spending of our customers, we will use a box plot

to make a box plot, we first start with initializing the y-axis data which will be the total spending value

```
boxplot(x = data$total,
          main = "Distribution of Total Spending",
# we use argument main to set a title for the plot
          xlab = "Total Spending",
# we give a name for the x-axis
          col = "#7500ff")
# setting a color the box plot
```



Calculate the statistics

```
  max_value <- max(data$total)
  min_value <- min(data$total)
  median_value <- median(data$total)
```

now we have the min , max , median values, let's show them

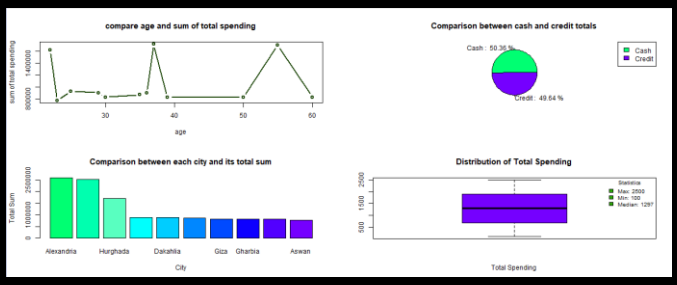
we will use a legend for showing them

```
  legend("topright",  # Position of the legend
         legend = c(paste("Max:", max_value), paste("Min:", min_value), paste("Median:",
median_value)),  # Legend labels
         fill = c("#7500ff", "#7500ff", "#7500ff"),  # Colors (same as boxplot color)
         title = "Statistics",  # Legend title
         bty = "n",  # Remove the legend box
         text.col = "black",  # Text color
         cex = 0.8)  # Legend size
```

now we have shown the distribution of total spending and have the min , max , median values

6- dashboard 📋 (Khaled Nabil)

```
#collect all the visualization usage in a one dashboard
plot5 <- function(input)
{
  par(mfrow=c(2,2))
  plot1(input)
  plot2(input)
  plot3(input)
  plot4(input)}
```



7-kmeans(library("dplyr")) 🌿 (Lamia Araby & Manar Mohamed)

I started by classifying the data according to age and consumers. After that, I combined all of the costs for every age group and stored them in a variable named data_summary.

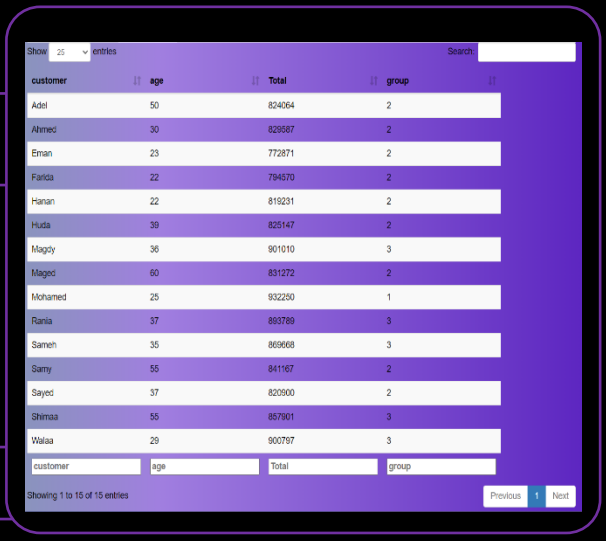
```
data_summary<- data %>% group_by(customer,age) %>% summarise(Total = sum(total))
```

I then performed k-means clustering on age and total expenses by removing the customer column from data_summary, and I saved the outcome in a new variable named without_customer.

```
#get the summary of the data without the customer colomn
without_customer<-data_summary[,-c(1)]
```

In order to guarantee consistent outcomes from the k-means clustering, I then employed a seed function.

```
set.seed(1)
```



Next, I let users enter the number of centers by

applying the k-means function to the without_customer variable, and I saved the outcomes in a variable named k_means.

```
k_means<-kmeans(without_customer,centers =input$clus)
#show the clusters
k_means
```

Afterwards, I used cbind to combine data_summary and k_means$cluster, saving the outcome in a variable called data_summaryy and changing the k_means$cluster column to group.

```
#adding the cluster vector to the data summary
    data_summaryy<-cbind(data_summary,k_means$cluster)
    #rename the forth colomn to "group" instade of default name
    colnames(data_summaryy)[4]="group"
```

Ultimately, I generated a DataFrame called LEVA and produced data_summary.

```
 #convert the table to data frame
    LEVA=as.data.frame(data_summaryy)
```

8-Association rules (`library(arules)`) ✨ (Adham Ibrahim)

Installing the “arules” package to be able to generate the association rules

```
install.packages("arules")
library(arules)
```

In the first , I Converted the first column into transaction data

```
#converting the first column into transaction data for the apriori algorithm
    trans <- unlist(data[, "items"])
```

This line unlists the values from the data frame's "items" column and extracts them into a single vector trans.

```
# to write the data into a text file
    writeLines(trans, "transactions.txt")
```

Reading the transaction data from the text file

```
trans = read.transactions("transactions.txt", sep = ',')
```

Using a comma (,) as the separator,
this line parses
the transaction data into transaction data format after reading it from the text file "transactions.txt".

Generating the Apriori :

```
#using apriori
apororo = apriori(trans,parameter = list(support= input$supp, confidence=input$conf
,minlen=2))
```

convert the rules to dynamic table using "DATAFRAME()"

```
#convert to dataframe to use it in table (this function to convert the arules to
df(as.dataframe()doesn't work))

df=DATAFRAME(apororo)
```

لقد ذهبت معنا الي ابعد الحدود , اذا انت مهتم حقا بتحقيق الأفضل لنا , نتمنتي لك حياة مليئة بالازدهار

الفريق ٣٦