1. Explore Basic Data Structure in R.

In R, there are several basic data structures that are commonly used for storing and organizing data. These data structures include vectors, matrices, arrays, lists, and data frames. Data structures in R programming are tools for holding multiple values. The idea is to reduce the space and time complexities of different tasks.

R's base data structures are often organized by their dimensionality (1D, 2D, or nD) and whether they're homogeneous (all elements must be of the identical type) or heterogeneous (the elements are often of various types). This gives rise to the six data types which are most frequently utilized in data analysis.

The most essential Data Structures used in R are:

1.Vectors: A vector is an ordered collection of basic data types of a given length. The only key thing here is all the elements of a vector must be of the identical data type e.g homogeneous data structures. Vectors are one-dimensional data structures. They can be created using the `c()` function.

CODE:

```
# R program to illustrate Vector

# Vectors(ordered collection of same data type)
X = c(1, 3, 5, 7, 8)

# Printing those elements in console
print(X)
```

```
Output

Rscript /tmp/QCfEAGbZuc.r
[1] 1 3 5 7 8
```

2.Lists: A list is a generic object consisting of an ordered collection of objects. Lists are heterogeneous data structures. These are also one-dimensional data structures. A list can be a list of vectors, list of matrices, a list of characters and a list of functions and so on. They can be created using the `list()` function.

CODE:

```
# R program to illustrate a List

# The first attributes is a numeric vector containing the employee IDs which is created using the 'c'
command here
empId = c(1, 2, 3, 4)

# The second attribute is the employee name which is created using this line of code here which is
the character vector
empName = c("Debi", "Sandeep", "Subham", "Shiba")

# The third attribute is the number of employees which is a single numeric variable.
numberOfEmp = 4

# We can combine all these three different data types into a list containing the details of employees
which can be done using a list command
empList = list(empId, empName, numberOfEmp)
print(empList)
```

```
Output
Rscript /tmp/QCfEAGbZuc.r
[[1]]
[1] 1 2 3 4

[[2]]
[1]"Debi"    "Sandeep" "Subham"  "Shiba"

[[3]]
[1] 4
```

3.Data frames: Data frames are generic data objects of R which are used to store the tabular data. Dataframes are the foremost popular data objects in R programming because we are comfortable in seeing the data within the tabular form. They are two-dimensional, heterogeneous data structures. These are lists of vectors of equal lengths.
They can be created using the `data.frame()` function.

CODE:

```
# R program to illustrate dataframe

# A vector which is a character vector
Name = c("Amiya", "Raj", "Asish")

# A vector which is a character vector
Language = c("R", "Python", "Java")

# A vector which is a numeric vector
Age = c(22, 25, 45)

# To create dataframe use data.frame command and then pass each of the vectors we have created as
arguments to the function data.frame()
df = data.frame(Name, Language, Age)

print(df)
```

```
Output
Rscript /tmp/QCfEAGbZuc.r
Name Language Age
1 Amiya        R  22
2   Raj   Python  25
3 Asish     Java  45
```

4.Matrices: A matrix is a rectangular arrangement of numbers in rows and columns. In a matrix, as we know rows are the ones that run horizontally and columns are the ones that run vertically. Matrices are two-dimensional, homogeneous data structures. They can be created using the `matrix()` function.

CODE:

```
# R program to illustrate a matrix

A = matrix(
    # Taking sequence of elements
    c(1, 2, 3, 4, 5, 6, 7, 8, 9),

    # No of rows and columns
    nrow = 3, ncol = 3,

    # By default matrices are  in column-wise order So this parameter decides how to arrange the
matrix
    byrow = TRUE
)

print(A)
```

```
Output

Rscript /tmp/QCfEAGbZuc.r
[,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

5.Arrays: Arrays are the R data objects which store the data in more than two dimensions. Arrays are n-dimensional data structures. For example, if we create an array of dimensions (2, 3, 3) then it creates 3 rectangular matrices each with 2 rows and 3 columns. They are homogeneous data structures. They can be created using the `array()` function.

CODE:

```
# R program to illustrate an array

A = array(
    # Taking sequence of elements
    c(1, 2, 3, 4, 5, 6, 7, 8),

    # Creating two rectangular matrices each with two rows and two columns
    dim = c(2, 2, 2)
)
print(A)
```

```
Output

Rscript /tmp/QCfEAGbZuc.r
, , 1

     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

     [,1][,2]
[1,]    5    7
[2,]    6    8
```

6.Factors: Factors are the data objects which are used to categorize the data and store it as levels. They are useful for storing categorical data. They can store both strings and integers. They are useful to categorize unique values in columns like “TRUE” or “FALSE”, or “MALE” or “FEMALE”, etc.. They are useful in data analysis for statistical modeling.

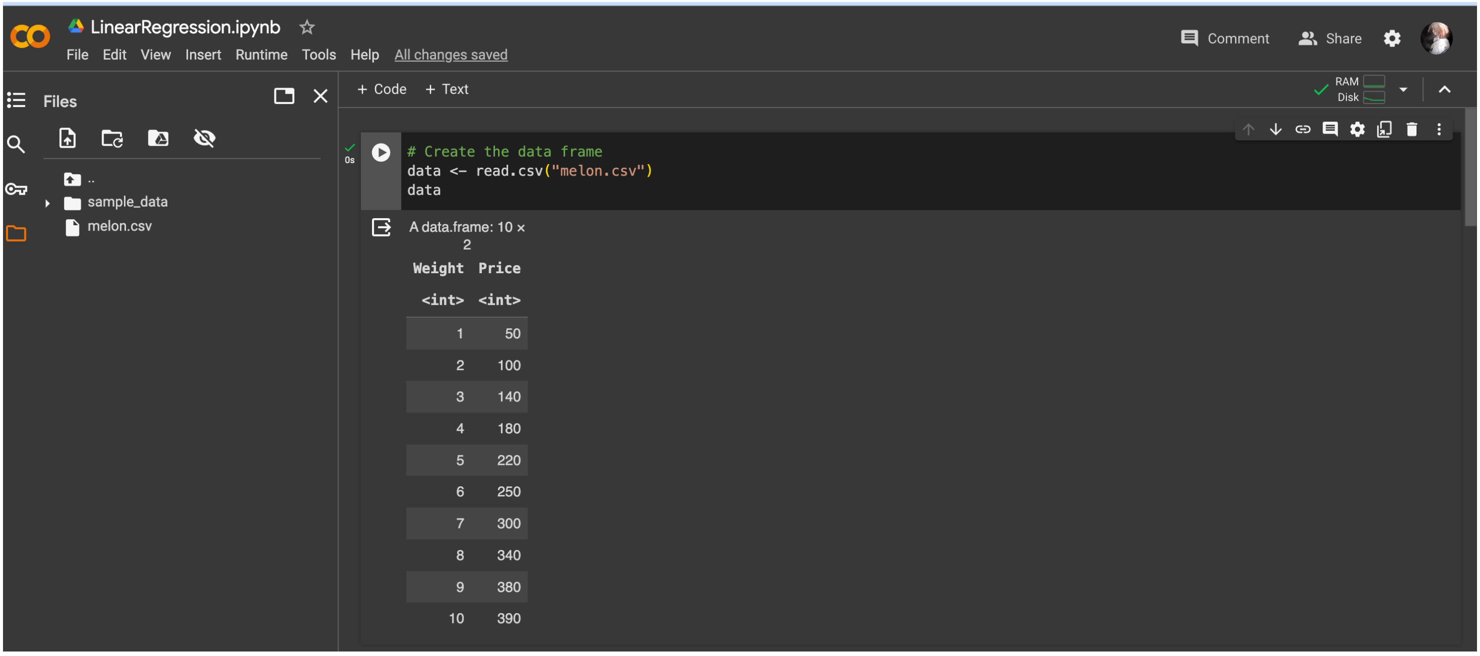
CODE:

```
# R program to illustrate factors

# Creating factor using factor()
fac = factor(c("Male", "Female", "Male",
            "Male", "Female", "Male", "Female"))

print(fac)
```

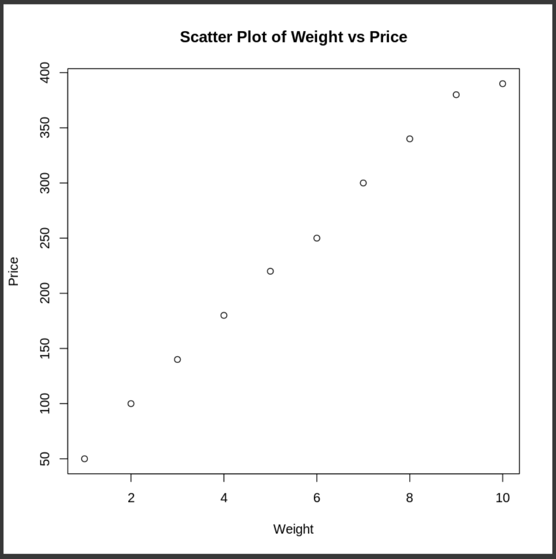
Output

```
Rscript /tmp/QCfFEAGbZuc.r
[1] Male   Female Male   Male   Female Male   Female
Levels: Female Male
```

Q2) **Implement Linear Regression in R and Visualize the results.**



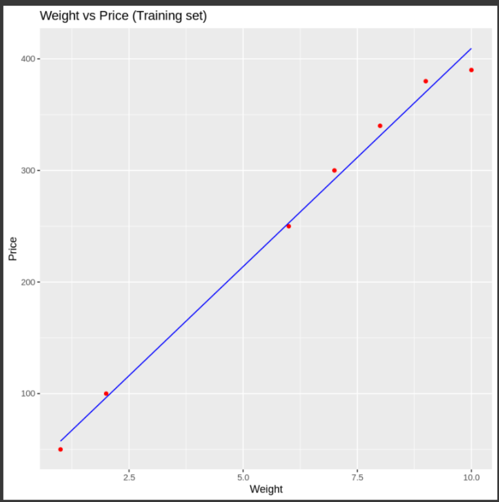
```
[ ] # Create the scatter plot
    plot(data$Weight,data$Price,
         xlab = "Weight",
         ylab = "Price",
         main = "Scatter Plot of Weight vs Price")
```



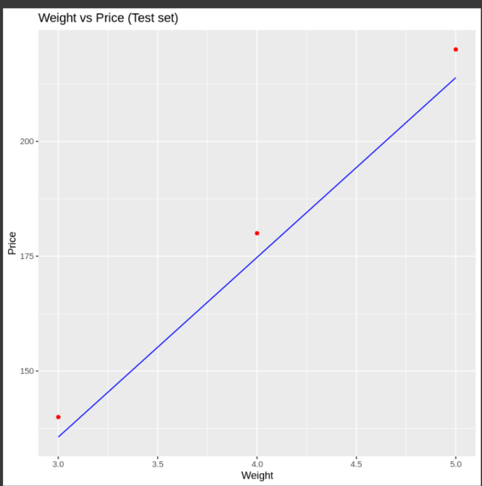

```
install.packages('caTools')
install.packages("ggplot2")
library(caTools)
library(ggplot2)
split = sample.split(data$Weight, SplitRatio = 0.7)
trainingset = subset(data, split == TRUE)
testset = subset(data, split == FALSE)
Weight<-data$Weight
Price<-data$Price
# Fitting Simple Linear Regression to the Training set
linear_model= lm(Price ~ Weight, trainingset)
#Summary of the model
summary(linear_model)
```

```
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)


Call:
lm(formula = Price ~ Weight, data = trainingset)

Residuals:
      1       2       3       4       5       7       8 
-5.8333  3.3333  2.5000  1.6667  0.8333 -0.8333 -1.6667 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   15.000      2.665   5.629  0.00245 ** 
Weight        40.833      0.544  75.067 7.95e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.416 on 5 degrees of freedom
Multiple R-squared:  0.9991,	Adjusted R-squared:  0.9989 
F-statistic:  5635 on 1 and 5 DF,  p-value: 7.948e-09
```

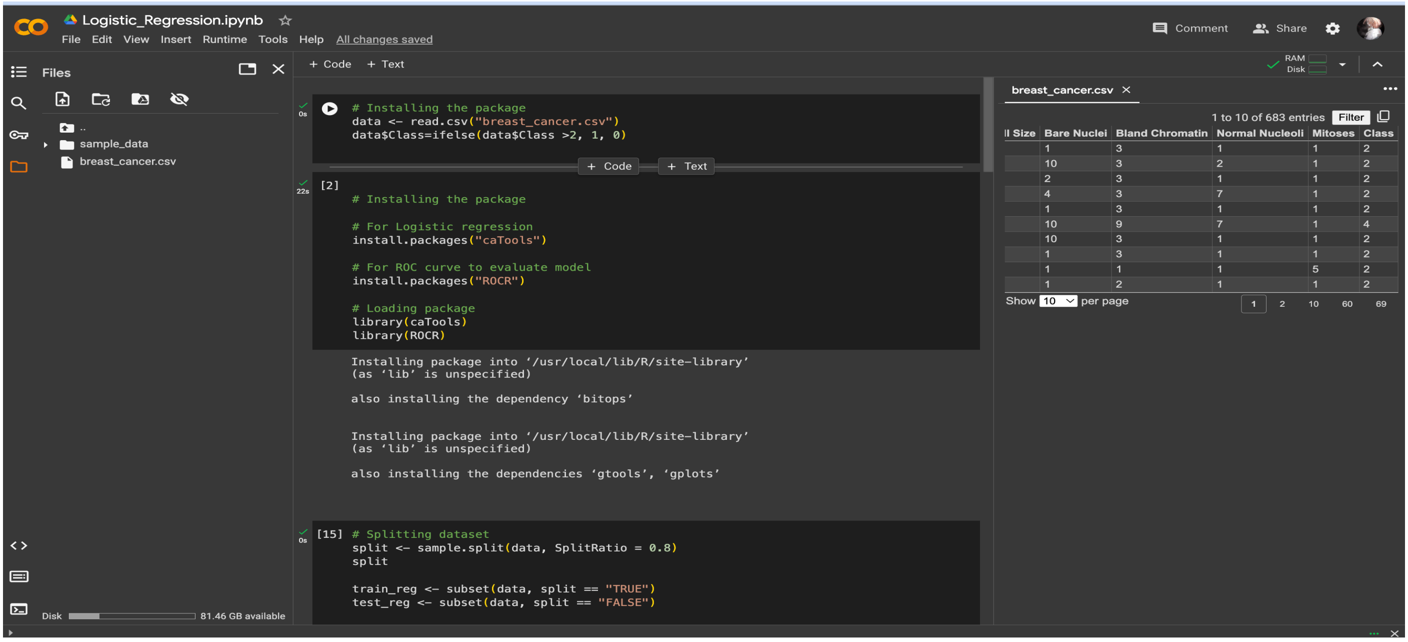
```
# Visualising the Training set results
ggplot() + geom_point(aes(x = trainingset$Weight,
                          y = trainingset$Price), colour = 'red') +
  geom_line(aes(x = trainingset$Weight,
                y = predict(linear_model, newdata = trainingset)), colour = 'blue') +

  ggtitle('Weight vs Price (Training set)') +
  xlab('Weight') +
  ylab('Price')
```




```
# Visualising the Test set results
ggplot() +
  geom_point(aes(x = testset$Weight, y = testset$Price),
             colour = 'red') +
  geom_line(aes(x = testset$Weight,
                y = predict(linear_model, newdata = testset)),
            colour = 'blue') +
  ggtitle('Weight vs Price (Test set)') +
  xlab('Weight') +
  ylab('Price')
```



Q.3) Implement Logistic Regression in R and visualize the results.




```r
# Splitting dataset
split <- sample.split(data, SplitRatio = 0.8)
split

train_reg <- subset(data, split == "TRUE")
test_reg <- subset(data, split == "FALSE")

# Training model
logistic_model <- glm(Class ~ Clump.Thickness,
                      data = train_reg,
                      family = "binomial")
logistic_model

# Summary
summary(logistic_model)
```

```
TRUE · TRUE · TRUE · FALSE · TRUE · TRUE · TRUE · TRUE · TRUE · FALSE

Call:  glm(formula = Class ~ Clump.Thickness, family = "binomial", data = train_reg)

Coefficients:
    (Intercept)  Clump.Thickness
        -5.2444           0.9566

Degrees of Freedom: 546 Total (i.e. Null);  545 Residual
Null Deviance:      717.2
Residual Deviance: 356.1        AIC: 360.1

Call:
glm(formula = Class ~ Clump.Thickness, family = "binomial", data = train_reg)

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)      -5.2444     0.4340  -12.08   <2e-16 ***
Clump.Thickness   0.9566     0.0843   11.35   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 717.20  on 546  degrees of freedom
Residual deviance: 356.14  on 545  degrees of freedom
AIC: 360.14

Number of Fisher Scoring iterations: 6
```

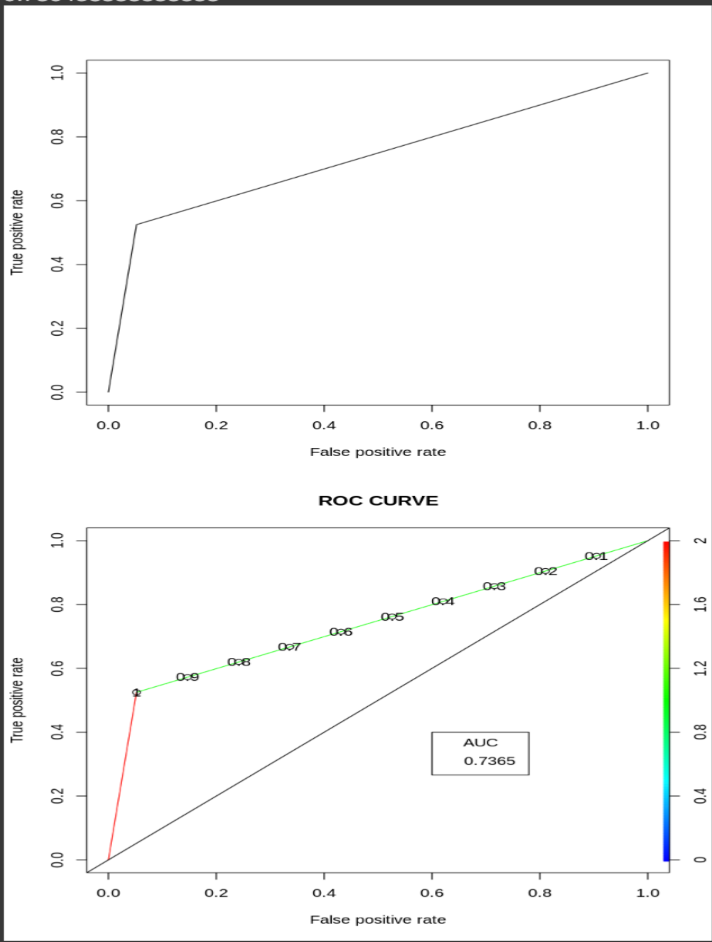
```r
predict_reg <- predict(logistic_model,
                       test_reg, type = "response")
predict_reg
```

```
4:     0.621368849099069 10:     0.194991217607226 14:     0.0135491432059273 20:     0.621368849099069 24:
    0.0135491432059273 30:     0.0851357091469239 34:     0.0851357091469239 40:     0.986899067841679 44:
    0.0135491432059273 50:     0.386685024613683 54:     0.986899067841679 60:     0.0135491432059273 64:
    0.986899067841679 70:     0.621368849099069 74:     0.0135491432059273 80:     0.194991217607226 84:
    0.386685024613683 90:     0.0851357091469239 94:     0.0135491432059273 100:     0.0345175616062113 104:
    0.810304947637753 110:     0.917482519326068 114:     0.0135491432059273 120:     0.194991217607226 124:
    0.0135491432059273 130:     0.0345175616062113 134:     0.386685024613683 140:     0.966600807721092 144:
    0.0135491432059273 150:     0.194991217607226 154:     0.0345175616062113 160:     0.194991217607226 164:
    0.0135491432059273 170:     0.386685024613683 174:     0.386685024613683 180:     0.0345175616062113 184:
    0.0135491432059273 190:     0.194991217607226 194:     0.0851357091469239 200:     0.386685024613683 204:
    0.386685024613683 210:     0.917482519326068 214:     0.621368849099069 220:     0.0135491432059273 224:
    0.986899067841679 230:     0.986899067841679 234:     0.386685024613683 240:     0.917482519326068 244:
    0.986899067841679 250:     0.0851357091469239 254:     0.386685024613683 260:     0.0851357091469239 264:
    0.386685024613683 270:     0.0135491432059273 274:     0.0135491432059273 280:     0.621368849099069 284:
    0.986899067841679 290:     0.0135491432059273 294:     0.986899067841679 300:     0.0135491432059273 304:
    0.386685024613683 310:     0.386685024613683 314:     0.0135491432059273 320:     0.386685024613683 324:
    0.0135491432059273 330:     0.0135491432059273 334:     0.0135491432059273 340:     0.0345175616062113 344:
    0.917482519326068 350:     0.194991217607226 354:     0.386685024613683 360:     0.386685024613683 364:
    0.0135491432059273 370:     0.0345175616062113 374:     0.386685024613683 380:     0.0135491432059273 384:
    0.194991217607226 390:     0.0851357091469239 394:     0.0135491432059273 400:     0.917482519326068 404:
    0.386685024613683 410:     0.0851357091469239 414:     0.0135491432059273 420:     0.621368849099069 424:
    0.194991217607226 430:     0.386685024613683 434:     0.0135491432059273 440:     0.0345175616062113 444:
    0.386685024613683 450:     0.194991217607226 454:     0.194991217607226 460:     0.386685024613683 464:
    0.386685024613683 470:     0.386685024613683 474:     0.0851357091469239 480:     0.386685024613683 484:
    0.194991217607226 490:     0.0135491432059273 494:     0.386685024613683 500:     0.621368849099069 504:
    0.0851357091469239 510:     0.0851357091469239 514:     0.621368849099069 520:     0.0345175616062113 524:
    0.194991217607226 530:     0.0345175616062113 534:     0.0851357091469239 540:     0.0851357091469239 544:
    0.0345175616062113 550:     0.194991217607226 554:     0.917482519326068 560:     0.986899067841679 564:
    0.0135491432059273 570:     0.386685024613683 574:     0.966600807721092 580:     0.194991217607226 584:
    0.0851357091469239 590:     0.386685024613683 594:     0.386685024613683 600:     0.0345175616062113 604:
    0.386685024613683 610:     0.0851357091469239 614:     0.194991217607226 620:     0.0851357091469239 624:
    0.386685024613683 630:     0.0851357091469239 634:     0.0851357091469239 640:     0.0851357091469239 644:
    0.0135491432059273 650:     0.0135491432059273 654:     0.386685024613683 660:     0.194991217607226 664:
    0.0345175616062113 670:     0.0135491432059273 674:     0.0135491432059273 680:     0.0345175616062113
```

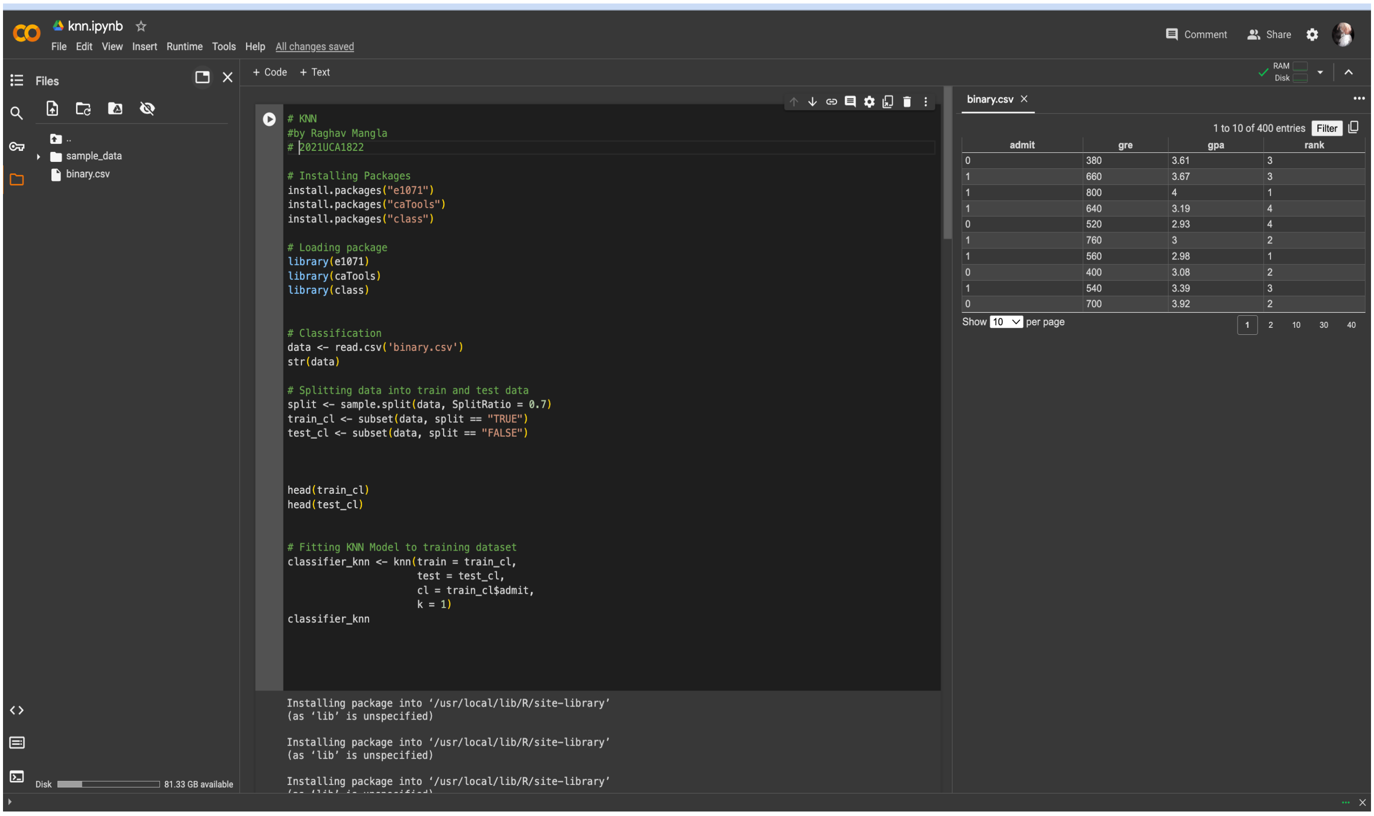
```r
# Changing probabilities
predict_reg <- ifelse(predict_reg >0.5, 1, 0)

# Evaluating model accuracy
# using confusion matrix
table(test_reg$Class, predict_reg)

missing_classerr <- mean(predict_reg != test_reg$Class)
print(paste('Accuracy =', 1 - missing_classerr))

# ROC-AUC Curve
ROCPred <- prediction(predict_reg, test_reg$Class)
ROCPer <- performance(ROCPred, measure = "tpr",
          x.measure = "fpr")

auc <- performance(ROCPred, measure = "auc")
auc <- auc@y.values[[1]]
auc

# Plotting curve
plot(ROCPer)
plot(ROCPer, colorize = TRUE,
  print.cutoffs.at = seq(0.1, by = 0.1),
  main = "ROC CURVE")
abline(a = 0, b = 1)

auc <- round(auc, 4)
legend(.6, .4, auc, title = "AUC", cex = 1)
```

```
   predict_reg
     0  1
  0 91  5
  1 19 21
[1] "Accuracy = 0.823529411764706"
```

```
[1] "Accuracy = 0.823529411764706"
0.736458333333333
```



Q.4) **Implement any Machine learning Algorithm along with feature selection and data visualization on any dataset of your choice.**



```
# KNN
#by Raghav Mangla
# 2021UCA1822

# Installing Packages
install.packages("e1071")
install.packages("caTools")
install.packages("class")

# Loading package
library(e1071)
library(caTools)
library(class)


# Classification
data <- read.csv('binary.csv')
str(data)

# Splitting data into train and test data
split <- sample.split(data, SplitRatio = 0.7)
train_cl <- subset(data, split == "TRUE")
test_cl <- subset(data, split == "FALSE")


head(train_cl)
head(test_cl)


# Fitting KNN Model to training dataset
classifier_knn <- knn(train = train_cl,
                      test = test_cl,
                      cl = train_cl$admit,
                      k = 1)

classifier_knn
```

```
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

Installing package into '/usr/local/lib/R/site-library'
```

binary.csv

1 to 10 of 400 entries

admit	gre	gpa	rank
0	380	3.61	3
1	660	3.67	3
1	800	4	1
1	640	3.19	4
0	520	2.93	4
1	760	3	2
1	560	2.98	1
0	400	3.08	2
1	540	3.39	3
0	700	3.92	2

Show 10 per page

```
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

'data.frame':   400 obs. of  4 variables:
 $ admit: int  0 1 1 1 0 1 1 0 1 0 ...
 $ gre  : int  380 660 800 640 520 760 560 400 540 700 ...
 $ gpa  : num  3.61 3.67 4 3.19 2.93 3 2.98 3.08 3.39 3.92 ...
 $ rank : int  3 3 1 4 4 2 1 2 3 2 ...
```

A data.frame: 6 × 4

	admit	gre	gpa	rank
	<int>	<int>	<dbl>	<int>
2	1	660	3.67	3
3	1	800	4.00	1
6	1	760	3.00	2
7	1	560	2.98	1
10	0	700	3.92	2
11	0	800	4.00	4

A data.frame: 6 × 4

	admit	gre	gpa	rank
	<int>	<int>	<dbl>	<int>
1	0	380	3.61	3
4	1	640	3.19	4
5	0	520	2.93	4
8	0	400	3.08	2
9	1	540	3.39	3
12	0	440	3.22	1

Clear output

executed by RAGHAV MA
8:24 PM (2 minutes ago)
executed in 11.353s

```
0·0·0·0·1·0·1·0·1·1·0·0·1·1·1·1·0·0·0·1·0·0·0·0·0·0·0·0·0·0·1·1·0·1·1·0·1·0·0·1·1·0·1·0·0·1·0·0·0·
0·0·0·1·0·0·0·0·0·0·0·1·0·0·0·0·0·0·0·0·1·0·0·0·0·1·0·1·0·0·0·0·0·0·0·0·0·0·1·0·0·0·1·0·0·0·0·0·0·
0·0·0·0·1·1·0·0·0·1·0·1·0·0·0·0·0·0·0·0·1·0·0·0·0·0·0·0·1·0·1·0·0·1·1·0·1·0·1·1·0·1·0·0·0·1·0·0·0·
0·1·0·0·1·0·0·0·0·0·0·0·0·0·0·0·1·0·0·0·1·0·1·0·0·0·0·0·0·1·1·0·0·1·0·1·0·0·1·1·0·0·0·0·0·1·0·0·1·
1·0·0·0
► Levels:
```

```
[24] # Confusiin Matrix
     cm <- table(test_cl$admit, classifier_knn)
     cm
```

```
   classifier_knn
      0   1
  0 139   7
  1   8  46
```

```
# Model Evaluation - Choosing K
# Calculate out of Sample error
misClassError <- mean(classifier_knn != test_cl$admit)
print(paste('Accuracy =', 1-misClassError))

# K = 3
classifier_knn <- knn(train = train_cl,
                      test = test_cl,
                      cl = train_cl$admit,
                      k = 3)
misClassError <- mean(classifier_knn != test_cl$admit)
print(paste('Accuracy =', 1-misClassError))

# K = 5
classifier_knn <- knn(train = train_cl,
                      test = test_cl,
                      cl = train_cl$admit,
                      k = 5)
misClassError <- mean(classifier_knn != test_cl$admit)
print(paste('Accuracy =', 1-misClassError))

# K = 7
classifier_knn <- knn(train = train_cl,
                      test = test_cl,
                      cl = train_cl$admit,
                      k = 7)
misClassError <- mean(classifier_knn != test_cl$admit)
print(paste('Accuracy =', 1-misClassError))

# K = 15
classifier_knn <- knn(train = train_cl,
                      test = test_cl,
                      cl = train_cl$admit,
                      k = 15)
misClassError <- mean(classifier_knn != test_cl$admit)
print(paste('Accuracy =', 1-misClassError))

# K = 19
classifier_knn <- knn(train = train_cl,
                      test = test_cl,
                      cl = train_cl$admit,
                      k = 19)
misClassError <- mean(classifier_knn != test_cl$admit)
print(paste('Accuracy =', 1-misClassError))
```

```
[1] "Accuracy = 0.925"
[1] "Accuracy = 0.895"
[1] "Accuracy = 0.86"
[1] "Accuracy = 0.8"
[1] "Accuracy = 0.795"
[1] "Accuracy = 0.79"
```

```
install.packages("ggplot2");
library(ggplot2)

# Data preparation
k_values <- c(1, 3, 5, 7, 15, 19)

# Calculate accuracy for each k value
accuracy_values <- sapply(k_values, function(k) {
  classifier_knn <- knn(train = train_scale,
                        test = test_scale,
                        cl = train_cl$admit,
                        k = k)
  1 - mean(classifier_knn != test_cl$admit)
})

# Create a data frame for plotting
accuracy_data <- data.frame(K = k_values, Accuracy = accuracy_values)

# Plotting
ggplot(accuracy_data, aes(x = K, y = Accuracy)) +
  geom_line(color = "lightblue", size = 1) +
  geom_point(color = "lightgreen", size = 3) +
  labs(title = "Model Accuracy for Different K Values",
       x = "Number of Neighbors (K)",
       y = "Accuracy") +
  theme_minimal()
```

```
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)
```