Data Science

Names	ID	Participated in
Nureen Ehab Mahmoud Mohamed Barakat	20221465124	Writing the project report & putting all the graphs in one dashboard
Zainab Mohammed Abdallah Mohamed Aly	20221310251	Cleaning the data & splitting the customer into clusters
Malak Mahmoud Medhat Mahmoud Aref	20221445867	Writing the project report & generating association rules
Ereeny Wagih Massoud Hanaa Takla	2022513146	Comparing between cash and credit totals & each age and sum of total spending
Nouran Mohamed Mohamed Hemdan Hassan	20221321932	Arranging each city by total descending & display the distribution of total spending

First:

A.

- This project worked on a specific dataset which we should clean from it the redundant data by **descending the total spending** by this code:

```
products<- arrange (city, desc(total))
```

- Also, by getting all **the unique items in the city & customer columns** in the list by using the following code:

```
transactions=strsplit(as.vector(products$city),',')
unique_city<-unique(unlist(transactions))
```

&&

```
transactions=strsplit(as.vector(products$customer),',')
 unique_customer<-unique(unlist(transactions))
```

- After cleaning our data, we will use the function **write.csv** to save the result to a csv file

B.

i. Is to compare between cash and credit total numbers

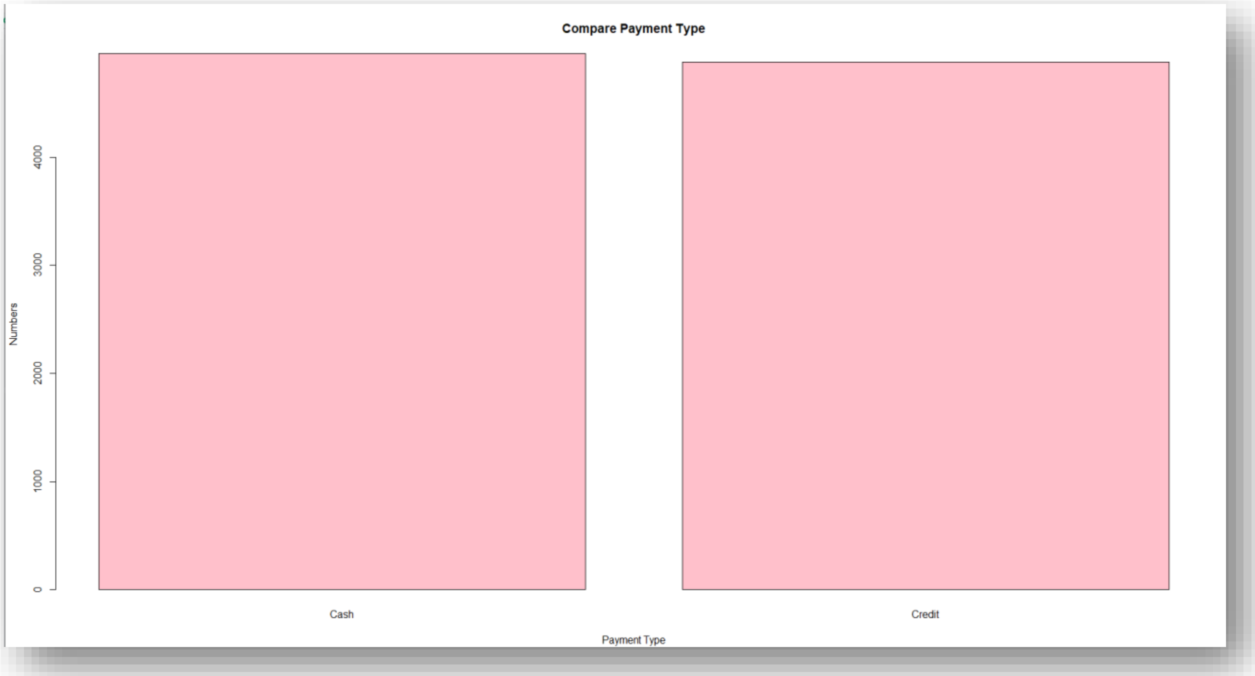
Our Input will contain **the contingency table to calculate the number of the total number of the cash and credit**, and we display the comparison by **barplot**

table(products$paymentType) the output of this code will be

```
Cash   Credit
 4957   4878
```

```
barplot (
  height = table(products$paymentType),
  col= "pink",
  main = "Compare Payment Type",
  xlab = "Payment Type",
  ylab = "Numbers",
)
```

The Output of this code is:

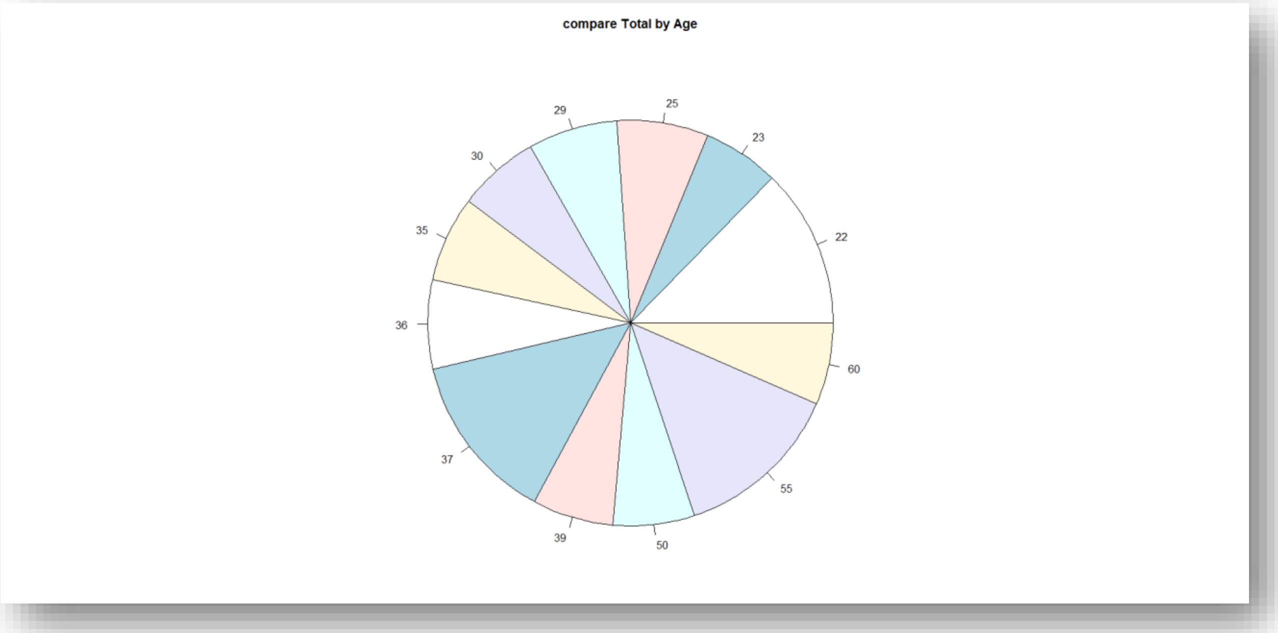


ii. Is to compare between each age and sum of total spending

The Input will contain function called **group by** to **separate the data frame which is age into groups**, and function called **summaries** to **summary the data and to unrepeat the data**, and we display the comparison by **pie chart**

```
totalPerage<-group_by(products,age)
totalPerage<-summarise(totalPerage,totaltotal=sum(total))
pie (
  x=totalPerage$totaltotal,
  labels=totalPerage$age,
  main="compare Total by Age"
)
```

The Output of this code is:

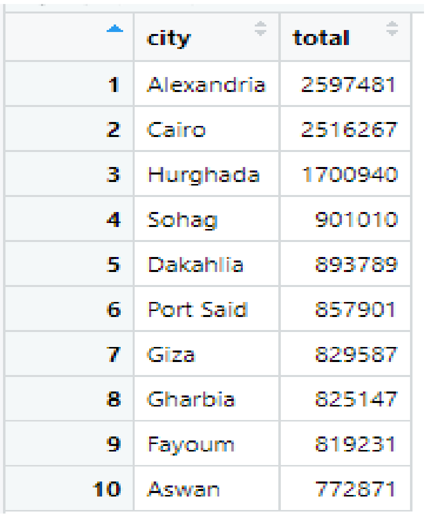


iii. Is to show each city total spending and arrange it by total descending.

The Input will contain function called **group by** to **separate the data frame which is city into groups**, and function called **summaries** to **summary the data and to unrepeat the cities**, also we will use the function **“desc”** to **descend the total spending according to the highest city**, and we display the comparison by **barplot**

```
totalPercity<-group_by(products,city)
totalPercity<-summarise(totalPercity,total=sum(total))
totalPercity<- arrange (totalPercity, desc(total))
```

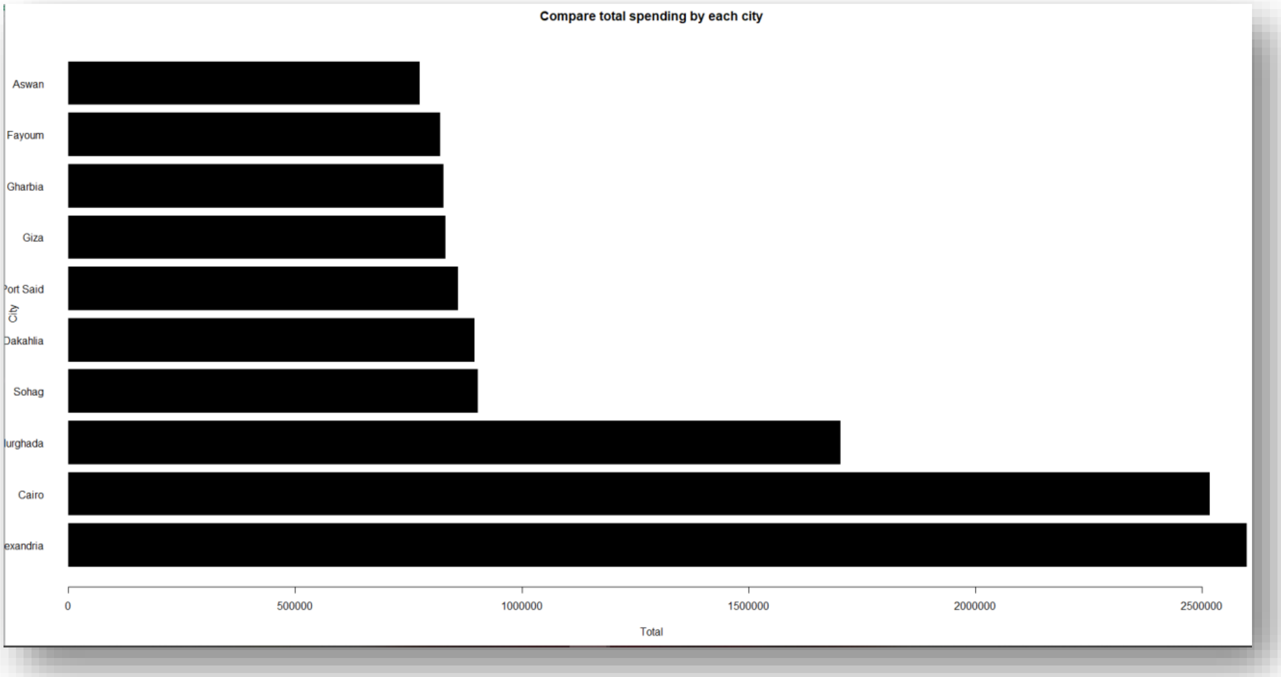
The Output of this code is:



	city	total
1	Alexandria	2597481
2	Cairo	2516267
3	Hurghada	1700940
4	Sohag	901010
5	Dakahlia	893789
6	Port Said	857901
7	Giza	829587
8	Gharbia	825147
9	Fayoum	819231
10	Aswan	772871

```
barplot (
 height=totalPercity$total,
 name=totalPercity$city,
 col="black",
main="Compare total spending by each city",
 xlab="Total",
 ylab="City",
 horiz = TRUE,
 las=1,)
```

The Output of this code is:

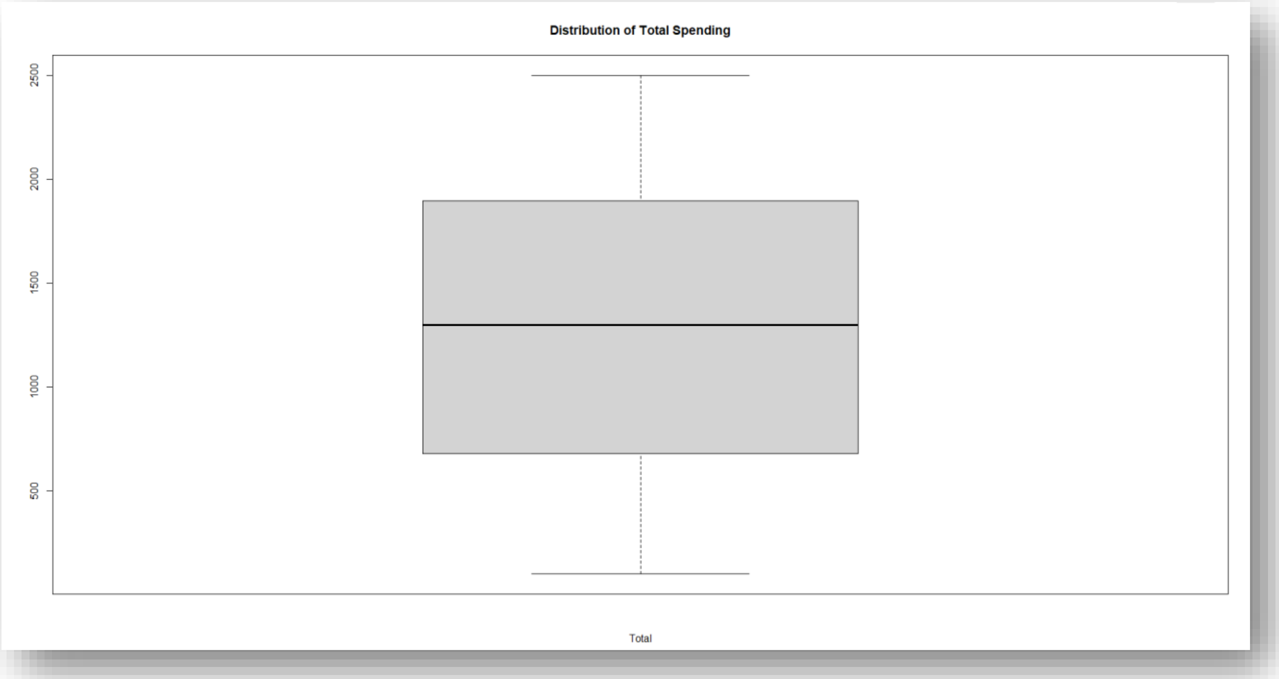


iv. Is to display the distribution of total spending.

The Input will display the comparison by **boxplot**

```
boxplot (
  x =products$total,
  main="Distribution of Total Spending",
  xlab="Total"
)
```

The Output of this code is



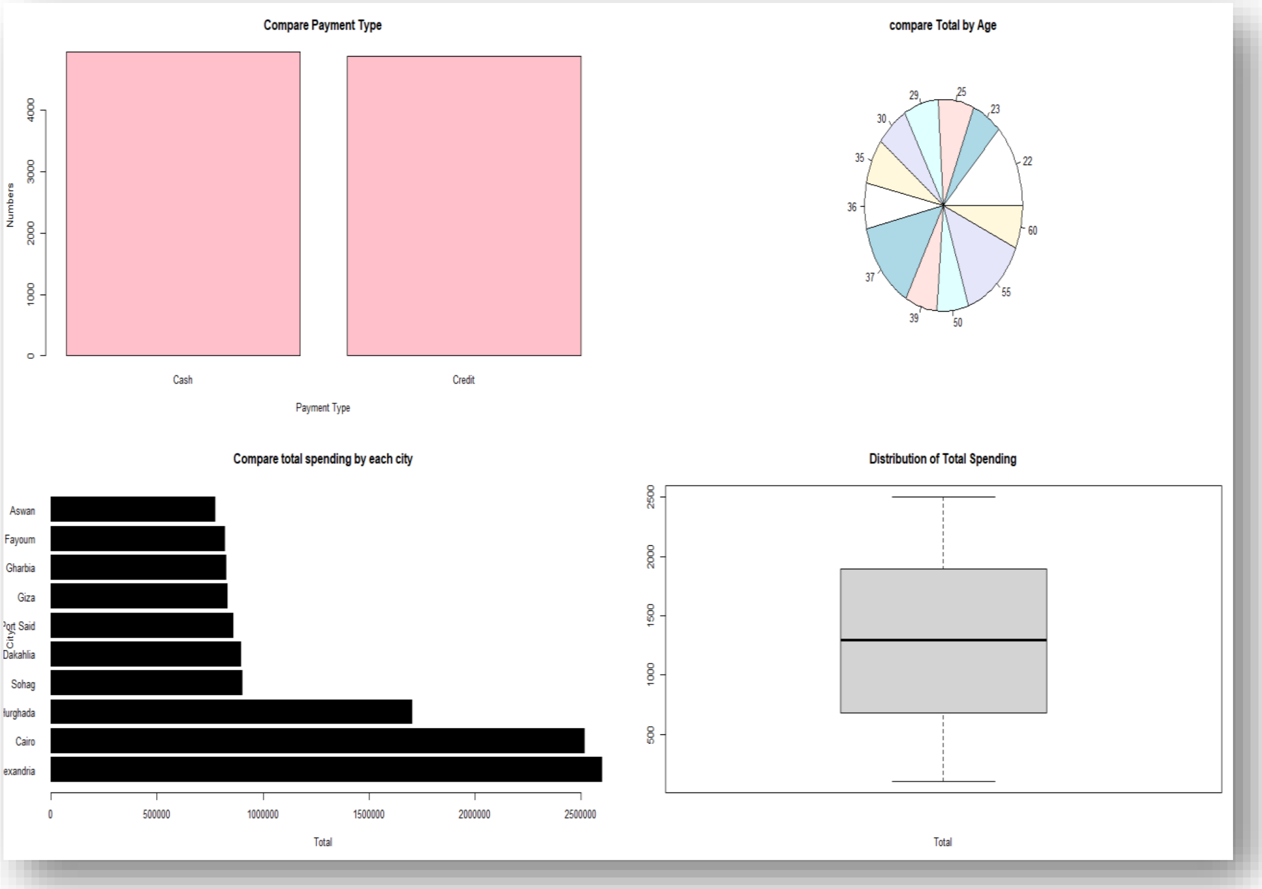
C. Is to put the previous graphs in one dashboard

The Input will display it by **par**

```
par (mfrow=c (2,2))
```

Then will add inside it all the codes of the previous graphs

The Output of this code is



D. Is to split the customers to (n) groups the user will enter it according to the sum of total spending and their ages and to print a table displaying name, age, total, and the computed cluster number

The Input will contain function called **group by** to **separate the data frame which is customer into groups** to show the **total spending of each customer**, and function called **summaries** to **summary the data and to unrepeat it**, then we will use the built-in function in R which is the **kmeans** to **split the customers into groups**.

- First, we should download **(The R Stats Package)**

```
library("stats")
```

- Then to initialize the data by the following code:

```
age <- c (50,29,37,39,30,35,60,36,55,22,55,22,25,37,23)
total<-c (824064, 829587, 772871, 794570, 819231, 825147, 901010, 831272, 932250,
893789, 869668, 841167, 820900, 857901, 900797)
dataPoints<-cbind(age,total)
colnames(dataPoints)<-c("Age(x)","Total(y)")
rownames(dataPoints)<-c
("Adel","Walaa","Rania","Huda","Ahmed","Sameh","Maged","Magdy",
              "Shimaa","Hanan","Samy","Farida","Mohamed","Sayed","Eman")
```

And we will **take the number of clusters from the user** by using function **“readline”**

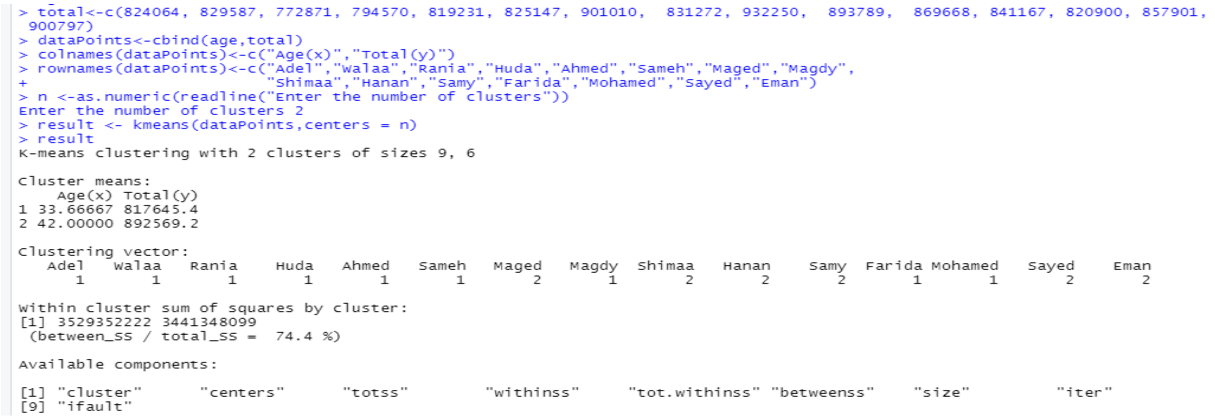
```
n <-as.numeric(readline("Enter the number of clusters"))
```

as if the user will enter n=2

then we will **calculate kmeans** using R-built in **kmeans** function

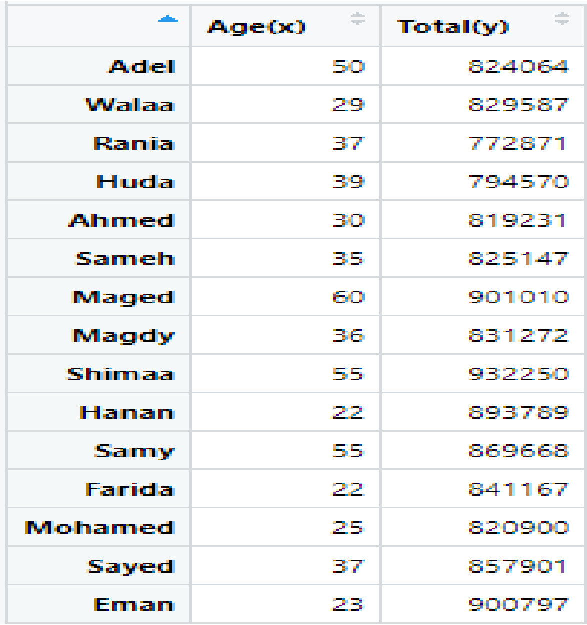
```
result <- kmeans(dataPoints,centers = n)
result
```

The Output of this code will be:

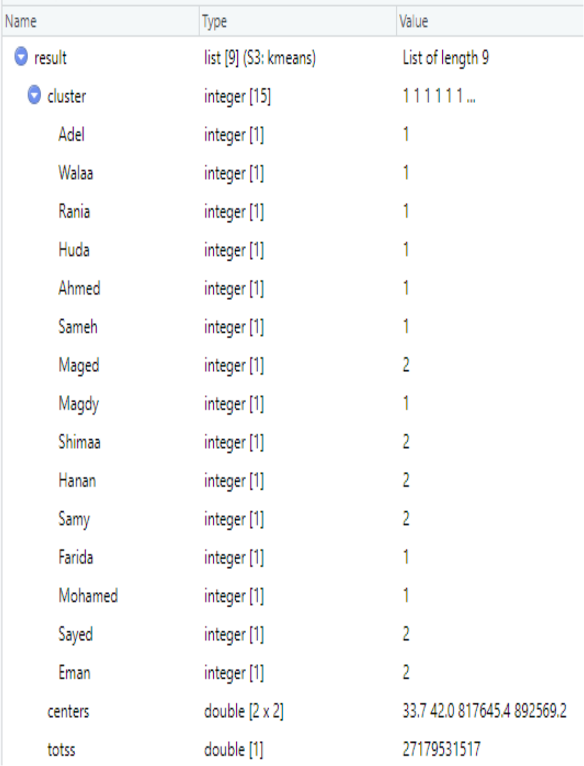


```
> total<-c(824064, 829587, 772871, 794570, 819231, 825147, 901010,  831272, 932250,  893789,  869668, 841167, 820900, 857901,
 900797)
> dataPoints<-cbind(age,total)
> colnames(dataPoints)<-c("Age(x)","Total(y)")
> rownames(dataPoints)<-c("Adel","Walaa","Rania","Huda","Ahmed","Sameh","Maged","Magdy",
+                         "Shimaa","Hanan","Samy","Farida","Mohamed","Sayed","Eman")
> n <-as.numeric(readline("Enter the number of clusters"))
Enter the number of clusters 2
> result <- kmeans(dataPoints,centers = n)
> result
K-means clustering with 2 clusters of sizes 9, 6

Cluster means:
    Age(x) Total(y)
1 33.66667 817645.4
2 42.00000 892569.2

Clustering vector:
   Adel    Walaa    Rania     Huda    Ahmed    Sameh    Maged    Magdy   Shimaa    Hanan     Samy  Farida Mohamed    Sayed     Eman 
      1        1        1        1        1        1        2        1        2        2        2        1        1        2        2 

Within cluster sum of squares by cluster:
[1] 3529352222 3441348099
 (between_SS / total_SS =  74.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"        
[9] "ifault"      
```

The table displaying name, age, total spending, and the computed cluster number:



	Age(x)	Total(y)
Adel	50	824064
Walaa	29	829587
Rania	37	772871
Huda	39	794570
Ahmed	30	819231
Sameh	35	825147
Maged	60	901010
Magdy	36	831272
Shimaa	55	932250
Hanan	22	893789
Samy	55	869668
Farida	22	841167
Mohamed	25	820900
Sayed	37	857901
Eman	23	900797



Name	Type	Value
result	list [9] (S3: kmeans)	List of length 9
cluster	integer [15]	1 1 1 1 1 1 ...
Adel	integer [1]	1
Walaa	integer [1]	1
Rania	integer [1]	1
Huda	integer [1]	1
Ahmed	integer [1]	1
Sameh	integer [1]	1
Maged	integer [1]	2
Magdy	integer [1]	1
Shimaa	integer [1]	2
Hanan	integer [1]	2
Samy	integer [1]	2
Farida	integer [1]	1
Mohamed	integer [1]	1
Sayed	integer [1]	2
Eman	integer [1]	2
centers	double [2 x 2]	33.7 42.0 817645.4 892569.2
totss	double [1]	27179531517

E. Is to generate association rules between items with minimum support and confidence taken from the user inputs.

The Input will contain then we will use the built-in function in R which is **apriori function**

- First, to **initialize data** we should install package called **gtools** to **calculate permutation** using this code:

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

  also, the user will input the minimum support & minimum confidence so we should use function to read the inputs from the user which is **readline**

  ```
  n <- as.numeric(readline("Enter the min support"))
  m<- as.numeric(readline("Enter the min confidence"))
  ```

  as if the user will enter min support = 0.001 & min confidence = 0.001

- Then we should install package called **arules** it is a special data type for the apriori algorithm

  using this code:

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

  also, to **load data** use this code

  ```
  tdata<-read.transactions("C:/Users/LENOVO/OneDrive/grc.csv", sep=",")
  class(tdata)
  inspect(tdata)
  ```

  output is ‘transactions’

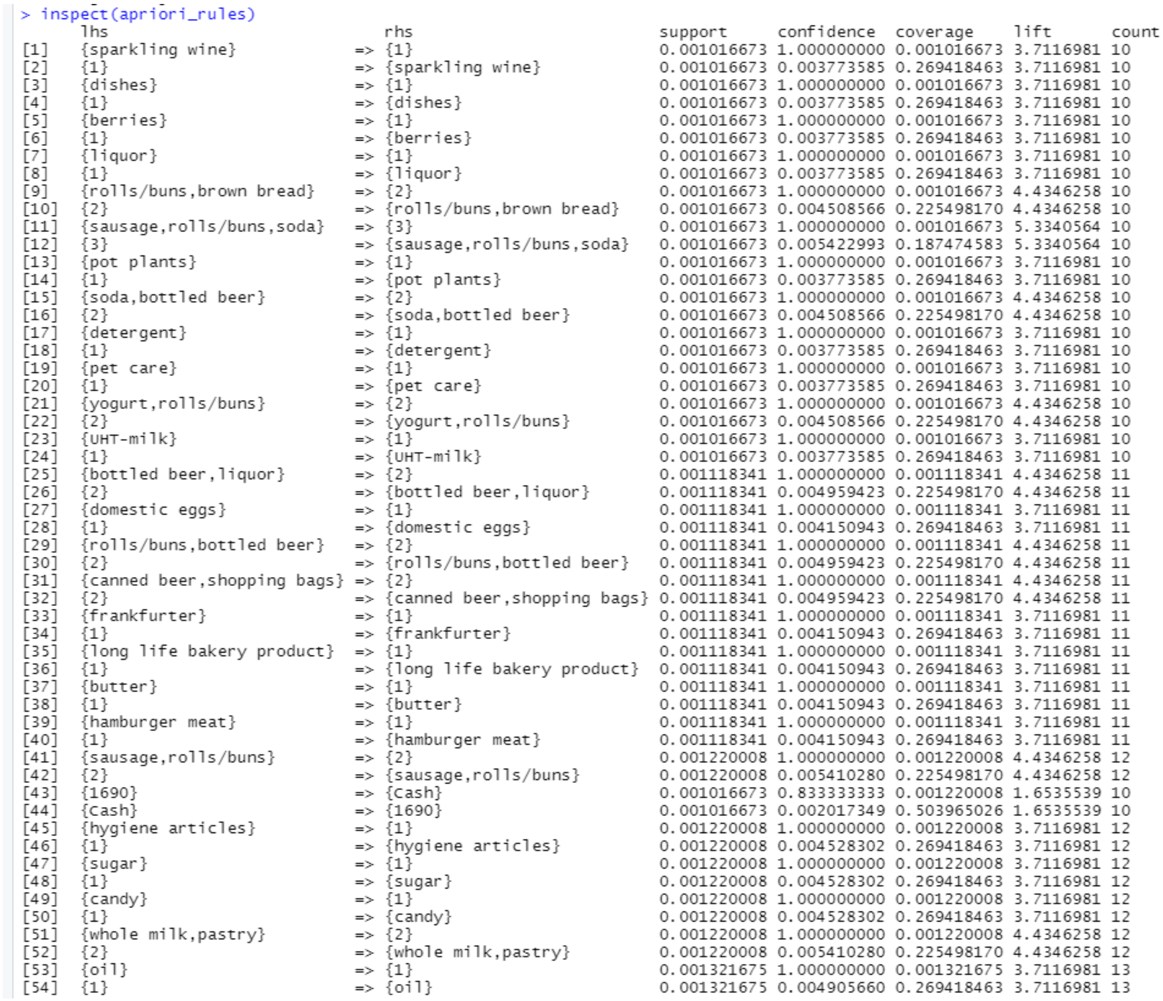
  tdata is a **shortcut of transaction data**

  we use **inspect to show the data**

- To **run the algorithm** use the following code:

  ```
  apriori_rules<-apriori(tdata,
                  parameter=list(supp=min_support, conf=min_confidence ,minlen=2))
  inspect(apriori_rules)
  ```

The Output of this code is:



```
> inspect(apriori_rules)
       lhs                            rhs                         support     confidence  coverage    lift      count
[1]    {sparkling wine}            => {1}                         0.001016673 1.000000000 0.001016673 3.7116981 10
[2]    {1}                         => {sparkling wine}            0.001016673 0.003773585 0.269418463 3.7116981 10
[3]    {dishes}                    => {1}                         0.001016673 1.000000000 0.001016673 3.7116981 10
[4]    {1}                         => {dishes}                    0.001016673 0.003773585 0.269418463 3.7116981 10
[5]    {berries}                   => {1}                         0.001016673 1.000000000 0.001016673 3.7116981 10
[6]    {1}                         => {berries}                   0.001016673 0.003773585 0.269418463 3.7116981 10
[7]    {liquor}                    => {1}                         0.001016673 1.000000000 0.001016673 3.7116981 10
[8]    {1}                         => {liquor}                    0.001016673 0.003773585 0.269418463 3.7116981 10
[9]    {rolls/buns,brown bread}    => {2}                         0.001016673 1.000000000 0.001016673 4.4346258 10
[10]   {2}                         => {rolls/buns,brown bread}    0.001016673 0.004508566 0.225498170 4.4346258 10
[11]   {sausage,rolls/buns,soda}   => {3}                         0.001016673 1.000000000 0.001016673 5.3340564 10
[12]   {3}                         => {sausage,rolls/buns,soda}   0.001016673 0.005422993 0.187474583 5.3340564 10
[13]   {pot plants}                => {1}                         0.001016673 1.000000000 0.001016673 3.7116981 10
[14]   {1}                         => {pot plants}                0.001016673 0.003773585 0.269418463 3.7116981 10
[15]   {soda,bottled beer}         => {2}                         0.001016673 1.000000000 0.001016673 4.4346258 10
[16]   {2}                         => {soda,bottled beer}         0.001016673 0.004508566 0.225498170 4.4346258 10
[17]   {detergent}                 => {1}                         0.001016673 1.000000000 0.001016673 3.7116981 10
[18]   {1}                         => {detergent}                 0.001016673 0.003773585 0.269418463 3.7116981 10
[19]   {pet care}                  => {1}                         0.001016673 1.000000000 0.001016673 3.7116981 10
[20]   {1}                         => {pet care}                  0.001016673 0.003773585 0.269418463 3.7116981 10
[21]   {yogurt,rolls/buns}         => {2}                         0.001016673 1.000000000 0.001016673 4.4346258 10
[22]   {2}                         => {yogurt,rolls/buns}         0.001016673 0.004508566 0.225498170 4.4346258 10
[23]   {UHT-milk}                  => {1}                         0.001016673 1.000000000 0.001016673 3.7116981 10
[24]   {1}                         => {UHT-milk}                  0.001016673 0.003773585 0.269418463 3.7116981 10
[25]   {bottled beer,liquor}       => {2}                         0.001118341 1.000000000 0.001118341 4.4346258 11
[26]   {2}                         => {bottled beer,liquor}       0.001118341 0.004959423 0.225498170 4.4346258 11
[27]   {domestic eggs}             => {1}                         0.001118341 1.000000000 0.001118341 3.7116981 11
[28]   {1}                         => {domestic eggs}             0.001118341 0.004150943 0.269418463 3.7116981 11
[29]   {rolls/buns,bottled beer}   => {2}                         0.001118341 1.000000000 0.001118341 4.4346258 11
[30]   {2}                         => {rolls/buns,bottled beer}   0.001118341 0.004959423 0.225498170 4.4346258 11
[31]   {canned beer,shopping bags} => {2}                         0.001118341 1.000000000 0.001118341 4.4346258 11
[32]   {2}                         => {canned beer,shopping bags} 0.001118341 0.004959423 0.225498170 4.4346258 11
[33]   {frankfurter}               => {1}                         0.001118341 1.000000000 0.001118341 3.7116981 11
[34]   {1}                         => {frankfurter}               0.001118341 0.004150943 0.269418463 3.7116981 11
[35]   {long life bakery product}  => {1}                         0.001118341 1.000000000 0.001118341 3.7116981 11
[36]   {1}                         => {long life bakery product}  0.001118341 0.004150943 0.269418463 3.7116981 11
[37]   {butter}                    => {1}                         0.001118341 1.000000000 0.001118341 3.7116981 11
[38]   {1}                         => {butter}                    0.001118341 0.004150943 0.269418463 3.7116981 11
[39]   {hamburger meat}            => {1}                         0.001118341 1.000000000 0.001118341 3.7116981 11
[40]   {1}                         => {hamburger meat}            0.001118341 0.004150943 0.269418463 3.7116981 11
[41]   {sausage,rolls/buns}        => {2}                         0.001220008 1.000000000 0.001220008 4.4346258 12
[42]   {2}                         => {sausage,rolls/buns}        0.001220008 0.005410280 0.225498170 4.4346258 12
[43]   {1690}                      => {Cash}                      0.001016673 0.833333333 0.001220008 1.6535539 10
[44]   {Cash}                      => {1690}                      0.001016673 0.002017349 0.503965026 1.6535539 10
[45]   {hygiene articles}          => {1}                         0.001220008 1.000000000 0.001220008 3.7116981 12
[46]   {1}                         => {hygiene articles}          0.001220008 0.004528302 0.269418463 3.7116981 12
[47]   {sugar}                     => {1}                         0.001220008 1.000000000 0.001220008 3.7116981 12
[48]   {1}                         => {sugar}                     0.001220008 0.004528302 0.269418463 3.7116981 12
[49]   {candy}                     => {1}                         0.001220008 1.000000000 0.001220008 3.7116981 12
[50]   {1}                         => {candy}                     0.001220008 0.004528302 0.269418463 3.7116981 12
[51]   {whole milk,pastry}         => {2}                         0.001220008 1.000000000 0.001220008 4.4346258 12
[52]   {2}                         => {whole milk,pastry}         0.001220008 0.005410280 0.225498170 4.4346258 12
[53]   {oil}                       => {1}                         0.001321675 1.000000000 0.001321675 3.7116981 13
[54]   {1}                         => {oil}                       0.001321675 0.004905660 0.269418463 3.7116981 13
```

Those are some of the rules because our data is huge

GOALS
SUCCESS
TEAMWORK
VISION
SKILLS