EXPERIMENT : 1

Write R Program using ‘apply’ group of functions to create and apply normalization function on each of the numeric variables/columns of iris dataset to transform them into
I. 0 to 1 range with min-max normalization.
II. a value around 0 with z-score normalization.

AIM:

To write R Program using ‘apply’ group of functions to create and apply normalization function on each of the numeric variables/columns of iris dataset.

DESCRIPTION :

Data Transformation

- Smoothing: remove noise from data
- Aggregation: summarization, data cube construction
- Generalization: concept hierarchy climbing
- Normalization: scaled to fall within a small, specified range
  - min-max normalization
  - z-score normalization
  - normalization by decimal scaling
- Attribute/feature construction
  - New attributes constructed from the given ones
- This step is taken in order to transform the data in appropriate forms suitable for mining process. This involves following ways:
  1. Normalization:
     It is done in order to scale the data values in a specified range (-1.0 to 1.0 or 0.0 to 1.0)
  2. Attribute Selection:
     In this strategy, new attributes are constructed from the given set of attributes to help the mining process.
  3. Discretization:
     This is done to replace the raw values of numeric attribute by interval levels or conceptual levels.
  4. Concept Hierarchy Generation:
     Here attributes are converted from low level to higher level in hierarchy. For Example-The attribute “city” can be converted to “country”.

Data Transformation: Normalization

- Min-max normalization: to [new_min$_A$, new_max$_A$]

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

  - Ex. Let income range \$12,000 to \$98,000 normalized to [0.0, 1.0]. Then \$73,600 is mapped to

$$\frac{73{,}600-12{,}000}{98{,}000-12{,}000}(1.0-0)+0=0.716$$

- Z-score normalization (μ: mean, σ: standard deviation):

$$v'=\frac{v-\mu_A}{\sigma_A} \qquad \frac{73{,}600-54{,}000}{16{,}000}=1.225$$

- Ex. Let μ = 54,000, σ = 16,000. Then
- Normalization by decimal scaling

$$v'=\frac{v}{10^j}$$

Normalizaton:

The most common reason to normalize variables is when you’re conducting some type of multivariate analysis (i.e. you want to understand the relationship between several predictor variables and a response variable) and you want each variable to contribute equally to the analysis.

When variables are measured at different scales, they often do not contribute equally to the analysis. For example, if the values of one variable range from 0 to 100,000 and the values of another variable range from 0 to 100, the variable with the larger range will be given a larger weight in the analysis.

This is common when one variable measures something like salary ($0 to $100,000) and another variable measures something like age (0 to 100 years).

By normalizing the variables, we can be sure that each variable contributes equally to the analysis. Two common ways to normalize (or “scale”) variables include:

I. 0 to 1 range with min-max normalization.

PROGRAM:

```
 min_max=function(x)
 {
(x-min(x))/(max(x)-min(x))
}
r=data.frame(lapply(iris[1:4],min_max))
 r$Species=iris$Species
 head(r)
```

OUTPUT:

```
    Sepal.Length   Sepal.Width   Petal.Length   Petal.Width   Species
 1 0.22222222   0.6250000   0.06779661     0.04166667    setosa
 2 0.16666667   0.4166667   0.06779661      0.04166667   setosa
 3 0.11111111   0.5000000   0.05084746      0.04166667   setosa
 4 0.08333333    0.4583333   0.08474576     0.04166667   setosa
 5 0.19444444    0.6666667   0.06779661     0.04166667    setosa
 6 0.30555556      0.7916667  0.11864407      0.1250000 setosa
```

II.a value around 0 with z-score normalization.
PROGRAM:

```
  z_score=function(x)

 {

(x-mean(x))/sd(x)

}

s=data.frame(lapply(iris[1:4],z_score))

 s$Species=iris$Species

 head(s)
```

Output:

```
     Sepal.Length    Sepal.Width    Petal.Length    Petal.Width   Species

 1   -0.8976739     1.01560199      -1.335752      -1.311052      setosa

  2   -1.1392005    -0.13153881      -1.335752      -1.311052      setosa

  3   -1.3807271     0.32731751     -1.392399      -1.311052       setosa

 4   -1.5014904      0.09788935     -1.279104      -1.311052       setosa

  6   -1.0184372     1.24503015     -1.335752     -1.311052        setosa

  6   -0.5353840     1.93331463     -1.165809     -1.048667         setosa
```

```
r=data.frame(scale(iris[1:4]))

head(r)
```

OUTPUT

```
  Sepal.Length  Sepal.Width  Petal.Length   Petal.Width

1 -0.8976739   1.01560199   -1.335752      -1.311052

2 -1.1392005   -0.13153881   -1.335752     -1.311052

3 -1.3807271   0.32731751   -1.392399      -1.311052

4 -1.5014904   0.09788935   -1.279104      -1.311052

5 -1.0184372    1.24503015   -1.335752     -1.311052

6 -0.5353840   1.93331463    -1.165809      -1.048667
```

EXPERIMENT : 2

Write a R Program to replace specific values in a column of Data Frame in R Programming Language

AIM:

To write a R program to replace specific values in a column of Data Frame in R Programming Language.

DESCRIPTION :

METHOD-1:

Using replace( ) method:

The replace() function, replaces the values in x with indices given in the list by those given in values. If necessary, the values in values are recycled.
Syntax: replace(list , position , replacement_value)

It takes on three parameters first is the list name, then the index at which the element needs to be replaced, and the third parameter is the replacement values.

METHOD-2:

Using the logical condition:

Now let us see how we can replace values of specific values in the column using logical conditions. First, let us create the data frame in R

Syntax: dataframe_name$column_name1[dataframe_name$column_name2==y]<-x

Parameters:

y:It is the value which help us to fetch the data location the column

x: It is the value which needs to be replaced

PROGRAM :

```
rollno=c(20,19,NA,18)
id=c(113,148,140,120)
marks=c(76,85,54,NA)
r=data.frame(rollno,id,marks)r
```

OUTPUT:

```
     rollno  id    marks
  1    20   113    76
  2    19   148    85
  3    NA   140    54
  4    18   120    NA
```

```
View(r)
a=r
  a
```

OUTPUT

```
   rollno  id marks
 1     20 113    76
 2     19 148    85
 3     NA 140    54
 4     18 120    NA
```

```
s=replace(rollno,2,88)
  s
[1] 20 88 NA 18
```

1.Replacing NA's WITH 0's

```
r[is.na(r)]=0
  r
   rollno   id   marks
 1     20  113    76
 2     19  148    85
 3      0  140    54
 4     18  120     0
```

```
a$rollno[is.na(a$rollno)]=mean(a$rollno,na.rm=T)
  a
   rollno  id marks
 1     20 113    76
 2     19 148    85
 3     19 140    54
 4     18 120    NA
```

EXPERIMENT : 3

Implement decision trees using 'Reading Skills' dataset

AIM :

To implement decision trees using 'Reading Skills' dataset.

DESCRIPTION :

A decision tree is a tool that builds regression models in the shape of a tree structure. Decision trees take the shape of a graph that illustrates possible outcomes of different decisions based on a variety of parameters. Decision trees break the data down into smaller and smaller subsets, they are typically used for machine learning and data mining, and are based on machine learning algorithms. Decision trees are also referred to as recursive partitioning.

The Algorithm: How decision trees work

- Decision trees are based on an algorithm called ID3 created by JR Quinlan
- ID3 employs entropy and information gain to create a decsion tree
- entropy:
  is a top-down process that partitons data into subsets that consist of homogeneous data points. If a sample is completely homogenous the entropy is zero, if the sample is completely divided entropy is one.
- information gain:
  the decrease in entropy after the dataset is split on an attribute/parameter. Decision trees make splits based on which attributes generate the highest information gain, which results in the most homogenous subsets. Entropy values are calculated for every parameter that is entered into the tree model, for each decision, the parameter with the highest information gain is selected. Then the process is repeated.

Decision Tree Components:

Decision trees are made up to two parts: nodes and leaves.

- Nodes:
  represent a decision test, examine a single variable and move to another node based on the outcome
- Leaves:
  represent the outcome of the decision.

Decision trees are useful to make various predictions. For example, to predict if an email is SPAM or not, to predict health outcomes, to predict what group an individual belongs to based on a variety of factors that are specified in the decision tree model.

ADVANTAGES:

- simple to understand and interpret
- help determine the expected outcomes of various scenarios
- help determine best and worst values for different scenarios
- can be combined with other decision techniques
- require a relatively low degree of data preparation
- can accommodate missing data

- low sensitivity to outliers
- low impact of nonlinear relationships between parameters
- can handle both categorical and numeric variables
- can translate the decision tree results into “decision rules”

DISADVANTAGES:

- for categorical variables, more levels of the variable creates more bias of the decision tree toward that variable
- if the tree is over-fitted to the data, the results can be poor predictors

READING SKILLS DATASET:

Description:
A toy data set illustrating the spurious correlation between reading skills and shoe size in school-children.

Usage:
data("readingSkills")

Format:
A data frame with 200 observations on the following 4 variables.

nativeSpeaker	:a factor with levels no and yes, where yes indicates that the child is a native speaker of the language of the reading test.
Age	: age of the child in years.
shoeSize	: shoe size of the child in cm.
Score	: raw score on the reading test.

The package we will use to create decision trees is called 'party'.
To install the package

install.packages("party")

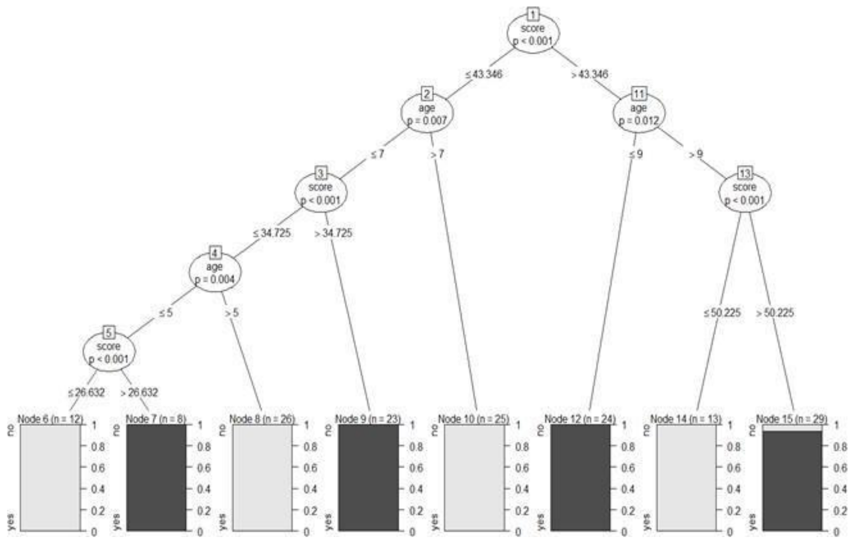
ctree: Conditional Inference Trees:

The package "party" has the function ctree() which is used to create and analyze decision tree.
Syntax The basic syntax for creating a decision tree in R is −
ctree(formula, data)
formula is a formula describing the predictor and response variables. •
data is the name of the data set used.

PROGRAM:

```
install.packages("party")
library(party)
head(readingSkills)
s=sample(1:200,140)
train=readingSkills[s,]
test=readingSkills[-s,]
tree=ctree(nativeSpeaker~age+shoeSize+score,data=train)plot(tree)
pred=predict(tree,test)table(pred)
acc=table(pred,test$nativeSpeaker)acc
ac_test=sum(diag(acc))/sum(acc) ac_test
```

PLOT TREE:



OUTPUT:

```
pred
no
yes26
34
acc

Pred no yes
No 21 5
yes 6 28

 [1] 0.8166667
```

EXPERIMENT : 4

Implement association rule mining on market basket data

AIM:

To implement association rule mining on market basket data.

DESCRIPTION:

Market Basket Analysis is one of the key techniques used to uncover links between items by large retailers. It works by searching for combinations of items that often happen in transactions together. In a different way, retailers can identify relations among the items they buy.

It is a method of identifying object associations that "go together" in a commercial context. Market basket analysis, in actuality, goes beyond the supermarket scenario from which it gets its name. The investigation of any collection of commodities to uncover affinities that may be exploited in some way is known as market basket analysis. In big transactional or relational data sets, frequent itemset mining leads to the finding of relationships and correlations between items. With vast volumes of data being collected and stored on a regular basis, many companies are interested in extracting patterns from their databases. Many commercial decision-making processes, such as catalogue design, cross-marketing, and customer buying behaviour research, might benefit from the finding of interesting correlation patterns across massive amounts of business transaction records.

Association rules are widely used to analyse retail basket or transaction data, with the objective of establishing strong rules based on a strong transaction information rules concept.

Market Basket Analysis uses the information to:

- Capable of recognizing customer purchasing patterns
- To identify who customers are(not by name)
- Understand why you buy certain items

Learn about your goods (products):

- Slow movements fast and slow
- Products that are jointly purchased
- Products that could be promoted

Association Rules in Market Basket Analysis:
The similarities between items are many ways to see. These techniques fall within the framework of the general association.
the end result of this method is a set of rules that can be interpreted as "if that is so."

PROGRAM:

```
install.packages('arules')
library(arules)
install.packages('arulesViz')
library(arulesViz)
data("Groceries")
summary(Groceries)
inspect(head(Groceries))
rule1=apriori(Groceries,parameter=list(supp=0.001,conf=0.8))
inspect(head(rule1)) rule2=sort(rule1,by="confidence",decreasing=TRUE)
inspect(head(rule2))
rule3=apriori(Groceries,parameter=list(supp=0.001,conf=0.8,minlen=4))
inspect(head(rule3))
rule4=apriori(Groceries,parameter=list(supp=0.001,conf=0.08),appearance =
list(default="lhs",rhs="whole milk"),control=list(verbose=F))
inspect(head(rule4))
rule5=apriori(data=Groceries,parameter=list(supp=0.001,conf=0.08),appearance=list(def
ault="rhs",lhs="whole milk"),control=list(verbose=F))
inspect(head(rule5))) itemFrequencyPlot(Groceries,topN=20,type="relative",main='Relative
Item FrequencyPlot',ylab="Item Frequency Relative)")
```

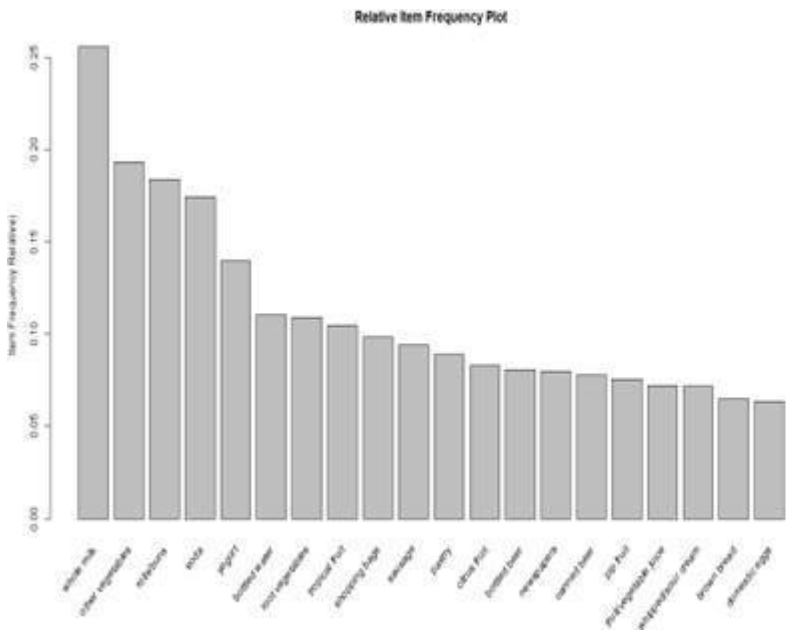
OUTPUT:

```
summary(Groceries)
transactions as itemMatrix in sparse format with
 9835 rows (elements/itemsets/transactions) and
 169 columns (items) and a density of 0.02609146

most frequent items:
      whole milk other vegetables       rolls/buns             soda
yogurt          (Other)
            2513             1903             1809             1715
1372            34055

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12   13   14
15   16   17   18   19   20   21   22   23
2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77
55   46   29   14   14    9   11    4    6
  24   26   27   28   29   32
   1    1    1    1    3    1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   3.000   4.409   6.000  32.000

includes extended item information - examples:
       labels  level2           level1
1 frankfurter sausage meat and sausage
2     sausage sausage meat and sausage
3  liver loaf sausage meat and sausage
> inspect(head(Groceries))
    items
[1] {citrus fruit, semi-finished bread, margarine, ready soups}
[2] {tropical fruit, yogurt, coffee}
[3] {whole milk}
[4] {pip fruit, yogurt, cream cheese , meat spreads}
[5] {other vegetables, whole milk, condensed milk, long life bakery pro
duct}
[6] {whole milk, butter, yogurt, rice, abrasive cleaner}
> rule1=apriori(Groceries,parameter=list(supp=0.001,conf=0.8))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minl
en maxlen target  ext
        0.8    0.1    1 none FALSE            TRUE       5   0.001
1     10  rules TRUE
Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 9
```

```
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [157 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].


checking subsets of size 1 2 3 4 5 6 done [0.01s].
writing ... [410 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
> inspect(head(rule1))
    lhs                                 rhs              support     con
fidence coverage    lift      count
[1] {liquor, red/blush wine}         => {bottled beer} 0.001931876 0.9
047619  0.002135231 11.235269 19
[2] {curd, cereals}                  => {whole milk}   0.001016777 0.9
090909  0.001118454  3.557863 10
[3] {yogurt, cereals}                => {whole milk}   0.001728521 0.8
095238  0.002135231  3.168192 17
[4] {butter, jam}                    => {whole milk}   0.001016777 0.8
333333  0.001220132  3.261374 10
[5] {soups, bottled beer}            => {whole milk}   0.001118454 0.9
166667  0.001220132  3.587512 11
[6] {napkins, house keeping products} => {whole milk}   0.001321810 0.8
125000  0.001626843  3.179840 13

rule2=sort(rule1,by="confidence",decreasing=TRUE)
> inspect(head(rule2))
    lhs                                              rhs
support     confidence coverage    lift
[1] {rice, sugar}                                 => {whole milk}
0.001220132 1          0.001220132 3.913649
[2] {canned fish, hygiene articles}               => {whole milk}
0.001118454 1          0.001118454 3.913649
[3] {root vegetables, butter, rice}               => {whole milk}
0.001016777 1          0.001016777 3.913649
[4] {root vegetables, whipped/sour cream, flour} => {whole milk}
0.001728521 1          0.001728521 3.913649
[5] {butter, soft cheese, domestic eggs}          => {whole milk}
0.001016777 1          0.001016777 3.913649
[6] {citrus fruit, root vegetables, soft cheese} => {other vegetables}
0.001016777 1          0.001016777 5.168156
    count
[1]  12
[2]  11
[3]  10
[4]  17
[5]  10
[6]  10
rule3=apriori(Groceries,parameter=list(supp=0.001,conf=0.8,minlen=4))
Aprior
```

```
Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minl
en maxlen target  ext
        0.8    0.1    1 none FALSE            TRUE       5   0.001
4     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 9

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [157 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 6 done [0.01s].
writing ... [381 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
> inspect(head(rule3))
    lhs                                                 rhs
support     confidence coverage    lift
[1] {other vegetables, yogurt, specialty cheese} => {whole milk}
0.001321810 0.8125000  0.001626843 3.179840
[2] {turkey, tropical fruit, root vegetables}    => {other vegetables}
0.001220132 0.8000000  0.001525165 4.134524
[3] {turkey, root vegetables, whole milk}        => {other vegetables}
0.001220132 0.8000000  0.001525165 4.134524
[4] {root vegetables, butter, rice}              => {whole milk}
0.001016777 1.0000000  0.001016777 3.913649
[5] {tropical fruit, other vegetables, rice}     => {whole milk}
0.001016777 0.8333333  0.001220132 3.261374
[6] {root vegetables, yogurt, rice}              => {other vegetables}
0.001423488 0.8750000  0.001626843 4.522136
    count
[1] 13
[2] 12
[3] 12
[4] 10
[5] 10
[6] 14
rule4=apriori(Groceries,parameter=list(supp=0.001,conf=0.08),appearance
= list(default="lhs",rhs="whole milk"),control=list(verbose=F))
> inspect(head(rule4))
    lhs                       rhs           support     confidence  coverage
lift     count
[1] {}                     => {whole milk} 0.255516014 0.2555160  1.000000
000 1.000000  2513
[2] {honey}                => {whole milk} 0.001118454 0.7333333  0.001525
165 2.870009   11
```

```
[3] {soap}               => {whole milk} 0.001118454 0.4230769  0.002643
620 1.655775   11
[4] {cocoa drinks}       => {whole milk} 0.001321810 0.5909091  0.002236
909 2.312611   13
[5] {pudding powder}     => {whole milk} 0.001321810 0.5652174  0.002338
587 2.212062   13
[6] {cooking chocolate} => {whole milk} 0.001321810 0.5200000  0.002541
942 2.035097   13

rule5=apriori(data=Groceries,parameter=list(supp=0.001,conf=0.08),appea
rance=list(default="rhs",lhs="whole   milk"),control=list(verbose=F))
> inspect(head(rule5))
    lhs    rhs              support    confidence coverage lift count
[1] {}  => {bottled beer}  0.08052872 0.08052872 1        1     792
[2] {}  => {pastry}        0.08896797 0.08896797 1        1     875
[3] {}  => {citrus fruit}  0.08276563 0.08276563 1        1     814
[4] {}  => {shopping bags} 0.09852567 0.09852567 1        1     969
[5] {}  => {sausage}       0.09395018 0.09395018 1        1     924
[6] {}  => {bottled water} 0.11052364 0.11052364 1        1    1087
```



EXPERIMENT : 5

Implement k-means clustering using R

AIM:

To implement k-means clustering using R

DESCRIPTION:

An Unsupervised Non-linear algorithm that cluster data based on similarity or similar groups. It seeks to partition the observations into a pre-specified number of clusters. Segmentation of data takes place to assign each training example to a segment called a cluster. In the unsupervised algorithm, high reliance on raw data is given with large expenditure on manual review for review of relevance is given. It is used in a variety of fields like Banking, healthcare, retail, Media, etc.

K-Means clustering groups the data on similar groups. The algorithm is as follows:

1. Choose the number K clusters.
2. Select at random K points, the centroids (Not necessarily from the given data).
3. Assign each data point to closest centroid that forms K clusters.
4. Compute and place the new centroid of each centroid.
5. Reassign each data point to new cluster.

After final reassignment, name the cluster as Final cluster.

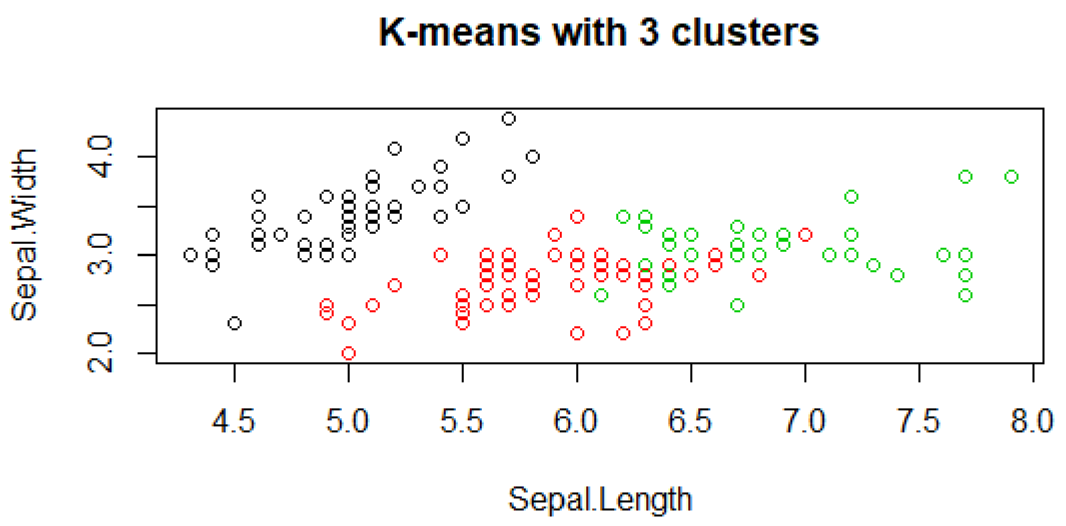
PROGRAM:

```
 install.packages("ClusterR")
 library(ClusterR)
 install.packages("cluster")
 library(cluster)
 iris_1=iris[, -5]
 set.seed(240)
 r= kmeans(iris_1, centers = 3, nstart = 20)
 r
 r$cluster
 r$centers
 r$size
 cm =table(iris$Species, r$cluster)
 cm
 plot(iris_1[c("Sepal.Length", "Sepal.Width")],col = r$cluster,main = "K-means with 3 clusters")
 r$centers
 r$centers[, c("Sepal.Length", "Sepal.Width")]
 points(r$centers[, c("Sepal.Length", "Sepal.Width")],col = 1:3,pch = 8, cex = 3)
 y_kmeans=r$cluster
 y_kmeans
clusplot(iris_1[, c("Sepal.Length", "Sepal.Width")],y_kmeans,lines = 0,shade = TRUE,
color = TRUE,labels =            2,plotchar = FALSE,span = TRUE,main = paste("Cluster iris"),xlab =
‘sepal.Length',ylab = 'Sepal.Width')
r
```

OUTPUT:

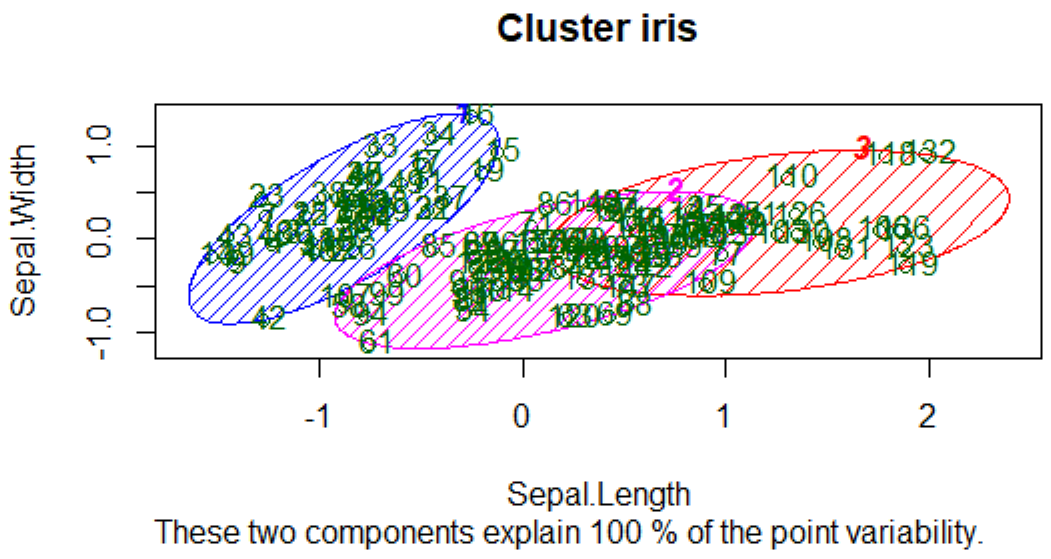
```
K-means clustering with 3 clusters of sizes 50, 62, 38
Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053
Clustering vector:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [48] 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [95] 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3
[142] 3 2 3 3 3 2 3 3 2

Within cluster sum of squares by cluster:
[1] 15.15100 39.82097 23.87947
 (between_SS / total_SS =  88.4 %)
Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
r$cluster
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [48] 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [95] 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3
[142] 3 2 3 3 3 2 3 3 2
```

```
> r$centers
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053
> r$size
[1] 50 62 38
> cm =table(iris$Species, r$cluster)
> cm
1  2  3
  setosa      50 0  0
  versicolor  0 48  2
  virginica   0 14 36
> plot(iris_1[c("Sepal.Length", "Sepal.Width")],col = r$cluster,main = "K-means with 3 clusters")
```



```
> r$centers
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053
> r$centers[, c("Sepal.Length", "Sepal.Width")]
Sepal.Length Sepal.Width
```

```
1     5.006000    3.428000
2     5.901613    2.748387
3     6.850000    3.073684
> points(r$centers[, c("Sepal.Length", "Sepal.Width")],col = 1:3,pch = 8, cex = 3)
> y_kmeans=r$cluster
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [48] 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [95] 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3
 [142] 3 2 3 3 3 2 3 3 3
> clusplot(iris_1[, c("Sepal.Length", "Sepal.Width")],y_kmeans,lines = 0,shade = TRUE,color =
TRUE,labels = 2,plotchar = FALSE,span = TRUE,main = paste("Cluster iris"),xlab =
'Sepal.Length',ylab = 'Sepal.Width')
```


Cluster iris
Sepal.Width
Sepal.Length
These two components explain 100 % of the point variability.