Plots and Charts

Algorithm:

- Assign the values for the x axis
- Assign the values for the y axis
- Generate the plot using the corresponding plotting function based on the plot type:
- For "dotplot": Use dotplot() function.
- For "histogram": Use hist() function.
- For "barchart": Use barplot() function.

Code:

3D pie:

```
install.packages("plotrix")

library(plotrix)

x <- c(9, 6, 10, 9)

mylabel <- c("R", "S", "A", "J")

colors <- c("blue", "yellow", "green", "black")

pie3D(x, labels = mylabel, main = "DATA", col = colors)
```

Scatter Plot:

```
x=c(1,2,3,4,5)

y=c(2,3,5,4,6)

plot(x,y,main="ScatterPlot",xlab="x values",ylab="y values",col='blue',pch=19)
```

Box Plot:

```
x=c(10,12,21,18,5,7,4,3,9,11)

boxplot(x,horizontal=TRUE,main="BoxPlot",xlab="Values",col='skyblue',border="black")
```

Histogram:

```
x=c(10,12,21,18,5,7,4,3,9,11)

boxplot(x,horizontal=TRUE,main="BoxPlot",xlab="Values",col='skyblue',border="black")
```

Dot Plot:

```
x <- c("A", "B", "C", "D", "E")

y <- c(10, 15, 8, 12, 20)

plot(y, type = "o", xlab = "Values", ylab = "Categories", main = "Dotplot")

points(y, 1:length(y), pch = 19, col = "blue")

text(y, 1:length(y), labels = x, pos = 4, col = 'red')
```

Using Loops

For Loop:

It is a type of control statement that enters one to one easily construct on r loop that has to run statements or a set of statements multiple times

Syntax:

For(value in sequence)

{

Statement

}

Code:

```
week=c('Sunday',"Monday",'Tuesday','Wednesday','Thursday','Friday','Saturday')

for(day in week)

{

  print(day)

  }
```

While Loop:

It is a type of control statement that will run the statement or a set of statements repeatedly under the given condition becomes false.

Syntax:

While(condtion)

{

Statement

}

Code:

```
n=5
factorial=1
i=1
while(i<=n)
{
 factorial=factorial*i
 i=i+1
}
print(factorial)
```

Repeat Loop:

It is a simple loop that will run the same statement or a group of statements repeatedly until the stop condition has been encountered.

Syntax:

```
repeat
{
Statement
If(condition)
{
Break
}
}
```

Code:

```
i <- 0
repeat {
 print('data science')
 i <- i + 1
 if (i == 5) {
  break
```

```
  }
}
```

Break statement:

The break statement Is used to terminate the loop at a particular iteration

Code:

```
for (val in 1:5) {
  if (val == 3) {
    break
  }
  print(val)
}
```

Creating custom function in R

Algorithm:

- Initialize variable name to the function
- User defined functions are created using function()

Function() syntax:

```
func.name=function(arg1,arg2,arg3……){
Function body
}
```

Code:

#create a function

```
new.func=function(){
  for(i in 1:5){
    print(i^2)
  }
}
new.func()


#create a function with arguments
sec.func=function(x,y,z){
  result=x*y+z
  print(result)
}
sec.func(2,3,4)
```

Using list in R

Algorithm:

- Initialize a variable list_1 and list() function creates a list
- The list elements can be grouped using names() function
- Elements are accessed using the indices
- Removal of element is done by initializing the index to NULL
- Unlist() function is used to convert list to vectors

Code:

```
list1 <- list(1, 2, 3)

list2 <- list("Jason", "Antony", "Vishal")

list3 <- list(c(1, 2, 3))

list4 <- list(TRUE, FALSE, TRUE)

list1

list2

list3

list4


list_data <- list(
  students = c("Jason", "Gill", "Harry"),
  marks = matrix(c(40, 80, 60, 50, 45, 90), nrow = 2),
  course = list("BCA", "MCA", "B.tech")
)
#giving names to elements in the list
names(list_data)=c("students","marks","course")
print(list_data)
#accessing the first element in the list
print(list_data[1])
#accessing the third element
print(list_data[3])
#accessing the first element of the list
```

```
print(list_data["student"])
print(list_data$student)
print(list_data)
#adding element at the end of the list
list_data[4]="Moradabad"
print(list_data[4])
#removing the last element
list_data[4]=NULL
print(list_data[4])
#updating the 3rd element
list_data[3]="R programming"
print(list_data[3])
list_1=list(10:20)
print(list_1)
list_2=list(5:4)
print(list_2)
#converting the list to vectors
v1=unlist(list_1)
v2=unlist(list_2)
print(v1)
print(v2)
#adding the vectors
result=v1+v2
print(result)
#merging two lists
elist=list(2,4,6,8,10)
olist=list(1,3,5,7,9)
merged.list=list(elist,olist)
print(merged.list)
```

Using Data Frames

Algorithm:

- Initialize the variables df and using functions data.frame() input the values as a list.
- By using c() input the list of values into the frame
- Using the variable df, extract the data and print it
- Use rbind() to add a row in the data frame
- Use cbind() to add a column in the data frame
- The inputs for rbind() and cbind() are given as a list
- Initialize the column to NULL to delete the column from the data frame
- Using print() we can print the output

Code:

```
# Creating a data frame

df <- data.frame(

  Name = c("John", "Alice", "Bob", "Emily"),

  Age = c(25, 30, 22, 28),

  Grade = c("A", "B", "C", "A"),

  Gender= c("M","M","M","F")

)

print(df)  # Printing the data frame


print(df$Name)  # Extracting specific columns


print(df[1, ])  # Extracting the first row


print(tail(df, 2))  # Extracting the last two rows


print(df[c(2, 3), c(1, 4)])  # Extracting second and third row corresponding to the first and fourth column


new_row <- data.frame(Name = "Mark", Age = 35, Grade = "B",Gender="M")

df <- rbind(df, new_row)

print(df)   # Adding a row
```

```
df$City <- c("New York", "Los Angeles", "Chicago", "Boston", "San Francisco")
print(df)  # Adding a column

df$City <- NULL
print(df)  # Deleting a column

summary(df)  # Printing the summary of the data frame
```

Vectors and Matrices

Algorithm:

- Initialize the variables and use the matrix() function to create a matrix of nrow and ncol.
- Row and column names are to be stored as a list in a variable and assigned using dimnames() function
- Matrix[] function retrieves the data from the matrix
- Initialize a values to a certain coordinate of the matrix to replace the element
- Rbind() and cbind() are used to add rows and columns respectively in the matrices
- Mathematical operations (+,-,*,/) are used to perform mathematical operations among the matrix

Code:

```
p <- matrix(c(5:16), nrow = 4, ncol = 3, byrow = TRUE)

print(p)


q <- matrix(c(3:14), nrow = 4, ncol = 3, byrow = FALSE)

print(q)


# Assigning row names and column names

rownames <- c("row1", "row2", "row3", "row4")

columnnames <- c("col1", "col2", "col3")

dimnames(p) <- list(rownames, columnnames)

print(p)


# Accessing the element from matrix

p[1, 3]

q[1, ]

p[3]


# Assigning a single element

p[2, 2] <- 20

print(p)
```

```
# Modifying multiple elements
q[q >= 11] <- 6
print(q)

# Adding rows
p <- rbind(p, c(20, 21, 22))
print(p)

# Adding columns
q <- cbind(q, c(17, 18, 19, 20))
print(q)

# Matrix operations
r <- matrix(c(1:16), nrow = 4, ncol = 4)
print(r)

# Addition
a <- r + q
print(a)

# Subtraction
b <- r - q
print(b)

# Multiplication
c <- r * q
print(c)
```

```
# Division
d <- r / q
print(d)

# Multiplication by constant
e <- r * 5
print(e)
```

Built-in functions

Code:

```
#math functions
x=-2
print(abs(x))
x=2
print(sqrt(x))
x=2.8
print(ceiling(x))
print(floor(x))
x=c(2.2,6.5,10.11)
print(trunc(x))
x=3.48579076
print(round(x,digits=3))
x=2
print(sin(x))
print(cos(x))
print(tan(x))
print(log(x))
print(exp(x))
print(log10(x))
#character functions
s="R PROGRAMMING LAB"
print(tolower(s))
s="r programming lab"
print(toupper(s))
s <- "r programming lab"
print(strsplit(s,""))   #str split(x,split)
paste("str",1:3,sep="")   #paste(--,sep=**)
```

```
st="you are a student of sathyabama university"
sub("university","college",st)   #sub(pattern,replace,xignore.case=FALSE,fixed=FALSE)
st=c('sa',"un","student")
pattern='un'
print(grep(pattern,st)) #grep(pattern x,ignore.case=FALSE,fixed=FALSE)
st <- "r programming"
substr(st, 1, 5)
substr(st, 4, 10)   #sub(x,start=n,stop=n2)
#statistical functions
x=c(2,3,4,5)
mean(x,trim=0,nrm=FALSE) #mean(x,trim=0,nrm=FALSE)
print(sd(x))
print(median(x))
print(range(x))
print(sum(x))
print(diff(x,log=1))
print(min(x))
print(max(x))
x=matrix(1:15,nrow=3,ncol=5,byrow=TRUE)
print(scale(x))  #scale(x,center=TRUE,scale=TRUE)
```

Implementing Machine Learning algorithm in R

Algorithm:

- Import and install party library
- Use ctree() to initialize the decision tree
- Use predict() for predicting the output
- Use addmargins() for creating a confusion matrix

Code:

```
library(party)

data(readingSkills)

d <- readingSkills

sam <- sample(1:150, 104)

train <- d[sam, ]

test <- d[-sam, ]

tree1 <- ctree(nativeSpeaker ~ age + shoeSize + score, data = train)

plot(tree1)

pred <- predict(tree1, train)

conf_matrix <- table(pred, train$nativeSpeaker)

acc <- addmargins(conf_matrix)

acc

accuracy <- sum(diag(prop.table(conf_matrix)))

print(paste("Accuracy:", accuracy))
```

Support Vector Machine

Algorithm:

- Import e1701 library
- Load sample dataset
- Divide dataset into train and test data
- Implement svm model using svm()
- Print the output

Code:

```
library(e1071)

iris1 <- iris

sam <- sample(1:150, 104)

train <- iris1[sam, ]

test <- iris1[-sam, ]

model <- svm(Species ~ ., train)

print(model)

datasets::npk
```

Fitting statistical model

Algorithm:

- Initialize the x and y using function()
- Use lm() function for linear regression
- Use summary() to summarize
- Use print() to display
- Use plot() to plot the graph

Code:

```
# Sample data

x <- 1:10

y <- 2*x + rnorm(10, mean = 0, sd = 1)  # Simulated linear relationship with noise

# Create a data frame

data <- data.frame(x = x, y = y)

# Fit a linear regression model

model <- lm(y ~ x, data = data)

# Summary of the model

summary(model)

# Plot the data and regression line

plot(y ~ x, data = data, main = "Linear Regression", xlab = "x", ylab = "y")

abline(model, col = "red")
```