Program #1

List and explain the different variables, constants and operators in R.

SOURCE CODE

/*Alan Payyappilly*/

Variables in R

Variables are used to store data, whose value can be changed according to our need. Unique name given to variable (function and objects as well) is identifier.

Rules for writing Identifiers in R

- Identifiers can be a combination of letters, digits, period (.) and underscore (_).
- It must start with a letter or a period. If it starts with a period, it cannot be followed by a digit.
- Reserved words in R cannot be used as identifiers.

  Valid identifiers in R
  total, Sum, .fine.with.dot, this_is_acceptable, Number5

  Invalid identifiers in R
  tot@l, 5um, _fine, TRUE, .0ne

Constants in R

Constants, as the name suggests, are entities whose value cannot be altered. Basic types of constant are numeric constants and character constants.

Numeric Constants

All numbers fall under this category. They can be of type integer, double or complex.
It can be checked with the typeof() function.
Numeric constants followed by L are regarded as integer and those followed by i are regarded as complex.

```
typeof(5)
typeof(5L)
typeof(5i)
```

OUTPUT

```
> typeof(5)
[1] "double"
> typeof(5L)
[1] "integer"
> typeof(5i)
[1] "complex"
>
```

Numeric constants preceded by 0x or 0X are interpreted as hexadecimal numbers.

Examples

0XFF

0XA + 2

```
> 0XFF
[1] 255
>  0XA + 2
[1] 12
>
```

Character Constants

Character constants can be represented using either single quotes (') or double quotes (") as delimiters.

Example

typeof("5")

typeof("Run")

OUTPUT

```
> typeof("5")
[1] "character"
> typeof("Run")
[1] "character"
>
```

Built-in Constants

Some of the built-in constants defined in R along with their values is shown below.

Example

LETTERS

OUTPUT

```
>
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T" "U"
[22] "V" "W" "X" "Y" "Z"
>
```

letters

```
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u"
[22] "v" "w" "x" "y" "z"
>
```

pi

```
> pi
[1] 3.141593
>
```

month.name

```
> month.name
 [1] "January"   "February"  "March"     "April"     "May"       "June"      "July"
 [8] "August"    "September" "October"   "November"  "December"
>
```

month.abb

```
> month.abb
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
>
```

Operators

The operators are those symbols which tell the compiler for performing precise mathematical or logical manipulations. R programming is loaded with built in operators and supplies below mentioned types of operators.

Types of Operators

- The Arithmetic Operators
- The Relational Operators
- The Logical Operators
- The Assignment Operators

The below mentioned table gives the arithmetic operators hold up by R language. The operators act on each element of the vector.

Arithmetic Operators

These operators are used to carry out mathematical operations like addition and multiplication. Here is a list of arithmetic operators available in R.

Operator	Description
+	Addition
–	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus (Remainder from division)
%/%	Integer Division

SOURCE CODE

```
x <- 4
y <- 16
x+y
x-y
x*y
y/x
y%/%x
y%%x
y^x
```

OUTPUT

```
> x <- 4
> y <- 16
> x+y
[1] 20
> x-y
[1] -12
> x*y
[1] 64
> y/x
[1] 4
> y%/%x
[1] 4
> y%%x
[1] 0
> y^x
[1] 65536
>
```

Relational Operators

Relational operators are used to compare between values. Here is a list of relational operators available in R.

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

SOURCE CODE

```
x <- 4
y <- 16
x<y
x>y
x<=5
y>=20
y == 16
x != 4
```

OUTPUT

```
> x <- 4
> y <- 16
> x<y
[1] TRUE
> x>y
[1] FALSE
> x<=5
[1] TRUE
> y>=20
[1] FALSE
> y == 16
[1] TRUE
> x != 4
[1] FALSE
```

Logical Operators

Logical operators are used to carry out Boolean operations like AND, OR etc.

Operator	Description
!	Logical NOT
&	Element-wise logical AND

<table>
<tr><td>&&</td><td>Logical AND</td></tr>
<tr><td>|</td><td>Element-wise logical OR</td></tr>
<tr><td>||</td><td>Logical OR</td></tr>
</table>

Operators & and | perform element-wise operation producing result having length of the longer operand. But && and || examines only the first element of the operands resulting into a single length logical vector. Zero is considered false and non-zero numbers are taken as true.

SOURCE CODE

```
x <- c(TRUE,FALSE,0,6)
y <- c(FALSE,TRUE,FALSE,TRUE)
!x
x&y
x&&y
x|y
x||y
```

OUTPUT

```
> x <- c(TRUE,FALSE,0,6)
> y <- c(FALSE,TRUE,FALSE,TRUE)
> !x
[1] FALSE  TRUE  TRUE FALSE
> x&y
[1] FALSE FALSE FALSE  TRUE
> x&&y
[1] FALSE
> x|y
[1]  TRUE  TRUE FALSE  TRUE
> x||y
[1] TRUE
```

Assignment Operators

These operators are used to assign values to variables.

Operator	Description
<-, <<-, =	Leftwards assignment
->, ->>	Rightwards assignment

The operators `<-` and `=` can be used, almost interchangeably, to assign to variable in the same environment.

The `<<-` operator is used for assigning to variables in the parent environments (more like global assignments). The rightward assignments, although available are rarely used.

SOURCE CODE

```
x <- 5
x

x = 9
x

10 -> x
x
```

OUTPUT

```
> x <- 5
> x
[1] 5
> x = 9
> x
[1] 9
> 10 -> x
> x
[1] 10
>
```

Program #2

List and explain the Vector data type.

/*Alan Payyappilly*/

Vector

Vectors are the most basic R data objects and there are six types of atomic vectors.

Vector Creation

Single Element Vector

Even when you write just one value in R, it becomes a vector of length 1 and belongs to one of the above vector types.

Examples

SOURCE CODE

```
# Atomic vector of type character.
print("abc");
# Atomic vector of type double.
print(12.5)
```

OUTPUT

```
[1] "abc"
[2] 12.5
```

Multiple Elements Vector

Using colon operator with numeric data

SOURCE CODE

```
# Creating a sequence from 5 to 13.
v <-5:13
print(v)
# Creating a sequence from 6.6 to 12.6.
```

OUTPUT

```
[1] 5 6 7 8 9 10 11 12 13
```

[1] 6.6 7.6 8.6 9.6 10.6 11.6 12.6

SOURCE CODE

```
# Create vector with elements from 5 to 9 incrementing by 0.4.
print(seq(5, 9, by = 0.4))
```

OUTPUT

[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0

Using the c() function

The non-character values are coerced to character type if one of the elements is a character.

SOURCE CODE

```
# The logical and numeric values are converted to characters.
s <- c('apple','red',5, TRUE)
print(s)
```

OUTPUT

[1] "apple" "red" "5" "TRUE"

Accessing Vector Elements

Elements of a Vector are accessed using indexing. The [ ] brackets are used for indexing. Indexing starts with position 1. Giving a negative value in the index drops that element from result. TRUE, FALSE or 0 and 1 can also be used for indexing.

SOURCE CODE

```
# Accessing vector elements using position.
t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u <- t[c(2,3,6)]
print(u)
# Accessing vector elements using logical indexing.
v <- t[c(TRUE, FALSE,FALSE,FALSE,FALSE,TRUE,FALSE)]
print(v)
```

OUTPUT

[1] "Mon" "Tue" "Fri"
[1] "Sun" "Fri"

Program #3

List and explain the List data type.

/*Alan Payyappilly*/

List

Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using **list()** function.

Creating a List

Following is an example to create a list containing strings, numbers, vectors and a logical values.

SOURCE CODE

```
# Create a list containing strings, numbers, vectors and a logical
# values.
list_data<- list("Red", "Green", c(21,32,11), TRUE, 51.23)
print(list_data)
```

OUTPUT

```
[1] "Red"
[1] "Green"
[1] 21 32 11
[1] TRUE
[1] 51.23
```

Accessing List Elements

Elements of the list can be accessed by the index of the element in the list. In case of named lists it can also be accessed using the names.

We continue to use the list in the above example –

SOURCE CODE

```
# Create a list containing a vector, a matrix and a list.
list_data<- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8),nrow=2), list("green",12.3))
# Give names to the elements in the list.
names(list_data)<- c("1st Quarter","A_Matrix","A Inner list")
# Access the first element of the list.
print(list_data[1])
```

OUTPUT

```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"
```

Program #4

List and explain the Matrix data type.

/*Alan Payyappilly*/

Matrices

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types. Though we can create a matrix containing only characters or only logical values, they are not of much use. We use matrices containing numeric elements to be used in mathematical calculations.
A Matrix is created using the **matrix**() function.

Syntax:

The basic syntax for creating a matrix in R is − matrix(data, nrow, ncol, byrow, dimnames)
Following is the description of the parameters used −

- **data** is the input vector which becomes the data elements of the matrix.
- **nrow** is the number of rows to be created.
- **ncol** is the number of columns to be created.
- **byrow** is a logical clue. If TRUE then the input vector elements are arranged by row.
- **dimname** is the names assigned to the rows and columns.

Example

Create a matrix taking a vector of numbers as input.

SOURCE CODE

```
# Elements are arranged sequentially by row.

M <- matrix(c(3:14), nrow = 4, byrow = TRUE)

print(M)
```

OUTPUT

```
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    6    7    8
[3,]    9   10   11
[4,]   12   13   14
```

Accessing Elements of a Matrix

Elements of a matrix can be accessed by using the column and row index of the element. We consider the matrix P above to find the specific elements below.

SOURCE CODE

```
# Define the column and row names.
rownames= c("row1","row2","row3","row4")
colnames= c("col1","col2","col3")
# Create the matrix.
P <- matrix(c(3:14),nrow=4,byrow= TRUE,dimnames= list(rownames,colnames))
# Access the element at 3rd column and 1st row.
```

OUTPUT

```
[1] 5
```

Program #5

List and explain the Arrays data type.

/*Alan Payyappilly*/

Arrays

Arrays are the R data objects which can store data in more than two dimensions. For example − If we create an array of dimension (2, 3, 4) then it creates 4 rectangular matrices each with 2 rows and 3 columns. Arrays can store only data type.
An array is created using the array() function. It takes vectors as input and uses the values in the **dim** parameter to create an array.
Example
The following example creates an array of two 3x3 matrices each with 3 rows and 3 columns.

SOURCE CODE

```
# Create two vectors of different lengths.

vector1 <- c(5,9,3) vector2 <- c(10,11,12,13,14,15)

 # Take these vectors as input to the array.

result<- array(c(vector1,vector2),dim = c(3,3,2))

print(result)
```

OUTPUT

```
 , , 1
     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

, , 2
     [,1]   [,2]   [,3]
[1,]  5      10     13
[2,]  9      11     14
[3,]  3      12     15
```

Program #6

List and explain the Factors data type.

/*Alan Payyappilly*/

Factors

Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like "Male, "Female" and True, False etc. They are useful in data analysis for statistical modeling.
Factors are created using the factor () function by taking a vector as input
Example

SOURCE CODE

```
# Create a vector as input.

data <-c("East","West","East","North","North","East","West","West","West","East","North")
print(data)

print(is.factor(data))

 # Apply the factor function.

factor_data<- factor(data)
```

OUTPUT

[1] "East" "West" "East" "North" "North" "East" "West" "West" "West" "East" "North"
[1] FALSE
[1] East West East North North East West WestWest East North Levels: East North West
[1]TRUE

Generating Factor Levels

We can generate factor levels by using the gl() function. It takes two integers as input which indicates how many levels and how many times each level.

Syntax

gl(n, k, labels)

Following is the description of the parameters used −

- **n** is a integer giving the number of levels.
- **k** is a integer giving the number of replications.
- **labels** is a vector of labels for the resulting factor levels.

SOURCE CODE

```
v <- gl(3, 4, labels = c("Tampa", "Seattle","Boston"))

print(v)
```

OUTPUT

When we execute the above code, it produces the following result −

```
Tampa TampaTampaTampa Seattle SeattleSeattleSeattle Boston
[10]     Boston BostonBoston
Levels: Tampa Seattle Boston
```

Program #7

List and explain the Data Frames data type.

/*Alan Payyappilly*/

DataFrames

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column. Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

Create Data Frame

SOURCE CODE

```
# Create the data frame.

emp.data<-data.frame( emp_id= c (1:5), emp_name=c("Rick","Dan","Michelle","Ryan","Gary"),
salary= c(623.3,515.2,611.0,729.0,843.25),

start_date=as.Date(c("2012-01-01","2013-09-23","2014-11-15","2014-05-11", "2015-03-27")),

stringsAsFactors= FALSE )

# Print the data frame.

print(emp.data)
```

OUTPUT

```
   emp_idemp_name  salary  start_date
1  1     Rick      623.30 2012-01-01
2  2     Dan       515.20 2013-09-23
3  3     Michelle  611.00 2014-11-15
4  4     Ryan      729.00 2014-05-11
5  5     Gary      843.25 2015-03-27
```

Program #8

Explain read and write from console (print and scan).

/*Alan Payyappilly*/

- **scan()**

Read Data Values: This is used for reading data into the input vector or an input list from the environment console or file. Keywords: File, connection.
For example:

```
>#Author DataFlair
>inp = scan()
>inp
```

- **print()**

Print prints its argument and returns it invisibly (via invisible(x)). It is a generic function which means that new printing methods can be easily added for new classes.
Keywords: print

Usage

```
print(x, …)
# S3 method for factor
print(x, quote = FALSE, max.levels = NULL, width = getOption("width"), …)
# S3 method for table
print(x, digits = getOption("digits"), quote = FALSE, na.print = "", zero.print = "0",
right = is.numeric(x) || is.complex(x), justify = "none", …)
# S3 method for function print(x, useSource = TRUE, …)
```

Program #9

Explain read and write from files (.csv) in R.

/*Alan Payyappilly*/

CSV Files in R

Let’s start by opening a .csv file containing information on the speeds at which cars of different colors were clocked in 45 mph zones in the four-corners states (CarSpeeds.csv). We will use the builtin read.csv(...) function call, which reads the data in as a data frame, and assign the data frame to a variable (using <-) so that it is stored in R’s memory

SOURCE CODE

```
carSpeeds <- read.csv(file =
'data/carspeeds.csv') head(carSpeeds)
```

OUTPUT

```
 Color  Speed  State
1 Blue  32     NewMexico
2 Red   45     Arizona
3 Blue  35     Colorado
4 White        34     Arizona
5 Red   25     Arizona
6 Blue  41     Arizona
```

Program #10

Demonstrate summary function, different measures of Central Tendency and measures of Dispersion?

/*Alan Payyappilly*/

Summary()

Summary function is a generic function used to produce result summaries of the results of various model fitting functions.

create vector
gender<-c("male","female") height<-c(152,171.5) weight<-c(81,55)

create data frame

BMI<-data.frame(gender,height,weight)
BMI
summary(BMI)

OUTPUT

```
  gender height weight
1   male  152.0     81
2 female  171.5     55
> summary(BMI)
    gender      height          weight
 female:1   Min.   :152.0   Min.   :55.0
 male  :1   1st Qu.:156.9   1st Qu.:61.5
            Median :161.8   Median :68.0
            Mean   :161.8   Mean   :68.0
            3rd Qu.:166.6   3rd Qu.:74.5
            Max.   :171.5   Max.   :81.0
> |
```

Measures of central tendency:

Mean,Median,Mode

SOURCE CODE

```
x1 <- c(18, 19, 19, 19, 19, 20, 20, 20, 20, 20, 21, 21, 21, 21, 22, 23, 24, 27, 30, 36)
x1
mean(x1)
median(x1)
x1[x1<25]
median(x1[x1<25])
```

OUTPUT

```
> x1
 [1] 18 19 19 19 19 20 20 20 20 20 21 21 21 21 22 23 24 27 30 36
> mean(x1)
[1] 22
> median(x1)
[1] 20.5
> x1[x1<25]
 [1] 18 19 19 19 19 20 20 20 20 20 21 21 21 21 22 23 24
> median(x1[x1<25])
[1] 20
```

SOURCE CODE

```
modex1 <-which(xt==max(xt))
modex1
x1 <- c(x1,19,19)
x1 mean(x1) median(x1)
xt <- table(x1)
xt
```

OUTPUT

```
> modex1 <-which(xt==max(xt))
> modex1
19
 2
> x1 <- c(x1,19,19)
> x1
 [1] 18 19 19 19 19 20 20 20 20 20 21 21 21 21 22 23 24 27 30 36 19 19 19 19
> mean(x1)
[1] 21.5
> median(x1)
[1] 20
> xt <- table(x1)
> xt
x1
18 19 20 21 22 23 24 27 30 36
 1  8  5  4  1  1  1  1  1  1
```

Measures of dispersion

Range, Quartile Range, Mean Deviation and Standard Deviation

SOURCE CODE

```
x2<-c(1.2, 1.4, 1.3, 1.6, 1.0, 1.5, 1.7, 1.1, 1.2, 1.3)
summary(x2)
rangex2 <- max(x2) - min(x2)
rangex2
range(x2)
IQR(x2)
var(x2)
sd(x2)
mean(x2)
```

OUTPUT

```
> x2<-c(1.2, 1.4, 1.3, 1.6, 1.0, 1.5, 1.7, 1.1, 1.2, 1.3)
> summary(x2)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.200   1.300   1.330   1.475   1.700
> rangex2 <- max(x2) - min(x2)
> rangex2
[1] 0.7
> range(x2)
[1] 1.0 1.7
> IQR(x2)
[1] 0.275
> var(x2)
[1] 0.049
> sd(x2)
[1] 0.2213594
> mean(x2)
[1] 1.33
>
```

SOURCE CODE

```
x3 <- rnorm(20,5,2) x3
mean(x3)
median(x3)
sd(x3)
```

OUTPUT

```
> x3 <- rnorm(20,5,2)
> x3
 [1] 4.334153 7.726227 4.061705 6.685751 2.084013 4.199388 3.447165 4.26140
7 7.480203 4.785132 5.345187 5.509203
[13] 3.770932 2.141570 4.338049 5.256772 7.036240 4.488853 4.394918 8.23038
1
> mean(x3)
[1] 4.978863
> median(x3)
[1] 4.441885
> sd(x3)
[1] 1.711571
```

SOURCE CODE

```
set.seed(100)
x<-rnorm(100, mean=0, sd=1)
mean(x)
median(x)
IQR(x)
var(x)
sd(x)
summary(x)
```

OUTPUT

```
>
> set.seed(100)
> x<-rnorm(100, mean=0, sd=1)
> mean(x)
[1] 0.002912563
> median(x)
[1] -0.0594199
> IQR(x)
[1] 1.264738
> var(x)
[1] 1.04185
> sd(x)
[1] 1.02071
> summary(x)
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-2.271926 -0.608847 -0.059420  0.002913  0.655891  2.581959
```

SOURCE CODE

```
q90<-qnorm(.90, mean = 0, sd = 1) q90
x<-seq(-4,4,.1)
x
```

OUTPUT

```
>
> q90<-qnorm(.90, mean = 0, sd = 1)
> q90
[1] 1.281552
> x<-seq(-4,4,.1)
> x
 [1] -4.0 -3.9 -3.8 -3.7 -3.6 -3.5 -3.4 -3.3 -3.2 -3.1 -3.0 -2.9 -2.8
[14] -2.7 -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2.0 -1.9 -1.8 -1.7 -1.6 -1.5
[27] -1.4 -1.3 -1.2 -1.1 -1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2
[40] -0.1  0.0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.0  1.1
[53]  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  2.0  2.1  2.2  2.3  2.4
[66]  2.5  2.6  2.7  2.8  2.9  3.0  3.1  3.2  3.3  3.4  3.5  3.6  3.7
[79]  3.8  3.9  4.0
>
```

SOURCE CODE

```
f<-dnorm(x, mean=0, sd=1)
f
```

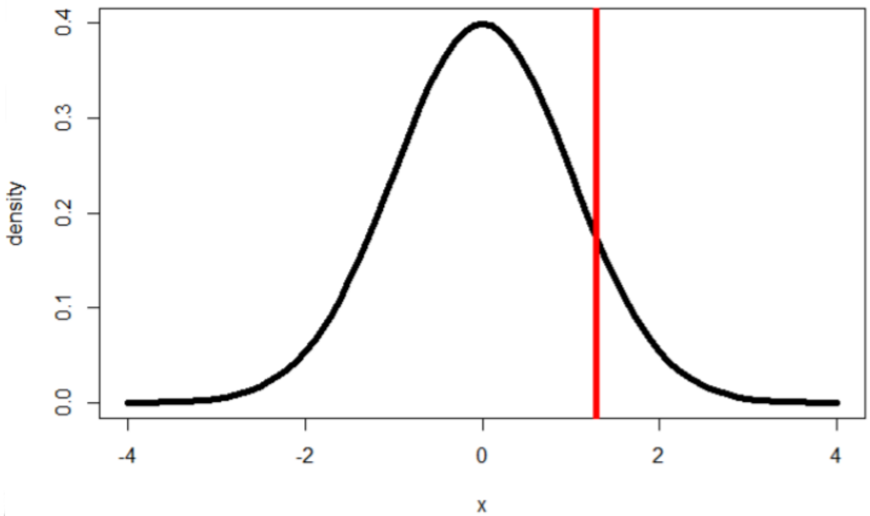
OUTPUT

```
>
> f<-dnorm(x, mean=0, sd=1)
> f
 [1] 0.0001338302 0.0001986555 0.0002919469 0.0004247803 0.0006119019
 [6] 0.0008726827 0.0012322192 0.0017225689 0.0023840882 0.0032668191
[11] 0.0044318484 0.0059525324 0.0079154516 0.0104209348 0.0135829692
[16] 0.0175283005 0.0223945303 0.0283270377 0.0354745928 0.0439835960
[21] 0.0539909665 0.0656158148 0.0789501583 0.0940490774 0.1109208347
[26] 0.1295175957 0.1497274656 0.1713685920 0.1941860550 0.2178521770
[31] 0.2419707245 0.2660852499 0.2896915528 0.3122539334 0.3332246029
[36] 0.3520653268 0.3682701403 0.3813878155 0.3910426940 0.3969525475
[41] 0.3989422804 0.3969525475 0.3910426940 0.3813878155 0.3682701403
[46] 0.3520653268 0.3332246029 0.3122539334 0.2896915528 0.2660852499
[51] 0.2419707245 0.2178521770 0.1941860550 0.1713685920 0.1497274656
[56] 0.1295175957 0.1109208347 0.0940490774 0.0789501583 0.0656158148
[61] 0.0539909665 0.0439835960 0.0354745928 0.0283270377 0.0223945303
[66] 0.0175283005 0.0135829692 0.0104209348 0.0079154516 0.0059525324
[71] 0.0044318484 0.0032668191 0.0023840882 0.0017225689 0.0012322192
[76] 0.0008726827 0.0006119019 0.0004247803 0.0002919469 0.0001986555
[81] 0.0001338302
>
```

SOURCE CODE

```
plot(x,f,xlab="x",ylab="density",type="l",lwd=5)
bline(v=q90,col=2,lwd=5)
```

OUTPUT



Program #11

Write R functions to find the sum, factorial and power.

SOURCE CODE

/*Alan Payyappilly*/

#Sum function
sum(c(2,5,6,7,1,2))

Output
[1] 23

#Factorial function
factorial(5)

Output
[1] 120

#Power function
Pow(2,2)

Output
[1] 4

Program #12

How to generate random numbers in R.

SOURCE CODE

/*Alan Payyappilly*/

#Random Generation of Numbers

```
runif(1)
runif(4)
floor(runif(3, min=0, max=101))
sample(1:100, 3, replace=TRUE)
sample(1:100, 3, replace=FALSE)
```

OUTPUT

```
> runif(1)
[1] 0.3695961
> runif(4)
[1] 0.9563228 0.9135767 0.8233363 0.3194822
> floor(runif(3, min=0, max=101))
[1] 88 80 61
> sample(1:100, 3, replace=TRUE)
[1]  8 43 35
> sample(1:100, 3, replace=FALSE)
[1] 76 22 29
> 
```

Program #13

"Generate the Cumulative Distribution Function and Probability Density Function of Normal distribution".

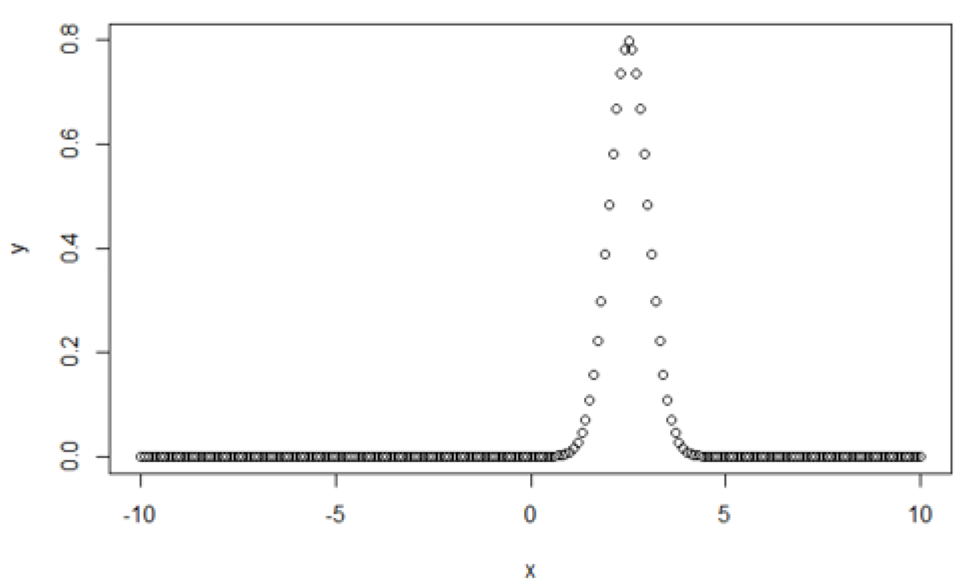
SOURCE CODE

/*Alan Payyappilly*/

#dnorm

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.1.
x <- seq(-10, 10, by = .1)
# Choose the mean as 2.5 and standard deviation as 0.5.
y <- dnorm(x, mean = 2.5, sd = 0.5)
plot(x,y)
# Save the file.
dev.off()
```

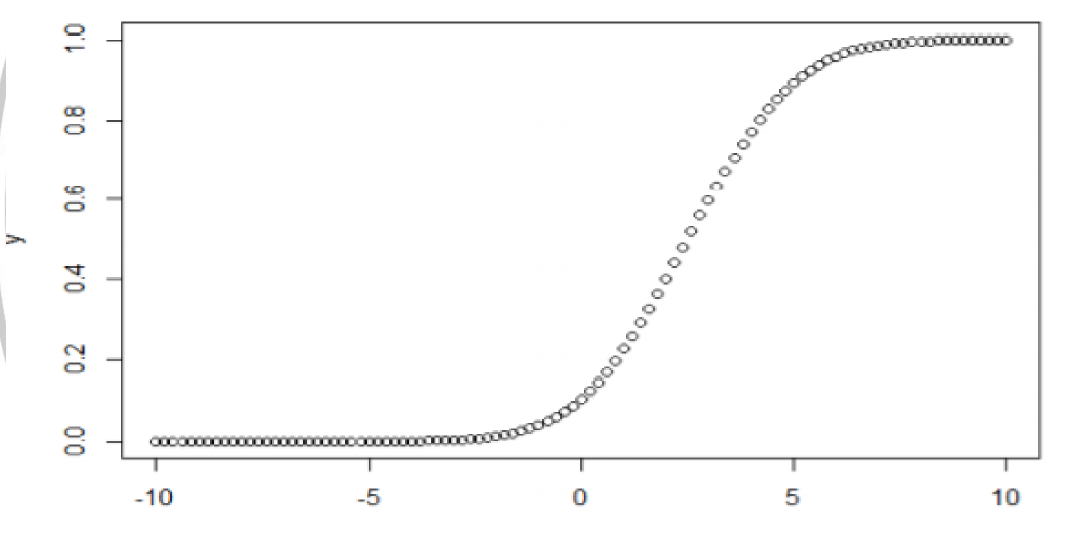
OUTPUT



SOURCE CODE

```
#pnorm
# Create a sequence of numbers between -10 and 10 incrementing by 0.2.
x <- seq(-10,10,by = .2)
# Choose the mean as 2.5 and standard deviation as 2.
y <- pnorm(x, mean = 2.5, sd = 2)
# Plot the graph.
plot(x,y)

# Save the file.
dev.off()
```

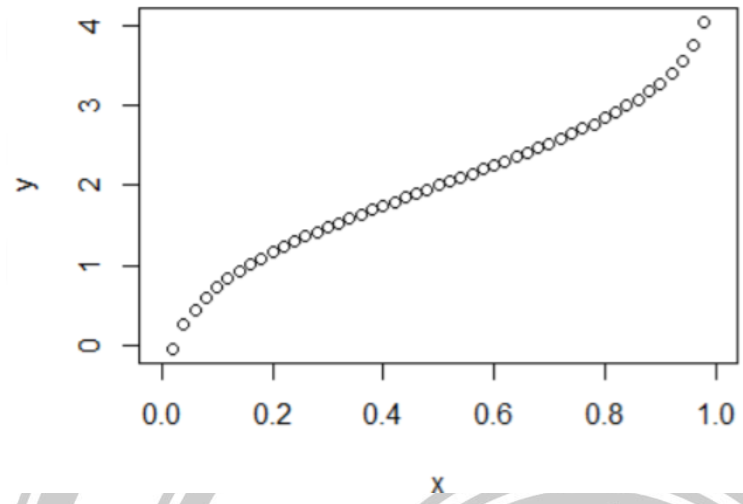
OUTPUT



SOURCE CODE

```
#qnorm
# Create a sequence of probability values incrementing by 0.02.
x <- seq(0, 1, by = 0.02)
# Choose the mean as 2 and standard deviation as 3.
y <- qnorm(x, mean = 2, sd = 1)
# Plot the graph.
plot(x,y)
# Save the file.
dev.off()
```

OUTPUT

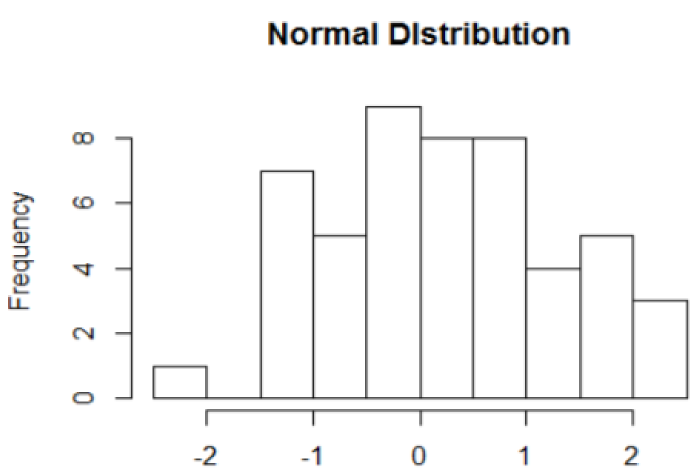


SOURCE CODE

#rnorm

```
# Create a sample of 50 numbers which are normally distributed.
y <- rnorm(50) # Plot the histogram for this sample.
hist(y, main = "Normal DIstribution")
# Save the file.
dev.off()
```

OUTPUT



Program #14

Assume that the test scores of a college entrance exam fits a normal distribution. Furthermore, the mean test score is 72, and the standard deviation is 15.2. What is the percentage of students scoring 84 or more in the exam?

SOURCE CODE

/*Alan Payyappilly*/

Solution

We apply the function pnorm of the normal distribution with mean 72 and standard deviation 15.2. Since we are looking for the percentage of students scoring higher than 84, we are interested in the upper tail of the normal distribution.

OUTPUT

```
> pnorm(84, mean=72, sd=15.2, lower.tail=FALSE)
[1] 0.2149176
> 
```

Answer

The percentage of students scoring 84 or more in the college entrance exam is 21.5%.

Program #15

"Generate the Cumulative Distribution Function and Probability Density Function of Binomial distribution".

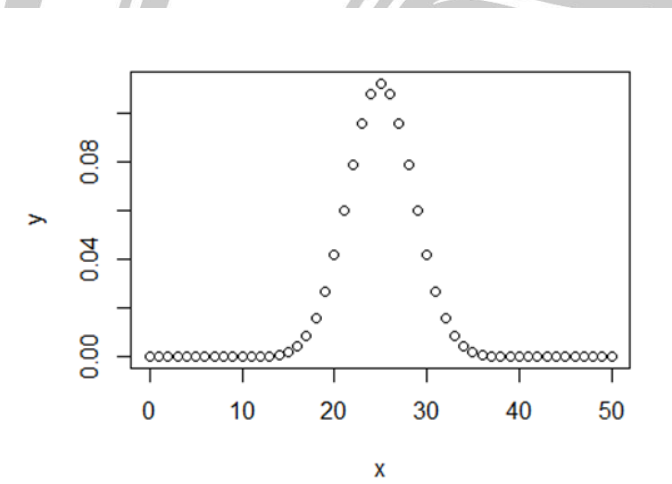
SOURCE CODE

/*Alan Payyappilly*/

#dbinom

Create a sample of 50 numbers which are incremented by 1. x <- seq(0,50,by = 1)
Create the binomial distribution. y <- dbinom(x,50,0.5)
Plot the graph for this sample. plot(x,y)
Save the file. dev.off()

OUTPUT



SOURCE CODE

#pbinom

```
# Probability of getting 26 or less heads from a 51 tosses of a coin.
x <- pbinom(26,51,0.5)
print(x)
```

OUTPUT

[1] 0.610116

SOURCE CODE

```
#qbinom
x <- qbinom(0.25,51,1/2)
print(x)
```

OUTPUT

```
[1] 23
```

SOURCE CODE

```
#rbinom
# Find 8 random values from a sample of 150 with probability of 0.4.
x <- rbinom(8,150,.4)
print(x)
```

OUTPUT

```
[1] 56 64 60 71 56 64 57 77
```

Program #16

Suppose there are twelve multiple choice questions in an English class quiz. Each question has five possible answers, and only one of them is correct. Find the probability of having four or less correct answers if a student attempts to answer every question at random.

SOURCE CODE

/*Alan Payyappilly*/

Solution

Since only one out of five possible answers is correct, the probability of answering a question correctly by random is 1/5=0.2. We can find the probability of having exactly 4 correct answers by random attempts as follows.

```
> dbinom(4, size=12, prob=0.2)
[1] 0.1328756
```

To find the probability of having four or less correct answers by random attempts, we apply the function dbinom with x = 0,…,4.

OUTPUT

```
> dbinom(0, size=12, prob=0.2) +
+    + dbinom(1, size=12, prob=0.2) +
+    + dbinom(2, size=12, prob=0.2) +
+    + dbinom(3, size=12, prob=0.2) +
+    + dbinom(4, size=12, prob=0.2)
[1] 0.9274445
```

Alternatively, we can use the cumulative probability function for binomial distribution pbinom.

```
> pbinom(4, size=12, prob=0.2)
[1] 0.9274445
>
```

Answer

The probability of four or less questions answered correctly by random in a twelve question multiple choice quiz is 92.7%.

Program #17

"Generate the Cumulative Distribution Function and Probability Density Function of Poisson distribution".

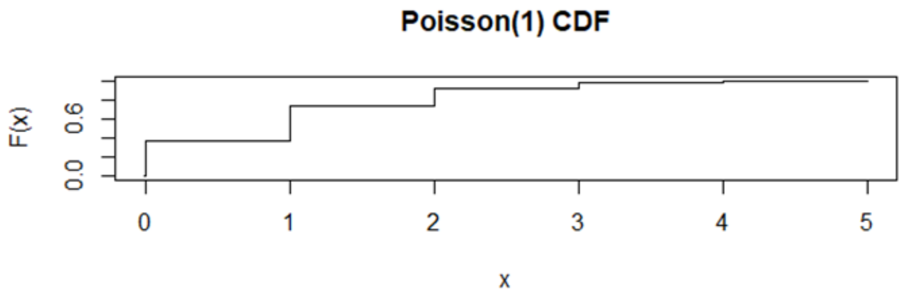
SOURCE CODE

/*Alan Payyappilly*/

```
require(graphics)
-log(dpois(0:7, lambda = 1) * gamma(1+ 0:7)) # == 1
Ni <- rpois(50, lambda = 4); table(factor(Ni, 0:max(Ni)))
1 - ppois(10*(15:25), lambda = 100) # becomes 0 (cancellation) ppois(10*(15:25), lambda =
100, lower.tail = FALSE) # no cancellation par(mfrow = c(2, 1))
x <- seq(-0.01, 5, 0.01)
plot(x, ppois(x, 1), type = "s", ylab = "F(x)", main = "Poisson(1) CDF") #qpois function
lower<-qpois(0.001, lambda=2.5) upper<-qpois(0.999, lambda=2,5) n<-seq(lower,upper,1)
q<-seq(0.001,0.999,0.001)
dPoisson25 <- data.frame(N=n,
Density=dpois(n, lambda=2.5), Distribution=ppois(n, lambda=2.5))
qPoisson25 <- data.frame(Q=q, Quantile=qpois(q, lambda=2.5)) head(dPoisson25)
head(qPoisson25)
```

OUTPUT

```
>
> require(graphics)
> -log(dpois(0:7, lambda = 1) * gamma(1+ 0:7)) # == 1
[1] 1 1 1 1 1 1 1 1
> Ni <- rpois(50, lambda = 4); table(factor(Ni, 0:max(Ni)))

 0  1  2  3  4  5  6  7  8
 0  0  5 19 10  8  4  2  2
> 1 - ppois(10*(15:25), lambda = 100)  # becomes 0 (cancellation)
 [1] 1.233094e-06 1.261664e-08 7.085799e-11 2.252643e-13 4.440892e-16
 [6] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
[11] 0.000000e+00
> ppois(10*(15:25), lambda = 100, lower.tail = FALSE)  # no cancellation
 [1] 1.233094e-06 1.261664e-08 7.085800e-11 2.253110e-13 4.174239e-16
 [6] 4.626179e-19 3.142097e-22 1.337219e-25 3.639328e-29 6.453883e-33
[11] 7.587807e-37
> par(mfrow = c(2, 1))
> x <- seq(-0.01, 5, 0.01)
> plot(x, ppois(x, 1), type = "s", ylab = "F(x)", main = "Poisson(1) CDF")
> |
```




```
>
> #qpois function
> lower<-qpois(0.001, lambda=2.5)
> upper<-qpois(0.999, lambda=2,5)
> n<-seq(lower,upper,1)
> q<-seq(0.001,0.999,0.001)
> dPoisson25 <- data.frame(N=n,
+                          Density=dpois(n, lambda=2.5),
+                          Distribution=ppois(n, lambda=2.5))
> qPoisson25 <- data.frame(Q=q, Quantile=qpois(q, lambda=2.5))
> head(dPoisson25)
  N    Density Distribution
1 0 0.08208500    0.0820850
2 1 0.20521250    0.2872975
3 2 0.25651562    0.5438131
4 3 0.21376302    0.7575761
5 4 0.13360189    0.8911780
6 5 0.06680094    0.9579790
> head(qPoisson25)
      Q Quantile
1 0.001        0
2 0.002        0
3 0.003        0
4 0.004        0
5 0.005        0
6 0.006        0
>
```

Program #18

If there are twelve cars crossing a bridge per minute on average, find the probability of having seventeen or more cars crossing the bridge in a particular minute.

SOURCE CODE

/*Alan Payyappilly*/

Solution

The probability of having sixteen or less cars crossing the bridge in a particular minute is given by the function ppois.

```
> ppois(16, lambda=12)   # lower tail
[1] 0.898709
>
```

Hence the probability of having seventeen or more cars crossing the bridge in a minute is in the upper tail of the probability density function.

OUTPUT

```
> ppois(16, lambda=12, lower=FALSE)   # upper tail
[1] 0.101291
>
```

Answer

If there are twelve cars crossing a bridge per minute on average, the probability of having seventeen or more cars crossing the bridge in a particular minute is 10.1%

Program #19

Explain the Pie Chart, Bar Chart and Line Graph using the given dataset

/*Alan Payyappilly*/

1) **pie chart**

```
#Create data for the graph.
x<-c(21, 62, 10, 53)
labels<-c("London", "New York", "Singapore", "Mumbai")
#Give the chart file a name.
png(file = "city.jpg")
#Plot the chart.
pie(x,labels)
#Save the file.
dev.off()
```

2) **Pie Chart Title and Colors**

```
# Create data for the graph.
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")
# Give the chart file a name. png(file = "city_title_colours.jpg")
# Plot the chart with title and rainbow color pallet.
pie(x, labels, main = "City pie chart", col = rainbow(length(x)))
# Save the file. dev.off()
```

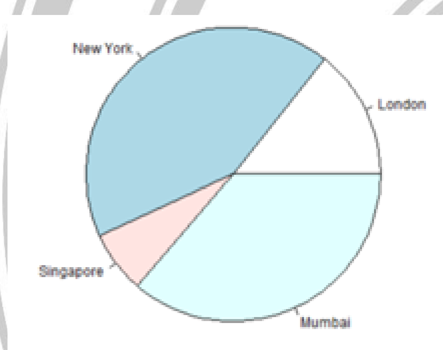
3) **Slice Percentages and Chart Legend**

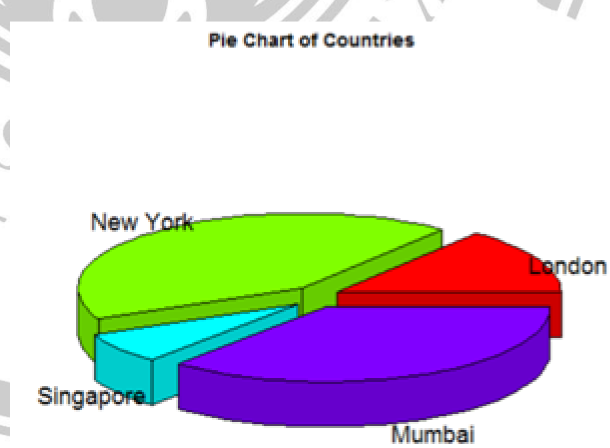
```
#Create data for the graph.
x <- c(21, 62, 10,53)
labels<-c("London","New York","Singapore","Mumbai") piepercent<-
round(100*x/sum(x), 1)
#Give the chart file a name.
png(file = "city_percentage_legends.jpg")
# Plot the chart.
pie(x, labels = piepercent, main = "City pie chart",col = rainbow(length(x)))
legend("topright", c("London","New York","Singapore","Mumbai"), cex = 0.8,
fill = rainbow(length(x)))
#Save the file.
dev.off()
```

4) 3D Pie Chart

```
# Get the library. library(plotrix)
# Create data for the graph.
x<-c(21, 62, 10,53)
lbl <-c("London","New York","Singapore","Mumbai")
# Give the chart file a name. png(file = "3d_pie_chart.jpg")
# Plot the chart.
pie3D(x,labels = lbl,explode = 0.1, main = "Pie Chart of Countries ")
# Save the file.
dev.off()
```

OUTPUT





SOURCE CODE

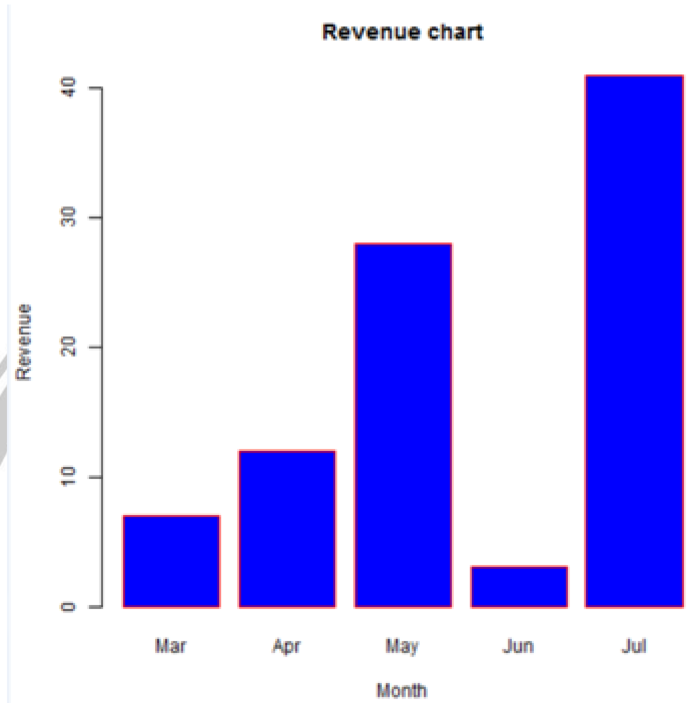
1) Bar Charts

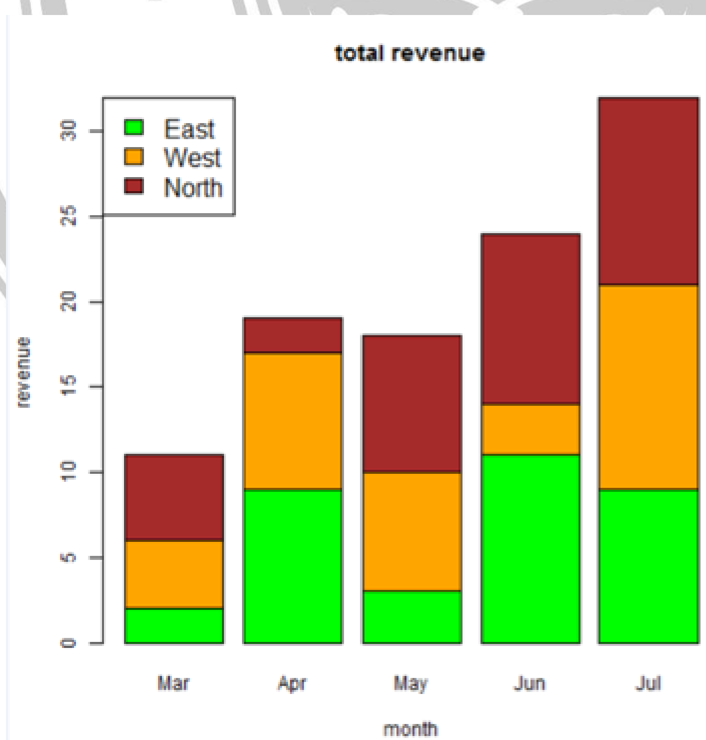
```
#Create the data for the chart.
H<-c(7,12,28,3,41)
M<-c("Mar","Apr","May","Jun","Jul")
#Give the chart file a name
png(file = "barchart_months_revenue.png")
#Plot the bar chart
barplot(H,names.arg=M,xlab="Month",ylab="Revenue",col="blue",main="Revenue chart",border="red")
#Save the file
dev.off()
```

2) Group Bar Chart and Stacked Bar Chart

```
# Create the input vectors.
colors=c("green","orange","brown")
months<-c("Mar","Apr","May","Jun","Jul")
regions<-c("East","West","North")
# Create the matrix of the values.
Values<-matrix(c(2,9,3,11,9,4,8,7,3,12,5,2,8,10,11), nrow = 3, ncol = 5, byrow = TRUE)
#Give the chart file a name
png(file = "barchart_stacked.png")
#Create the bar chart
barplot(Values, main = "total revenue", names.arg = months, xlab = "month", ylab = "revenue", col = colors)
# Add the legend to the chart
legend("topleft", regions, cex = 1.3, fill = colors)
#Save the file.
dev.off()
```

OUTPUT


Revenue chart
Revenue
0
10
20
30
40
Mar
Apr
May
Jun
Jul
Month

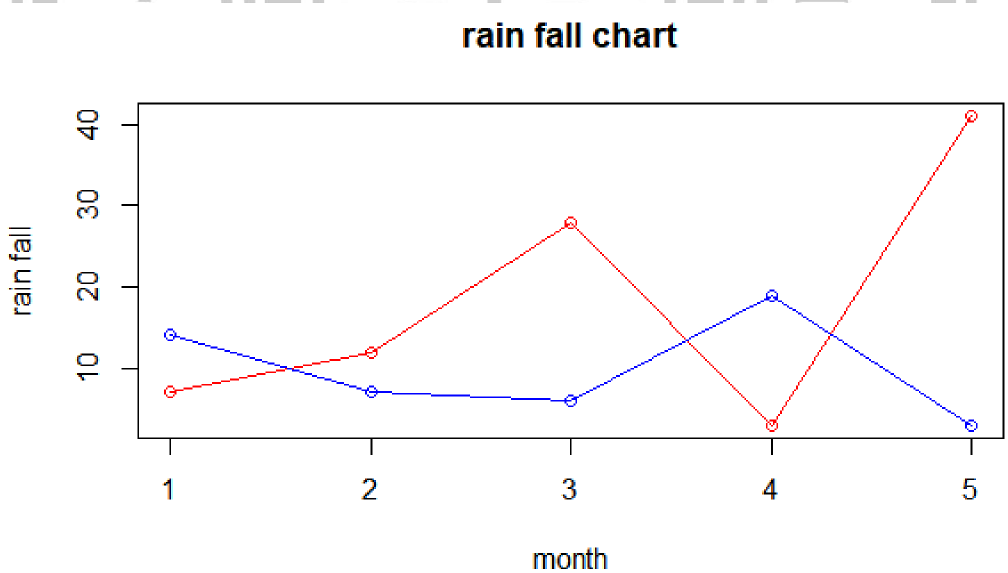

total revenue
East
West
North
revenue
0
5
10
15
20
25
30
Mar
Apr
May
Jun
Jul
month

SOURCE CODE

LINE GRAPH

```
#Create data for the graph.
x<-c(7,12,28,3,41)
y<-c(14,7,6,19,3)

#give the chart file a name.
png(file="line_chart_2_lines.jpg")

#plot the var chart.
plot(x,type="o",col="red",xlab="month",ylab="rain fall", main="rain fall chart")
lines(y,type="o",col="blue")

#save the file.
dev.off()
```

OUTPUT



Program #20

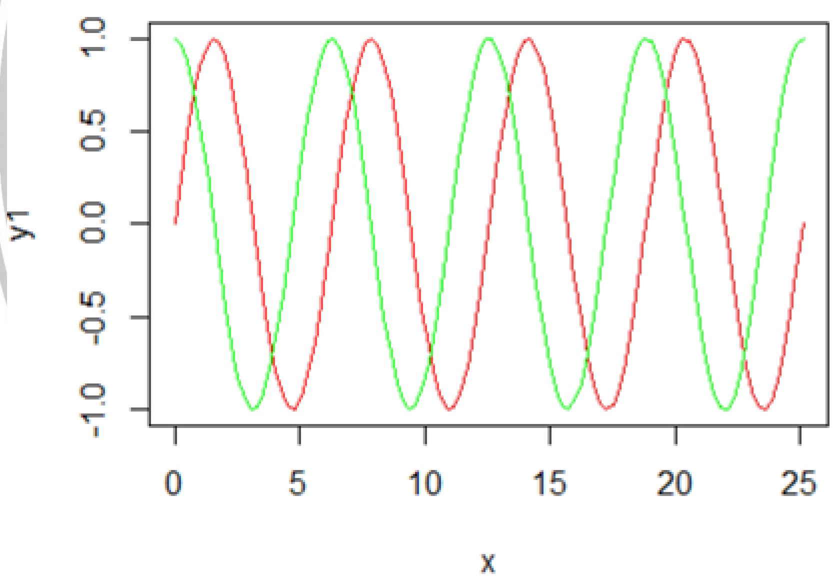
Explain the plot and line functions by constructing the Sine and Cosine wave.

SOURCE CODE

/*Alan Payyappilly*/

```
x <-seq (0,8*pi,length.out =100)
y1 <-sin(x)
y2 <-cos (x)
plot(x,y1,type="l",col="red")
lines(x,y2,col="green")
```

OUTPUT



Program #21

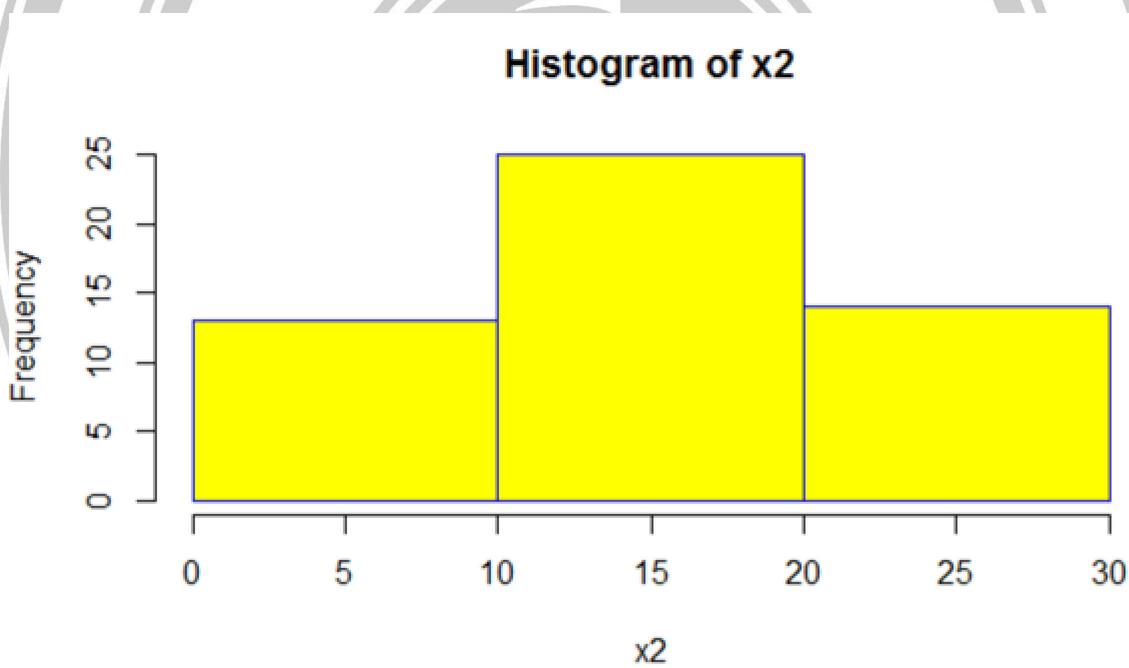
Explain the different features of Histogram using the given dataset.

SOURCE CODE

/*Alan Payyappilly*/

```
x2<-c(1, 1, 5, 5, 5, 5, 5,8, 8, 10, 10, 10, 10, 12, 14, 14, 14, 15, 15, 15, 15, 15, 15, 18, 18, 18, 18, 18, 18, 18, 18, 18, 20,20, 20, 20, 20, 20, 20, 21, 21, 21, 21, 25, 25, 25, 25, 25, 28, 28, 30, 30, 30.)
x2
hist(x2,seq(0,30,by=10),col = "yellow",border = "blue")
```

OUTPUT



Program #22

From a given data set plot a box plot, scatter plot and correl plot.

/*Alan Payyappilly*/

DATA SET

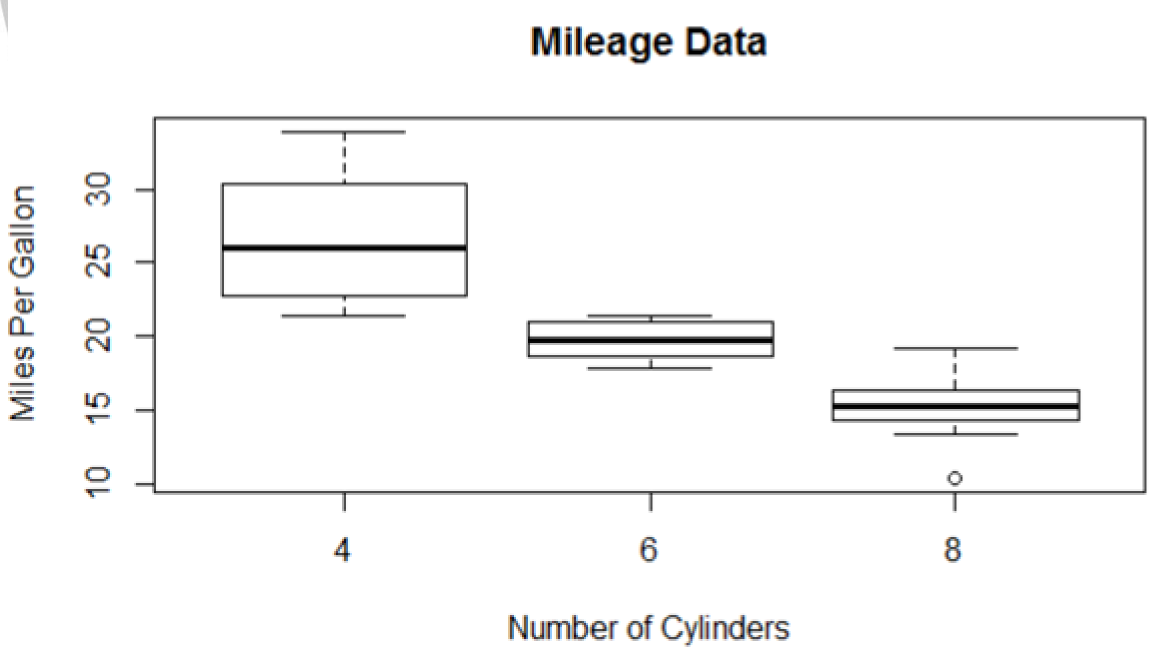
mpg cyl
Mazda RX4 21.0 6
Mazda RX4 Wag 21.0 6
Datsun 710 22.8 4
Hornet 4 Drive 21.4 6
Hornet Sportabout 18.7 8
Valiant 18.1 6

SOURCE CODE

Box Plot

```
boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders", ylab = "Miles Per Gallon", main = "Mileage Data")
```

OUTPUT

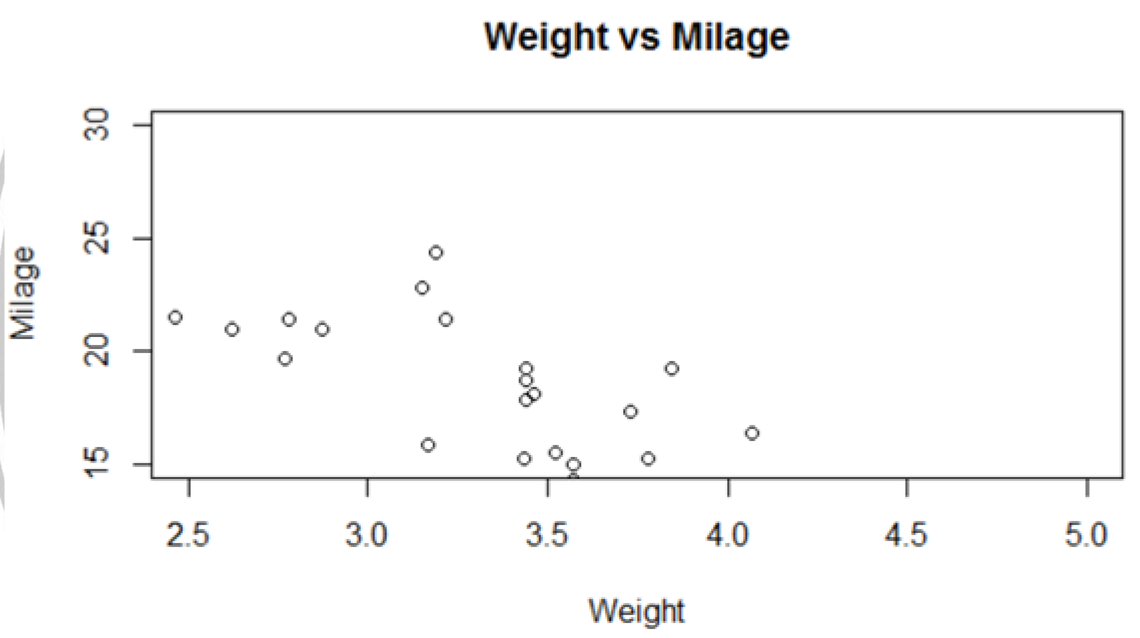


SOURCE CODE

Scatter plot

```
input<- mtcars[,c('wt','mpg')]
print(head(input))
#Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.
plot(x = input$wt,y = input$mpg,
xlab = "Weight", ylab = "Milage", xlim = c(2.5,5), ylim = c(15,30),main =
"WeightvsMilage")
```

OUTPUT

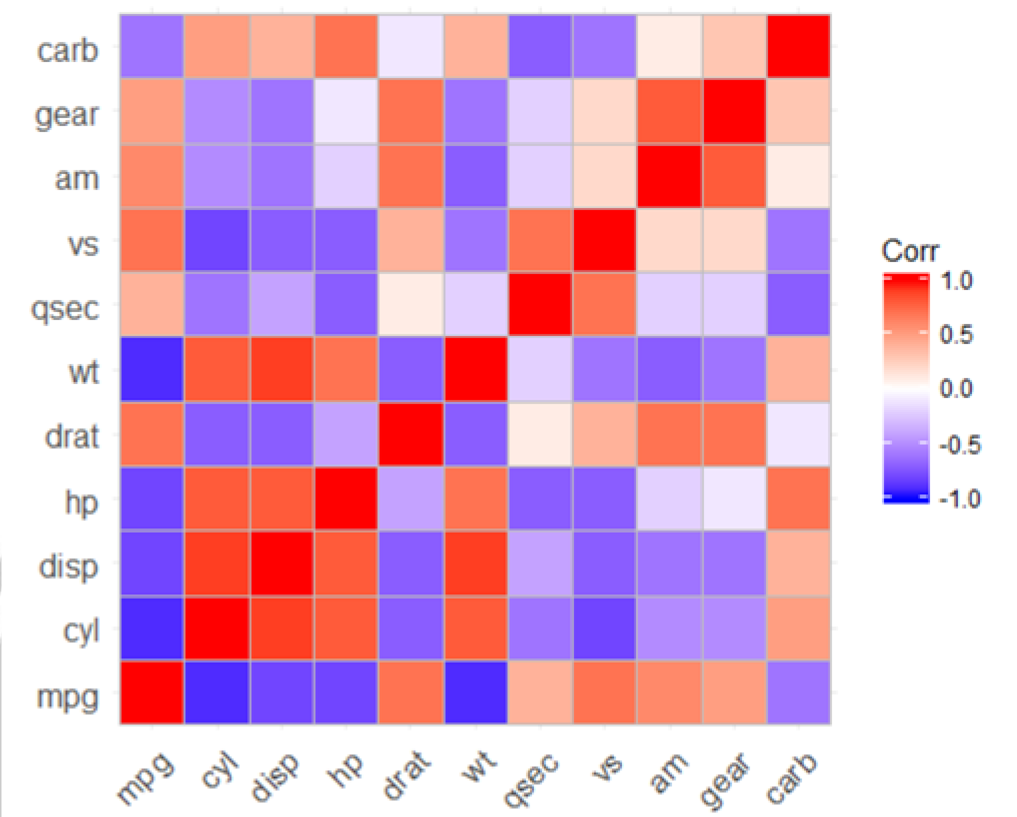


SOURCE CODE

Scatter plot

```
input<- mtcars[,c('wt','mpg')]
print(head(input))
#Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.
plot(x = input$wt,y = input$mpg,
xlab = "Weight", ylab = "Milage", xlim = c(2.5,5), ylim = c(15,30),
main = "Weight vsMilage")
```

OUTPUT


carb
gear
am
vs
qsec
wt
drat
hp
disp
cyl
mpg
mpg
cyl
disp
hp
drat
wt
qsec
vs
am
gear
carb
Corr
1.0
0.5
0.0
-0.5
-1.0

Program #23

Explain the power of ggplot2 using the menu dataset

SOURCE CODE

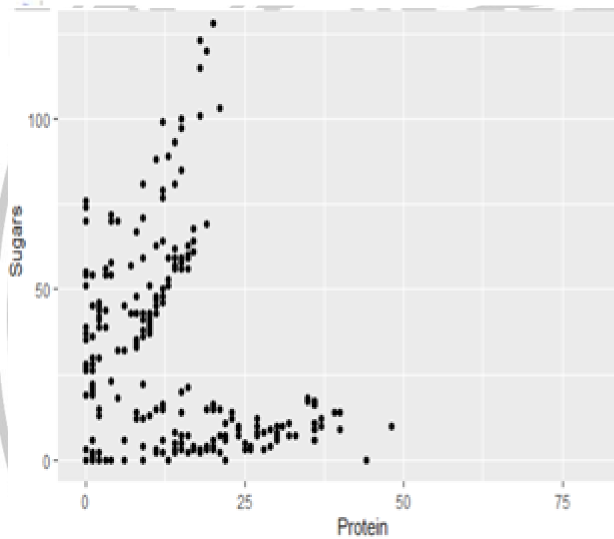
/*Alan Payyappilly*/

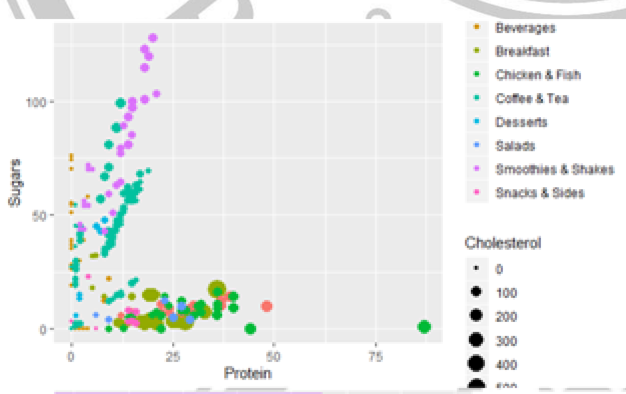
```
menu<-read.csv("D:/Rlab/menu.csv")
menu
summary(menu)
install.packages("ggplot2")
library(ggplot2)
ggplot()
ggplot(data=menu)
ggplot(data=menu) +geom_point(mapping = aes(x=Protein, y=Sugars))

ggplot(data=menu) +geom_point(mapping = aes(x=Protein, y=Sugars, color=Category))
ggplot(data=menu) +geom_point(mapping = aes(x=Protein, y=Sugars, color=Category,
size=Cholesterol))

ggplot(data=menu, fig.height = 5) +geom_point(mapping = aes(x=Protein, y=Sugars,
color=Category, size=Cholesterol , shape=Category)) #//GGPLOT geomlabel&text

ggplot(data=menu) +geom_text(mapping = aes(x=Protein, y=Sugars, color=Category,
label=Item))

ggplot(data=menu) +geom_label(mapping = aes(x=Protein, y=Sugars, color=Category,
label=Item))

#ggplotgeom_bar(one variable)
#1)
ggplot(data=menu) +geom_bar(mapping = aes(x=Category))
#2)
ggplot(data=menu) +geom_bar(mapping = aes(x=Category, color=Category))
#3)
ggplot(data=menu) +geom_bar(mapping = aes(x=Category, fill=Category))
#4)
ggplot(data=menu) +geom_bar(mapping = aes(x=Category, fill=Category),
color="slategrey")
#ggplotgeom_historam
```

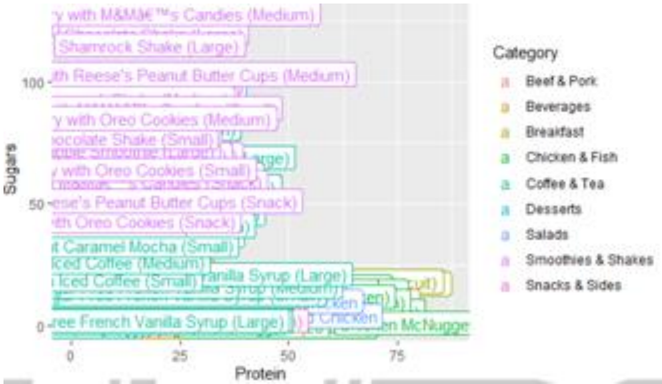
```
#1)
ggplot(data=menu) +geom_histogram(mapping = aes(x=Sugars))

#2)
ggplot(data=menu) +geom_histogram(mapping = aes(x=Sugars), fill="gold",
color="orangered")
#ggplotgeom_boxplot(ONE CONTINUOUS VARIABLE AND ONE CATEGORICAL
VARIABLE)

#1)
ggplot(data=menu) +geom_boxplot(mapping = aes(x=Category, y=Sugars))
#2)
ggplot(data=menu) +geom_boxplot(mapping = aes(x=Category, y=Sugars))
+geom_point(mapping = aes(x=Category, y=Sugars))
#ggplotgeom_violin
#1)
ggplot(data=menu, mapping = aes(x=Category, y=Protein)) +geom_violin()
#2)
ggplot(data=menu, mapping = aes(x=Category, y=Protein)) +geom_violin() +geom_point()
#OTHER GGPLOT2 FUNCTIONS
#1)
p <- ggplot(menu, aes(Category, Sugars)) +

geom_boxplot(color="red", fill="yellow", size=0.75) +geom_point(color="gray25") p
#2)
p + xlab("type of food") +ylab("sugar per serving (g)")

#3)title,labels,subtitle
p_with_text<- p +labs(x="type of food", y="sugar per serving (g)", title="Why McDonalds
food isn't the healthiest option", subtitle="Especially if you want to avoid sugar",
caption = "Figure 1: Distribution of sugar by category.Nice color scheme by the way ;)")
p_with_text
#4)themes
p_with_text + theme_bw()
p_with_text + theme_dark()

#5)FACETING
ggplot(menu, aes(Protein, Sugars, color=Category)) + geom_point() + theme_bw()
ggplot(menu, aes(Protein, Sugars, color=Category)) + geom_point() +
facet_wrap(~Category) + theme_bw()
ggplot(menu, aes(Protein, Sugars)) + geom_point() + facet_wrap(~Category, scales="free")

+ theme_bw()
```

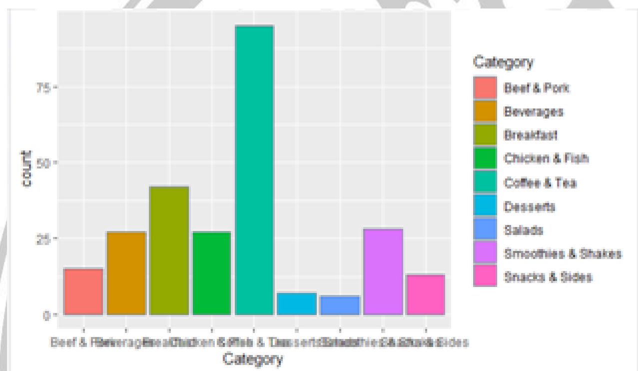
OUTPUT

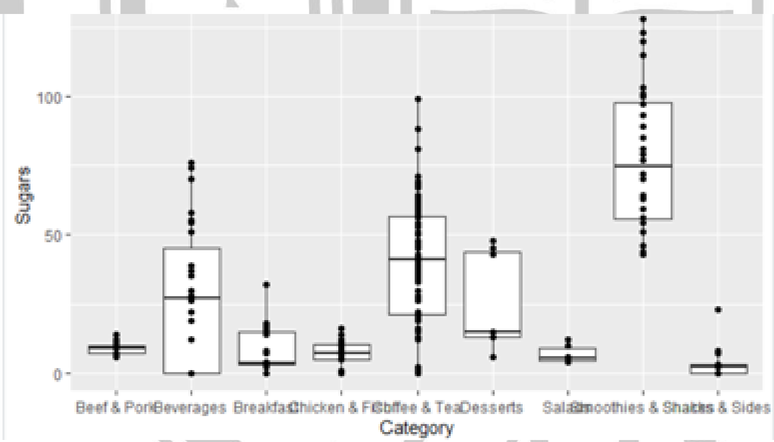
```
 Mean   :0.2038   Mean   : 54.94   Mean   : 18.39                  Mean   : 495.8
 3rd Qu.:0.0000   3rd Qu.: 65.00   3rd Qu.: 21.25                  3rd Qu.: 865.0
 Max.   :2.5000   Max.   :575.00   Max.   :192.00                  Max.   :3600.0

 Sodium....Daily.Value. Carbohydrates    Carbohydrates....Daily.Value.
 Min.   :  0.00         Min.   :  0.00   Min.   : 0.00
 1st Qu.:  4.75         1st Qu.: 30.00   1st Qu.:10.00
 Median :  8.00         Median : 44.00   Median :15.00
 Mean   : 20.68         Mean   : 47.35   Mean   :15.78
 3rd Qu.: 36.25         3rd Qu.: 60.00   3rd Qu.:20.00
 Max.   :150.00         Max.   :141.00   Max.   :47.00

 Dietary.Fiber   Dietary.Fiber....Daily.Value.      Sugars          Protein
 Min.   :0.000   Min.   : 0.000                  Min.   :  0.00   Min.   : 0.00
 1st Qu.:0.000   1st Qu.: 0.000                  1st Qu.:  5.75   1st Qu.: 4.00
 Median :1.000   Median : 5.000                  Median : 17.50   Median :12.00
 Mean   :1.631   Mean   : 6.531                  Mean   : 29.42   Mean   :13.34
 3rd Qu.:3.000   3rd Qu.:10.000                  3rd Qu.: 48.00   3rd Qu.:19.00
 Max.   :7.000   Max.   :28.000                  Max.   :128.00   Max.   :87.00

 Vitamin.A....Daily.Value. Vitamin.C....Daily.Value. Calcium....Daily.Value.
 Min.   :  0.00            Min.   :  0.000           Min.   : 0.00
 1st Qu.:  2.00            1st Qu.:  0.000           1st Qu.: 6.00
 Median :  8.00            Median :  0.000           Median :20.00
 Mean   : 13.43            Mean   :  8.535           Mean   :20.97
 3rd Qu.: 15.00            3rd Qu.:  4.000           3rd Qu.:30.00
 Max.   :170.00            Max.   :240.000           Max.   :70.00

 Iron....Daily.Value.
 Min.   : 0.000
 1st Qu.: 0.000
 Median : 4.000
 Mean   : 7.735
 3rd Qu.:15.000
 Max.   :40.000
```

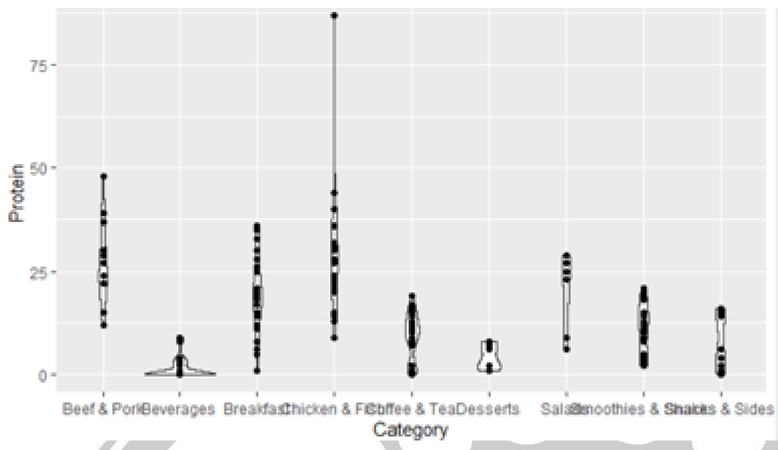


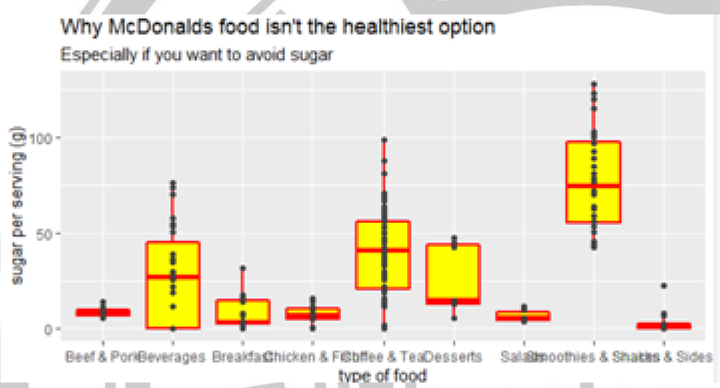


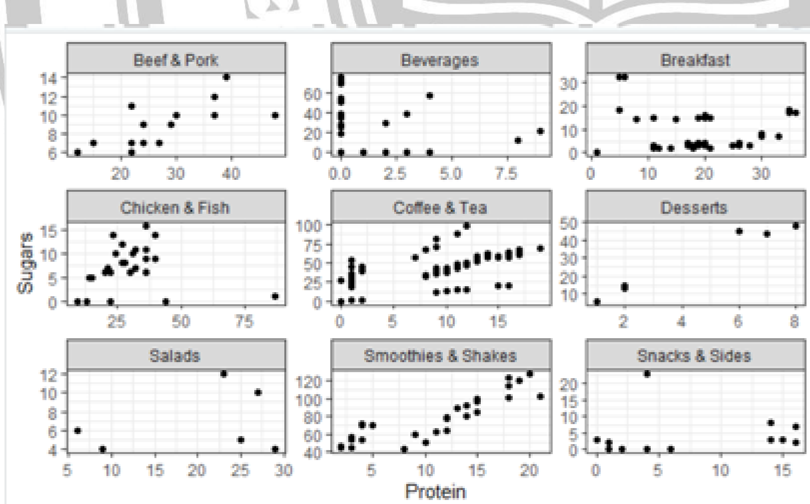

Shamrock Shake (Large)
Sugars
Protein
Category
Beef & Pork
Beverages
Breakfast
Chicken & Fish
Coffee & Tea
Desserts
Salads
Smoothies & Shakes
Snacks & Sides


count
Category
Category
Beef & Pork
Beverages
Breakfast
Chicken & Fish
Coffee & Tea
Desserts
Salads
Smoothies & Shakes
Snacks & Sides


Sugars
Category


Protein
75
50
25
0
Beef & Pork
Beverages
Breakfast
Chicken & Fish
Coffee & Tea
Desserts
Salads
Smoothies & Shakes
Snacks & Sides
Category


Why McDonalds food isn't the healthiest option
Especially if you want to avoid sugar
sugar per serving (g)
100
50
0
Beef & Pork
Beverages
Breakfast
Chicken & Fish
Coffee & Tea
Desserts
Salads
Smoothies & Shakes
Snacks & Sides
type of food


Beef & Pork
Beverages
Breakfast
Chicken & Fish
Coffee & Tea
Desserts
Salads
Smoothies & Shakes
Snacks & Sides
Sugars
Protein

Program #24

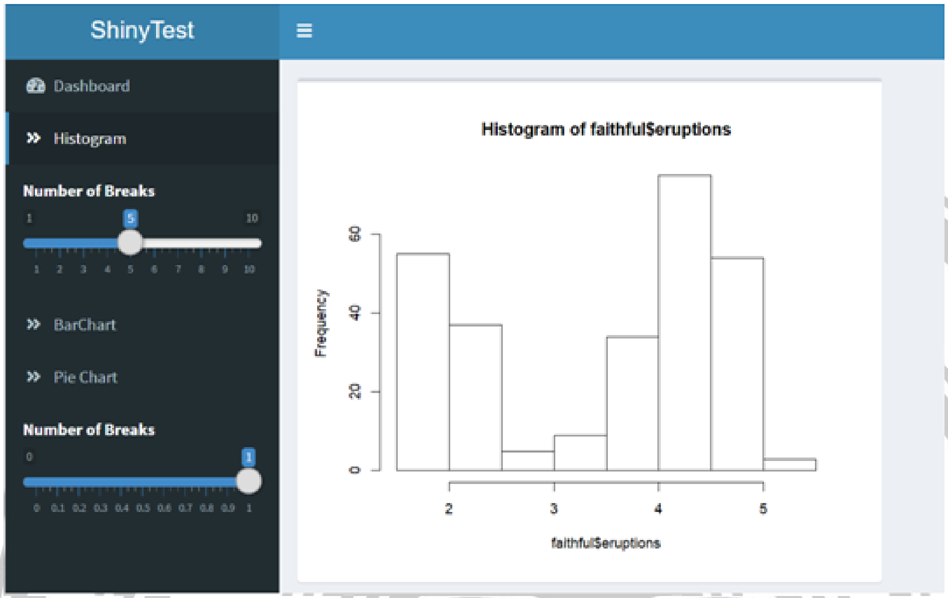
Explain the power of shiny using the Old Faithful Geyser / mtcars dataset

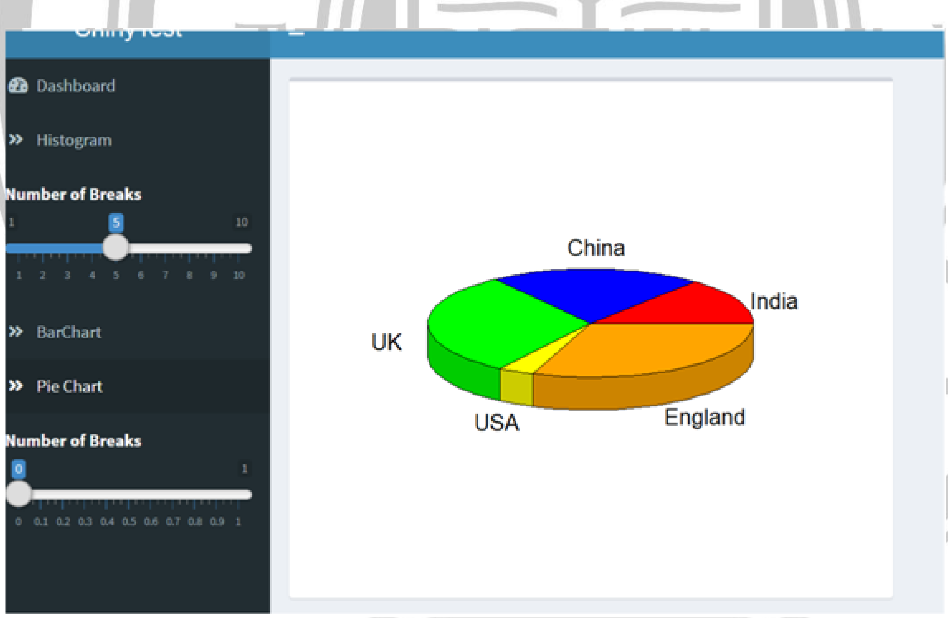
SOURCE CODE

/*Alan Payyappilly*/

```
#server library(shiny) library(shinydashboard) library(plotrix)
shinyServer(function(input,output){

output$histogram<-renderPlot({ hist(faithful$eruptions,breaks = input$bins)
})

output$bar<-renderPlot({ bar2<-tapply(mtcars$am,list(mtcars$gear),mean) barplot(bar2)

})

output$pie<-renderPlot({ c<-c(155,234,340,40,342) p<-
c("India","China","UK","USA","England")

pie3D(c,labels = p ,explode=input$bin,col=c("red","blue","green","yellow","orange")) })

})

#UI
library(shiny)
library(shinydashboard)
shinyUI(
dashboardPage(
dashboardHeader(title =
"ShinyTest"),dashboardSidebar(sidebarMenu(menuItem("Dashboard",tabName =
"dashboard",icon = icon("dashboard")), menuSubItem("Histogram",tabName = "hist"),
sliderInput("bins","Number of Breaks",1,10,5), menuSubItem("BarChart",tabName =
"bar"), menuSubItem("Pie Chart",tabName = "pie"),sliderInput("bin","Number of
Breaks",0,1,10))),dashboardBody(tabItems(tabItem(tabName="dashboard",h1("This is to
Display DataForms")),tabItem(tabName = "hist",fluidRow(box(plotOutput("histogram")))),
tabItem(tabName = "bar",fluidRow(box(plotOutput("bar")))),tabItem(tabName =
"pie",fluidRow(box(plotOutput("pie")))))))
```

OUTPUT


ShinyTest
Dashboard
Histogram
Number of Breaks
BarChart
Pie Chart
Number of Breaks
Histogram of faithful$eruptions
Frequency
faithful$eruptions


Dashboard
Histogram
Number of Breaks
BarChart
Pie Chart
Number of Breaks
China
India
UK
USA
England

Program #25

Explain Data wrangling with "dplyr" package to transform, organize and summarize the given dataset.

/*Alan Payyappilly*/

dplyr is a package which was built for the sole purpose of simplifying the process of manipulating, sorting, summarizing, and joining data frames. These fundamental functions of data transformation that the dplyr package offers includes:

- select() selects variables
- filter() provides basic filtering capabilities
- group_by() groups data by categorical levels
- summarize() summarizes data by functions of choice
- arrange() orders data
- join() joins separate dataframes
- mutate() creates new variables

Packages Utilized

Install.package("dplyr")
library(dplyr)

Following Data Set is used for the following examples:

Division State X1980 X1990 X2000 X2001 X2002 X2003 X2004 X2005 X2006 X2007 X2008 X2009 X2010 X2011

1 6 Alabama 1146713 2275233 4176082 4354794 4444390 4657643 4812479 5164406 5699076 6245031 6832439 6683843 6670517 6592925

2 9 Alaska 377947 828051 1183499 1229036 1284854 1326226 1354846 1442269 1529645 1634316 1918375 2007319 2084019 2201270

3 8 Arizona 949753 2258660 4288739 4846105 5395814 5892227 6071785 6579957 7130341 7815720 8403221 8726755 8482552 8340211

4 7 Arkansas 666949 1404545 2380331 2505179 2822877 2923401 3109644 3546999 3808011 3997701 4156368 4240839 4459910 4578136

5 9 California 9172158 21485782 38129479 42908787 46265544 47983402 49215866 50918654 53436103 57352599 61570555 60080929 58248662 57526835

```
## 6       8   Colorado 1243049  2451833  4401010  4758173  5151003  5551506  5666191
5994440  6368289  6579053  7338766  7187267  7429302  7409462
```

select( ) function

Objective: Reduce dataframe size to only desired variables for current task

Description: When working with a sizable dataframe, often we desire to only assess specific variables. The select() function allows you to select and/or rename variables.

Function: select(data, ...)
Same as: data %>% select(...)
Arguments:
data: data frame
...: call variables by name or by function

Special functions:
starts_with(x, ignore.case = TRUE): names starts with x
ends_with(x, ignore.case = TRUE): names ends in x
contains(x, ignore.case = TRUE): selects all variables whose name contains x
matches(x, ignore.case = TRUE): selects all variables whose name matches the regular expression x

Example: To assess the 5 most recent years worth of expenditure data. Applying the select() function we can *select* only the variables of concern.

SOURCE CODE

```
sub.exp <- expenditures %>% select(Division, State, X2007:X2011)
head(sub.exp)  # for brevity only display first 6 rows
```

OUTPUT

```
##   Division      State    X2007    X2008    X2009    X2010    X2011
## 1        6    Alabama  6245031  6832439  6683843  6670517  6592925
## 2        9     Alaska  1634316  1918375  2007319  2084019  2201270
## 3        8    Arizona  7815720  8403221  8726755  8482552  8340211
## 4        7   Arkansas  3997701  4156368  4240839  4459910  4578136
## 5        9 California 57352599 61570555 60080929 58248662 57526835
## 6        8   Colorado  6579053  7338766  7187267  7429302  7409462
```

filter( ) function:

Objective: Reduce rows/observations with matching conditions

Description: Filtering data is a common task to identify/select observations in which a particular variable matches a specific value/condition. The filter() function provides this capability.

Function: filter(data, ...)
Same as: data %>% filter(...)

Arguments:
 data: data frame
 ...: conditions to be met

Example: Continuing with our **sub.exp** dataframe which includes only the recent 5 years worth of expenditures, we can filter by *Division*:

SOURCE CODE

```
sub.exp %>% filter(Division == 3)
```

OUTPUT

```
##   Division     State   X2007    X2008    X2009    X2010    X2011
## 1        3  Illinois 20326591 21874484 23495271 24695773 24554467
## 2        3   Indiana  9497077  9281709  9680895  9921243  9687949
## 3        3  Michigan 17013259 17053521 17217584 17227515 16786444
## 4        3      Ohio 18251361 18892374 19387318 19801670 19988921
## 5        3 Wisconsin  9029660  9366134  9696228  9966244 10333016
```

group_by( ) function:

Objective: Group data by categorical variables

Description: Often, observations are nested within groups or categories and our goals is to perform statistical analysis both at the observation level and also at the group level. The group_by() function allows us to create these categorical groupings.

Function: group_by(data, ...)
Same as: data %>% group_by(...)

Arguments:

data: data frame
...: variables to group_by

*Use ungroup(x) to remove groups

SOURCE CODE

```
group.exp <- sub.exp %>% group_by(Division)
head(group.exp)
```

OUTPUT

```
## Source: local data frame [6 x 7]
## Groups: Division
##
##   Division     State    X2007    X2008    X2009    X2010    X2011
## 1        6   Alabama  6245031  6832439  6683843  6670517  6592925
## 2        9    Alaska  1634316  1918375  2007319  2084019  2201270
## 3        8   Arizona  7815720  8403221  8726755  8482552  8340211
## 4        7  Arkansas  3997701  4156368  4240839  4459910  4578136
## 5        9 California 57352599 61570555 60080929 58248662 57526835
## 6        8  Colorado  6579053  7338766  7187267  7429302  7409462
```

summarise( ) function:

Objective: Perform summary statistics on variables

Description: Obviously the goal of all this data wrangling is to be able to perform statistical analysis on our data. The summarise() function allows us to perform the majority of the initial summary statistics when performing exploratory data analysis.

Function: summarise(data, ...)
Same as: data %>% summarise(...)

Arguments:
data: data frame
...: Name-value pairs of summary functions like min(), mean(), max() etc.

Examples: Lets get the mean expenditure value across all states in 2011

SOURCE CODE

```
sub.exp %>% summarise(Mean_2011 = mean(X2011))
```

OUTPUT

```
##   Mean_2011
## 1  10513678
```

arrange( ) function:

Objective: Order variable values

Description: Often, we desire to view observations in rank order for a particular variable(s). The arrange() function allows us to order data by variables in accending or descending order.

Function: arrange(data, ...)
Same as: data %>% arrange(...)

Arguments:
- data: data frame
- ...: Variable(s) to order

SOURCE CODE

```
sub.exp %>%
    group_by(Division)%>%
    summarise(Mean_2010 = mean(X2010, na.rm=TRUE),
              Mean_2011 = mean(X2011, na.rm=TRUE)) %>%
    arrange(Mean_2011)
```

OUTPUT

```
## Source: local data frame [9 x 3]
##
##   Division Mean_2010 Mean_2011
## 1        8   3894003   3882159
## 2        4   4672332   4672687
## 3        1   5121003   5222277
## 4        6   6161967   6267490
## 5        5  10975194  11023526
## 6        7  14916843  15000139
## 7        9  15540681  15468173
## 8        3  16322489  16270159
```

```
## 9       2  32415457  32877923
```

<u>join( ) functions</u>:

Objective: Join two datasets together

Description: Often we have separate dataframes that can have common and differing variables for similar observations and we wish to *join* these dataframes together. The multiple xxx-join() functions provide multiple ways to join dataframes.
Description: Join two datasets

Function:

```
inner_join(x, y, by = NULL)
left_join(x, y, by = NULL)
right_join(x, y, by = NULL)
full_join(x, y, by = NULL)
semi_join(x, y, by = NULL)
anti_join(x, y, by = NULL)
```

Arguments:

x,y: data frames to join

by: a character vector of variables to join by. If NULL, the default, join will do a natural join, using all variables with common names across the two tables.

Example: The following is another dataframe which provides inflation adjustment factors for base-year 2012 dollars

```
##    Year  Annual Inflation
## 28 2007 207.342 0.9030811
## 29 2008 215.303 0.9377553
## 30 2009 214.537 0.9344190
## 31 2010 218.056 0.9497461
## 32 2011 224.939 0.9797251
## 33 2012 229.594 1.0000000
```

SOURCE CODE

```
long.exp <- sub.exp %>%
gather(Year, Expenditure, X2007:X2011) %>%        # turn to long format
separate(Year, into=c("x", "Year"), sep="X") %>%   # separate "X" from year value
select(-x)
long.exp$Year <- as.numeric(long.exp$ Year)  # convert from character to numeric

head(long.exp)
```

OUTPUT

```
##   Division      State Year Expenditure
## 1        6    Alabama 2007     6245031
## 2        9     Alaska 2007     1634316
## 3        8    Arizona 2007     7815720
## 4        7   Arkansas 2007     3997701
## 5        9 California 2007    57352599
## 6        8   Colorado 2007     6579053
```

mutate( ) function:

Objective: Creates new variables

Description: Often we want to create a new variable that is a function of the current variables in our dataframe or even just add a new variable. The mutate() function allows us to add new variables while preserving the existing variables.

Function: mutate(data, ...)
Same as: data %>% mutate(...)

Arguments:
data: data frame
...: Expression(s)

Examples: If we go back to our previous join.exp dataframe, remember that we joined inflation rates to our non-inflation adjusted expenditures for public schools. The dataframe looks like:

```
##   Division      State Year Expenditure  Annual Inflation
## 1        6    Alabama 2007     6245031 207.342 0.9030811
## 2        9     Alaska 2007     1634316 207.342 0.9030811
## 3        8    Arizona 2007     7815720 207.342 0.9030811
## 4        7   Arkansas 2007     3997701 207.342 0.9030811
## 5        9 California 2007    57352599 207.342 0.9030811
## 6        8   Colorado 2007     6579053 207.342 0.9030811
```

If we wanted to adjust our annual expenditures for inflation we can use mutate() to create a new inflation adjusted cost variable which we'll name *inflation_adj*:

SOURCE CODE

```
inflation_adj <- join.exp %>% mutate(Adj_Exp = Expenditure/Inflation)

head(inflation_adj)
```

OUTPUT

```
##   Division      State Year Expenditure  Annual Inflation  Adj_Exp
## 1        6    Alabama 2007     6245031 207.342 0.9030811  6915249
## 2        9     Alaska 2007     1634316 207.342 0.9030811  1809711
## 3        8    Arizona 2007     7815720 207.342 0.9030811  8654505
## 4        7   Arkansas 2007     3997701 207.342 0.9030811  4426735
## 5        9 California 2007    57352599 207.342 0.9030811 63507696
## 6        8   Colorado 2007     6579053 207.342 0.9030811  7285119
```

Program #26

Apriori Algorithm :- Market Basket Analysis in R Association Rule Mining

SOURCE CODE

/*Alan Payyappilly*/

Apriori algorithm is use in association rule learning and in frequent item set mining which is deployed over a transactional database. It is extensively used for finding out the various frequent items within a database and then extending it to a large set of items provided those items appear frequently in the database. The apriori algorithm in r is used for determining the association rule in a database that specifies the general trend in a database.

```
> library(arules)
Loading required package: Matrix

Attaching package: 'arules'

The following objects are masked from 'package:base':

    abbreviate, write

Warning message:
package 'arules' was built under R version 4.0.2
```

```
> library(arulesViz)
Loading required package: grid
Registered S3 method overwritten by 'seriation':
  method         from
  reorder.hclust gclus
Warning message:
package 'arulesViz' was built under R version 4.0.2
```

```
> data("Groceries")
> 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
            2513             1903             1809             1715
          yogurt          (Other)
            1372            34055

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

```
> apriori(Groceries)
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target   ext
        0.8    0.1    1 none FALSE            TRUE       5     0.1      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: 983

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [8 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 done [0.00s].
writing ... [0 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
set of 0 rules
```

Rule1

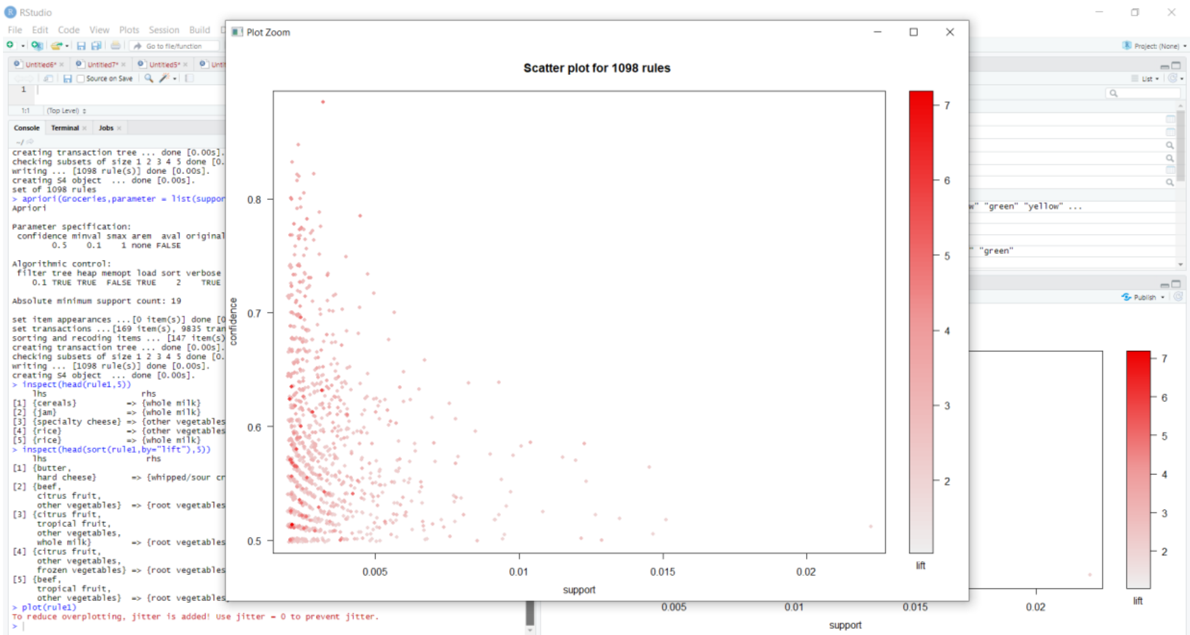
```
> apriori(Groceries,parameter = list(support=0.002,confidence=0.5))->rule1
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target   ext
        0.5    0.1    1 none FALSE            TRUE       5   0.002      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: 19

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [147 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 done [0.01s].
writing ... [1098 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
> inspect(head(rule1,5))
    lhs                   rhs                support     confidence coverage    lift     count
[1] {cereals}          => {whole milk}       0.003660397 0.6428571  0.005693950 2.515917 36
[2] {jam}              => {whole milk}       0.002948653 0.5471698  0.005388917 2.141431 29
[3] {specialty cheese} => {other vegetables} 0.004270463 0.5000000  0.008540925 2.584078 42
[4] {rice}             => {other vegetables} 0.003965430 0.5200000  0.007625826 2.687441 39
[5] {rice}             => {whole milk}       0.004677173 0.6133333  0.007625826 2.400371 46
```



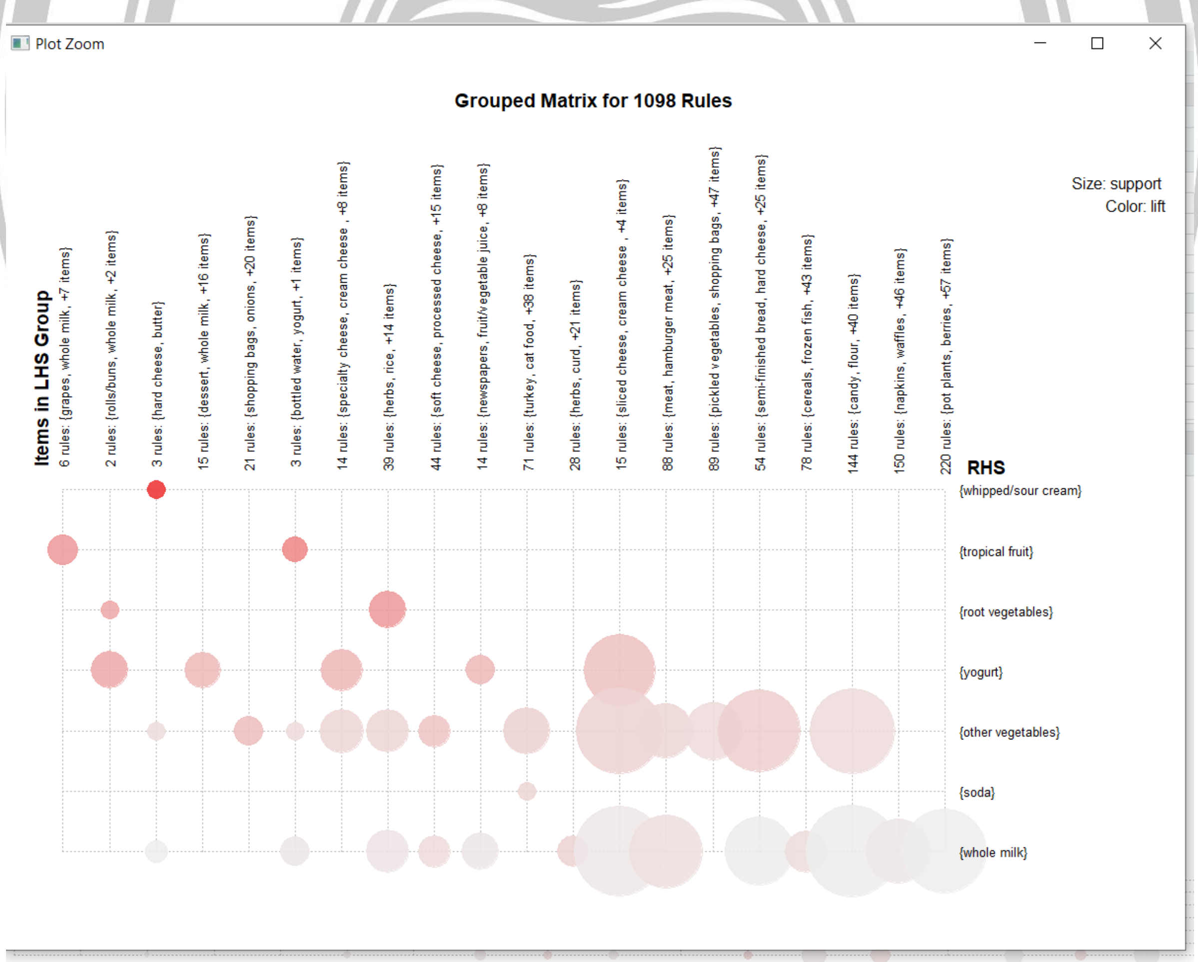
```
> inspect(head(rule1,5))
    lhs                   rhs                support     confidence coverage    lift     count
[1] {cereals}          => {whole milk}       0.003660397 0.6428571  0.005693950 2.515917 36
[2] {jam}              => {whole milk}       0.002948653 0.5471698  0.005388917 2.141431 29
[3] {specialty cheese} => {other vegetables} 0.004270463 0.5000000  0.008540925 2.584078 42
[4] {rice}             => {other vegetables} 0.003965430 0.5200000  0.007625826 2.687441 39
[5] {rice}             => {whole milk}       0.004677173 0.6133333  0.007625826 2.400371 46
```



```
> inspect(head(sort(rule1,by="lift"),5))
    lhs                   rhs                      support confidence    coverage     lift count
[1] {butter,
     hard cheese}      => {whipped/sour cream} 0.002033554  0.5128205 0.003965430 7.154028    20
[2] {beef,
     citrus fruit,
     other vegetables} => {root vegetables}    0.002135231  0.6363636 0.003355363 5.838280    21
[3] {citrus fruit,
     tropical fruit,
     other vegetables,
     whole milk}       => {root vegetables}    0.003152008  0.6326531 0.004982206 5.804238    31
[4] {citrus fruit,
     other vegetables,
     frozen vegetables} => {root vegetables}   0.002033554  0.6250000 0.003253686 5.734025    20
[5] {beef,
     tropical fruit,
     other vegetables} => {root vegetables}    0.002745297  0.6136364 0.004473818 5.629770    27
> plot(rule1)
To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



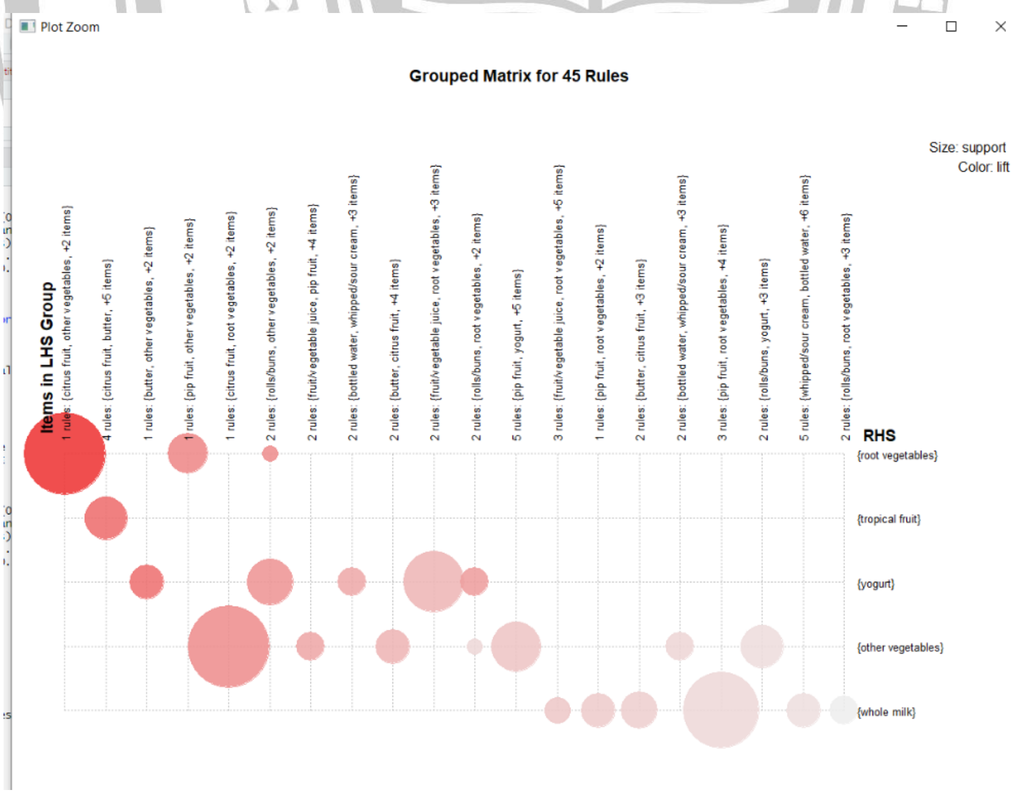

```
> plot(rule1,method = "grouped")
```



Rule2

```
> plot(rule2,method = "grouped")
```

```
> apriori(Groceries,parameter = list(support=0.002,confidence=0.5,minlen=5)) ->rule2
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support
        0.5    0.1    1 none FALSE            TRUE       5   0.002
 minlen maxlen target  ext
      5     10  rules TRUE

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

Absolute minimum support count: 19

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
sorting and recoding items ... [147 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 done [0.01s].
writing ... [45 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
> inspect(head(rule2,4))
    lhs                    rhs                    support confidence    coverage     lift count
[1] {tropical fruit,
     other vegetables,
     butter,
     yogurt}            => {whole milk}       0.002338587  0.7666667 0.003050330 3.000464    23
[2] {tropical fruit,
     whole milk,
     butter,
     yogurt}            => {other vegetables} 0.002338587  0.6969697 0.003355363 3.602048    23
[3] {tropical fruit,
     other vegetables,
     whole milk,
     butter}            => {yogurt}           0.002338587  0.6969697 0.003355363 4.996135    23
[4] {other vegetables,
     whole milk,
     butter,
     yogurt}            => {tropical fruit}   0.002338587  0.5348837 0.004372140 5.097463    23
```



Rule3

```
> apriori(Groceries,parameter = list(support=0.007,confidence=0.6)) ->rule3
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support
        0.6    0.1    1 none FALSE            TRUE       5   0.007
 minlen maxlen target  ext
      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: 68

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
sorting and recoding items ... [104 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [4 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
> inspect(head(rule3,4))
    lhs                     rhs              support confidence   coverage     lift count
[1] {root vegetables,
     butter}             => {whole milk} 0.008235892  0.6377953 0.01291307 2.496107    81
[2] {butter,
     yogurt}             => {whole milk} 0.009354347  0.6388889 0.01464159 2.500387    92
[3] {tropical fruit,
     other vegetables,
     yogurt}             => {whole milk} 0.007625826  0.6198347 0.01230300 2.425816    75
[4] {root vegetables,
     other vegetables,
     yogurt}             => {whole milk} 0.007829181  0.6062992 0.01291307 2.372842    77
> plot(rule3,method = "grouped")
>
```

Program #27

Rule Based Algorithm in R using mushrooms dataset

SOURCE CODE

/*Alan Payyappilly*/

```
mushrooms <- read.csv("C:/Users/dewdrops/Downloads/mushrooms.csv",
stringsAsFactors = TRUE)
str(mushrooms)
```

OUTPUT

```
'data.frame':   8124 obs. of  23 variables:
 $ cap.shape               : Factor w/ 6 levels "b","c","f","k",..: 6 6 1 6 6 6 1 1 6 1 ...
 $ cap.surface             : Factor w/ 4 levels "f","g","s","y": 3 3 3 4 3 4 3 4 4 3 ...
 $ cap.color               : Factor w/ 10 levels "b","c","e","g",..: 5 10 9 9 4 10 9 9 9 10 ...
 $ bruises.3F              : Factor w/ 2 levels "f","t": 2 2 2 2 1 2 2 2 2 2 ...
 $ odor                    : Factor w/ 9 levels "a","c","f","l",..: 7 1 4 7 6 1 1 4 7 1 ...
 $ gill.attachment         : Factor w/ 2 levels "a","f": 2 2 2 2 2 2 2 2 2 2 ...
 $ gill.spacing            : Factor w/ 2 levels "c","w": 1 1 1 1 2 1 1 1 1 1 ...
 $ gill.size               : Factor w/ 2 levels "b","n": 2 1 1 2 1 1 1 1 2 1 ...
 $ gill.color              : Factor w/ 12 levels "b","e","g","h",..: 5 5 6 6 5 6 3 6 8 3 ...
 $ stalk.shape             : Factor w/ 2 levels "e","t": 1 1 1 1 2 1 1 1 1 1 ...
 $ stalk.root              : Factor w/ 5 levels "","b","c","e",..: 4 3 3 4 4 3 3 3 4 3 ...
 $ stalk.surface.above.ring: Factor w/ 4 levels "f","k","s","y": 3 3 3 3 3 3 3 3 3 3 ...
 $ stalk.surface.below.ring: Factor w/ 4 levels "f","k","s","y": 3 3 3 3 3 3 3 3 3 3 ...
 $ stalk.color.above.ring  : Factor w/ 9 levels "b","c","e","g",..: 8 8 8 8 8 8 8 8 8 8 ...
 $ stalk.color.below.ring  : Factor w/ 9 levels "b","c","e","g",..: 8 8 8 8 8 8 8 8 8 8 ...
 $ veil.type               : Factor w/ 1 level "p": 1 1 1 1 1 1 1 1 1 1 ...
 $ veil.color              : Factor w/ 4 levels "n","o","w","y": 3 3 3 3 3 3 3 3 3 3 ...
 $ ring.number             : Factor w/ 3 levels "n","o","t": 2 2 2 2 2 2 2 2 2 2 ...
 $ ring.type               : Factor w/ 5 levels "e","f","l","n",..: 5 5 5 5 1 5 5 5 5 5 ...
 $ spore.print.color       : Factor w/ 9 levels "b","h","k","n",..: 3 4 4 3 4 3 3 4 3 3 ...
```

SOURCE CODE

```
mushrooms$veil_type <- NULL
table(mushrooms$class)
```

OUTPUT

```
   e    p
4208 3916
```

SOURCE CODE

```
#install.packages("OneR")
library(OneR)
```

```
mushrooms_1R <- OneR(class ~ ., data = mushrooms)
mushrooms_1R
```

OUTPUT

```
Call:
OneR.formula(formula = class ~ ., data = mushrooms)

Rules:
If odor = a then class = e
If odor = c then class = p
If odor = f then class = p
If odor = l then class = e
If odor = m then class = p
If odor = n then class = e
If odor = p then class = p
If odor = s then class = p
If odor = y then class = p

Accuracy:
8004 of 8124 instances classified correctly (98.52%)
```

SOURCE CODE

```
summary(mushrooms_1R)
```

OUTPUT

```
Call:
OneR.formula(formula = class ~ ., data = mushrooms)

Rules:
If odor = a then class = e
If odor = c then class = p
If odor = f then class = p
If odor = l then class = e
If odor = m then class = p
If odor = n then class = e
If odor = p then class = p
If odor = s then class = p
If odor = y then class = p

Accuracy:
8004 of 8124 instances classified correctly (98.52%)

Contingency table:
     odor
class     a     c      f     l    m      n     p     s     y  Sum
  e   * 400     0      0 * 400    0 * 3408     0     0     0 4208
  p       0 * 192 * 2160     0 * 36    120 * 256 * 576 * 576 3916
  Sum   400   192   2160   400   36   3528   256   576   576 8124
---
Maximum in each column: '*'

Pearson's Chi-squared test:
X-squared = 7659.7, df = 8, p-value < 2.2e-16
```

Program #28

Naïve Bayes classification in R using Admission into graduate school dataset.

SOURCE CODE

/*Alan Payyappilly*/

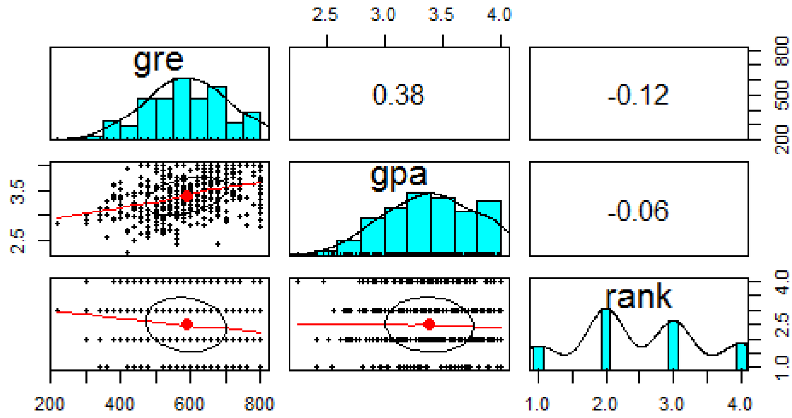
Naïve Bayes classification in R

Naïve Bayes classification is a kind of simple probabilistic classification methods based on Bayes' theorem with the assumption of independence between features. The model is trained on training dataset to make predictions by predict() function.

```
> library(naivebayes)
naivebayes 0.9.7 loaded
Warning message:
package 'naivebayes' was built under R version 4.0.2
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal,
    union

Warning message:
package 'dplyr' was built under R version 4.0.2
> library(ggplot2)
Warning message:
package 'ggplot2' was built under R version 4.0.2
> library(psych)

Attaching package: 'psych'

The following objects are masked from 'package:ggplot2':

    %+%, alpha

Warning message:
package 'psych' was built under R version 4.0.2
```

```
> data<-read.csv(file.choose(),header=T)
> str(data)
'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 ...
> str(data)
'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 ...
> xtabs( ~admit+rank,data=data)
     rank
admit  1  2  3  4
    0 28 97 93 55
    1 33 54 28 12
> data$rank<-as.factor(data$rank)
> data$admit<-as.factor(data$admit)
> str(data)
'data.frame':   400 obs. of  4 variables:
 $ admit: Factor w/ 2 levels "0","1": 1 2 2 2 1 2 2 1 2 1 ...
 $ 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 : Factor w/ 4 levels "1","2","3","4": 3 3 1 4 4 2 1 2 3 2 ...
```

pairs.panels(data[-1])

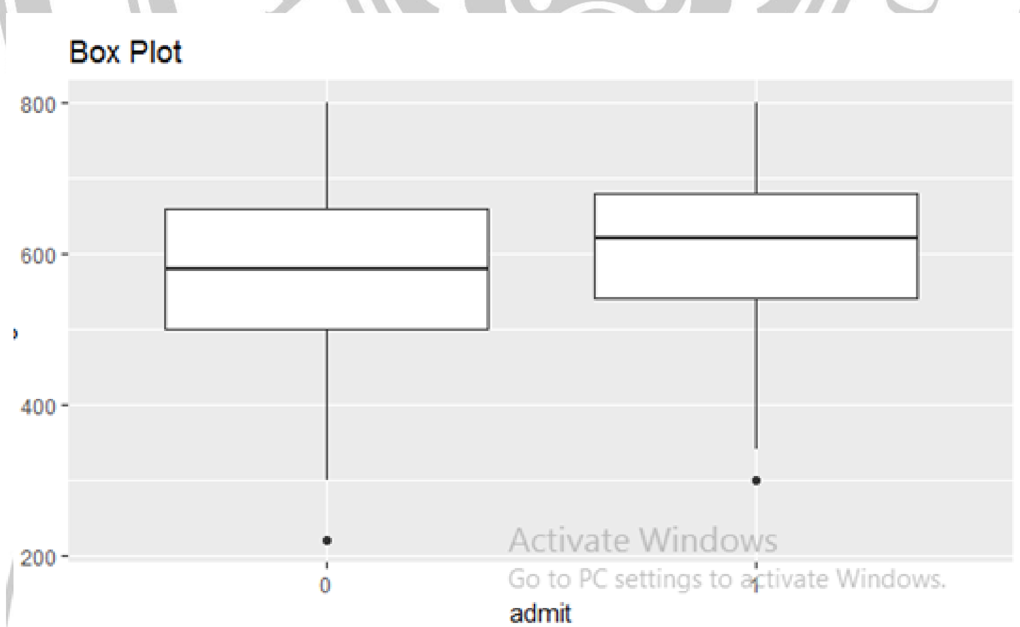


data%>%

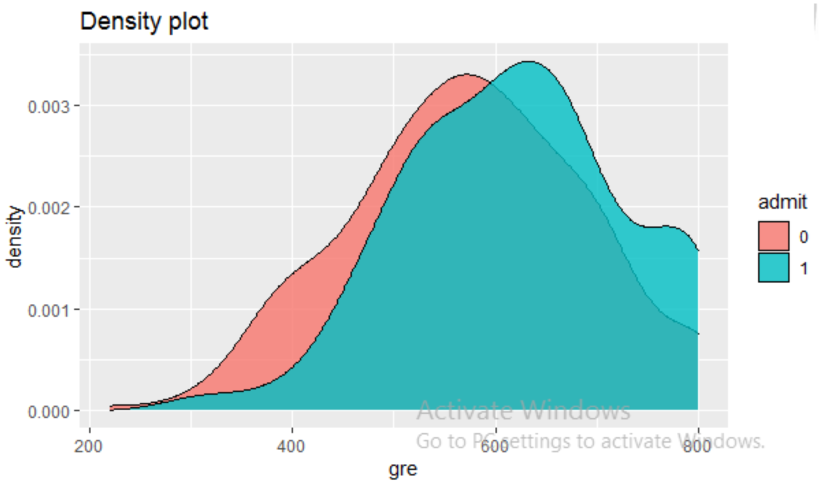
+ ggplot(aes(x=admit,y=gre,fil=admit))+

+ geom_boxplot()+

+ ggtitle("Box Plot")



data%>% ggplot(aes(x=gre,fill=admit))+geom_density(alpha=0.8,color='black')+ggtitle("Density plot")




```
> set.seed(1234)
> ind<-sample(2,nrow(data),replace=T,prob=c(0.8,0.2))
> train<-data[ind==1,]
> test<-data[ind==2,]
> view(train)
> model<-naive_bayes(admit~.,data=train)
> model

=========== Naive Bayes ============

 Call:
naive_bayes.formula(formula = admit ~ ., data = train)

------------------------------------

Laplace smoothing: 0

------------------------------------

 A priori probabilities:

        0         1 
0.6861538 0.3138462 

------------------------------------
```

```
Tables:

-------------------------------------
 ::: gre (Gaussian)
-------------------------------------

gre           0        1
  mean 578.6547 622.9412
  sd   116.3250 110.9240

-------------------------------------
 ::: gpa (Gaussian)
-------------------------------------

gpa            0         1
  mean 3.3552466 3.5336275
  sd   0.3714542 0.3457057

-------------------------------------
 ::: rank (Categorical)
-------------------------------------

rank          0          1
   1 0.10313901 0.24509804
   2 0.36771300 0.42156863
   3 0.33183857 0.24509804
   4 0.19730942 0.08823529

-------------------------------------
```

```
> train %>%
+ filter(admit=="0")%>%
+ summarise(mean(gre),sd(gre))
  mean(gre) sd(gre)
1  578.6547 116.325
> #predict
> p<-predict(model,train,type='prob')
Warning message:
predict.naive_bayes(): more features in the newdata are provided as there are probabili
ty tables in the object. Calculation is performed based on features to be found in the
 tables.
> head(cbind(p,train))
          0         1 admit gre
1 0.8449088 0.1550912     0 380
2 0.6214983 0.3785017     1 660
3 0.2082304 0.7917696     1 800
4 0.8501030 0.1498970     1 640
6 0.6917580 0.3082420     1 760
7 0.6720365 0.3279635     1 560
   gpa rank
1 3.61    3
2 3.67    3
3 4.00    1
4 3.19    4
6 3.00    2
7 2.98    1
```

```
> #confusion matrix -train data
> p1<-predict(model,train)
Warning message:
predict.naive_bayes(): more features in the newdata are provided as there are probabili
ty tables in the object. Calculation is performed based on features to be found in the
 tables.
> (tab1<-table(p1,train$admit))

p1    0   1
  0 196  69
  1  27  33
> 1-sum(diag(tab1))/sum(tab1)
[1] 0.2953846
> #confusion matrix-test data
> p2<-predict(model,test)
Warning message:
predict.naive_bayes(): more features in the newdata are provided as there are probabili
ty tables in the object. Calculation is performed based on features to be found in the
 tables.
> (tab2<-table(p2,train$admit))
Error in table(p2, train$admit) : all arguments must have the same length
> (tab2<-table(p2,test$admit))

p2   0  1
  0 47 21
  1  3  4
> 1-sum(diag(tab2))/sum(tab2)
[1] 0.32
```

Program #29

KNN Algorithm in R using iris dataset

SOURCE CODE

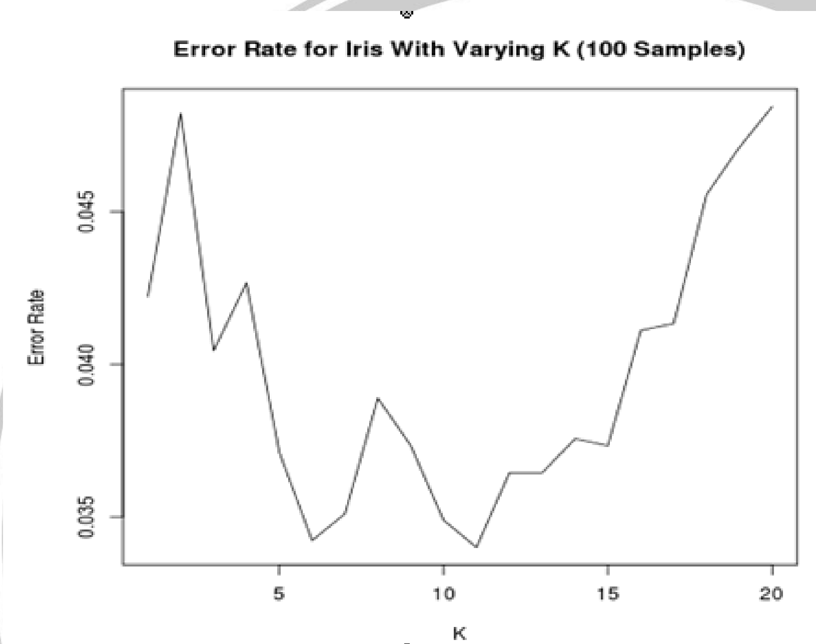
/*Alan Payyappilly*/

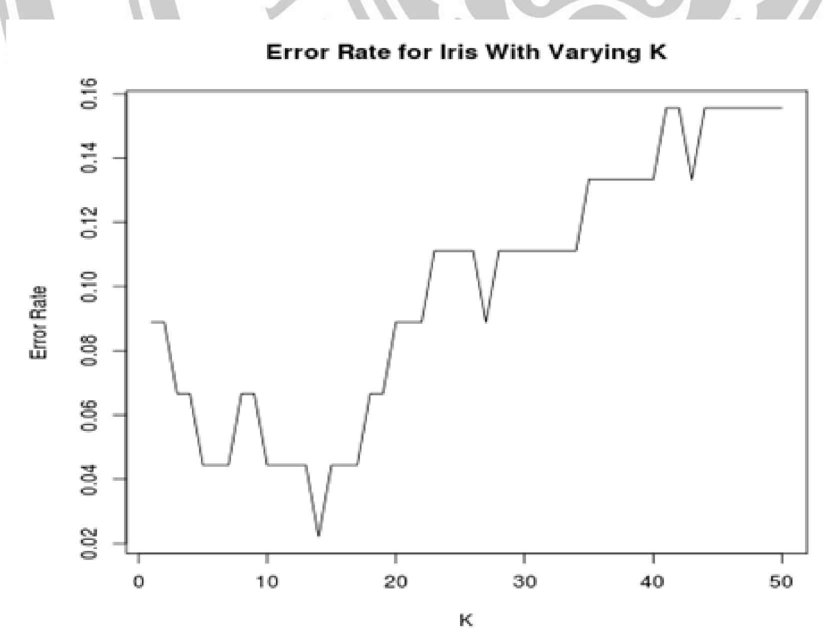
```
#kNN Tutotrial on Iris Data Set#### library(class) #Has the knn function
set.seed(4948493) #Set the seed for reproducibility #Sample the Iris data set (70% train,
30% test) ir_sample<-sample(1:nrow(iris),size=nrow(iris)*.7) ir_train<-iris[ir_sample,]
#Select the 70% of rows ir_test<-iris[-ir_sample,] #Select the 30% of rows

#First Attempt to Determine Right K#### iris_acc<-numeric() #Holding variable
for(i in 1:50){
#Apply knn with k = i
predict<-knn(ir_train[,-5],ir_test[,-5], ir_train$Species,k=i)
iris_acc<-c(iris_acc, mean(predict==ir_test$Species))
}
#Plot k= 1 through 30
plot(1-iris_acc,type="l",ylab="Error Rate", xlab="K",main="Error Rate for Iris With
Varying K")

#Try many Samples of Iris Data Set to Validate K#### trial_sum<-numeric(20)
trial_n<-numeric(20) set.seed(6033850) for(i in 1:100){
ir_sample<-sample(1:nrow(iris),size=nrow(iris)*.7) ir_train<-iris[ir_sample,]
ir_test<-iris[-ir_sample,] test_size<-nrow(ir_test) for(j in 1:20){
predict<-knn(ir_train[,-5],ir_test[,-5], ir_train$Species,k=j)
trial_sum[j]<-trial_sum[j]+sum(predict==ir_test$Species) trial_n[j]<-trial_n[j]+test_size
}
}
```

plot(1-trial_sum / trial_n,type="l",ylab="Error Rate", xlab="K",main="Error Rate for Iris With

Varying K (100 Samples)")

OUTPUT





Program #30

SVM Algorithm in R using iris dataset

SOURCE CODE

/*Alan Payyappilly*/

```
# install.packages("tidyverse")
library(tidyverse)
library(e1071)
set.seed(42) # To make our document recreatable
data(iris)
head(iris, 20)
index <- c(1:nrow(iris))
test.index <- sample(index, size = (length(index)/3))
train <- iris[-test.index ,]
test <- iris[test.index ,]
svm.model.linear <- svm(Species ~ ., data = train, kernel = 'linear')
svm.model.linear
table(Prediction = predict(svm.model.linear, train),Truth = train$Species)
```

OUTPUT

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa

```
Call:
svm(formula = Species ~ ., data = train, kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.25

Number of Support Vectors:  24

            Truth
Prediction   setosa versicolor virginica
  setosa         37          0         0
  versicolor      0         35         0
  virginica       0          1        27
```

Program #31

Random Forest Classification using (Titanic / Mushroom) dataset

SOURCE CODE

/*Alan Payyappilly*/

```
#install.packages("caret",dependencies=TRUE)
#install.packages("randomForest")

library("titanic")
library(caret)
library(randomForest)

#Training dataset
head(titanic_train)
```

```
> head(titanic_train)
  PassengerId Survived Pclass                                                Name    Sex Age SibSp Parch           Ticket    Fare Cabin Embarked
1           1        0      3                             Braund, Mr. Owen Harris   male  22     1     0        A/5 21171  7.2500              S
2           2        1      1 Cumings, Mrs. John Bradley (Florence Briggs Thayer) female  38     1     0         PC 17599 71.2833   C85        C
3           3        1      3                              Heikkinen, Miss. Laina female  26     0     0 STON/O2. 3101282  7.9250              S
4           4        1      1        Futrelle, Mrs. Jacques Heath (Lily May Peel) female  35     1     0           113803 53.1000  C123        S
5           5        0      3                            Allen, Mr. William Henry   male  35     0     0           373450  8.0500              S
6           6        0      3                                    Moran, Mr. James   male  NA     0     0           330877  8.4583              Q
```

```
#testing dataset
head(titanic_test)
```

```
> head(titanic_test)
  PassengerId Pclass                                         Name    Sex  Age SibSp Parch  Ticket    Fare Cabin Embarked
1         892      3                             Kelly, Mr. James   male 34.5     0     0  330911  7.8292              Q
2         893      3             Wilkes, Mrs. James (Ellen Needs) female 47.0     1     0  363272  7.0000              S
3         894      2                    Myles, Mr. Thomas Francis   male 62.0     0     0  240276  9.6875              Q
4         895      3                             Wirz, Mr. Albert   male 27.0     0     0  315154  8.6625              S
5         896      3 Hirvonen, Mrs. Alexander (Helga E Lindqvist) female 22.0     1     1 3101298 12.2875              S
6         897      3                   Svensson, Mr. Johan Cervin   male 14.0     0     0    7538  9.2250              S
```

```
#cross-tabs between “Survived” and each other variable
table(titanic_train[,c('Survived','Pclass')])
```

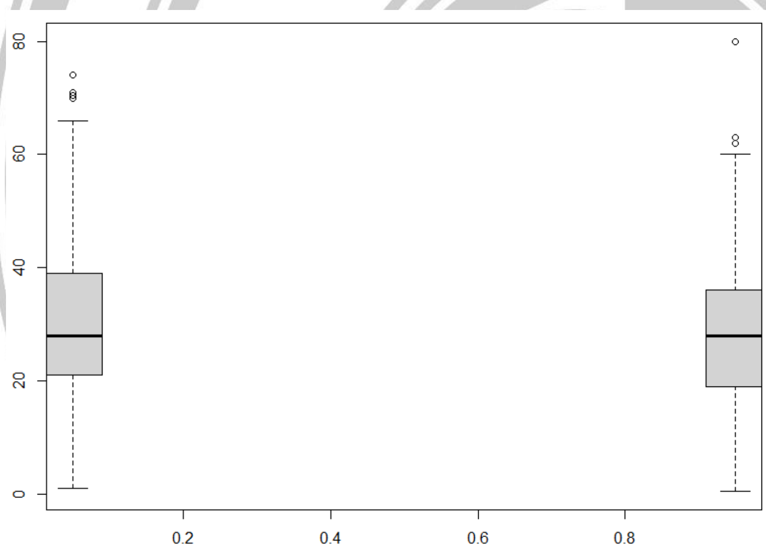
```
> table(titanic_train[,c('Survived','Pclass')])
        Pclass
Survived   1   2   3
       0  80  97 372
       1 136  87 119
```

#“conditional” box plots to compare the distribution of each continuous variable, conditioned on whether the passengers survived or not

#install.packages("fields")

library(fields)

bplot.xy(titanic_train$Survived, titanic_train$Age)



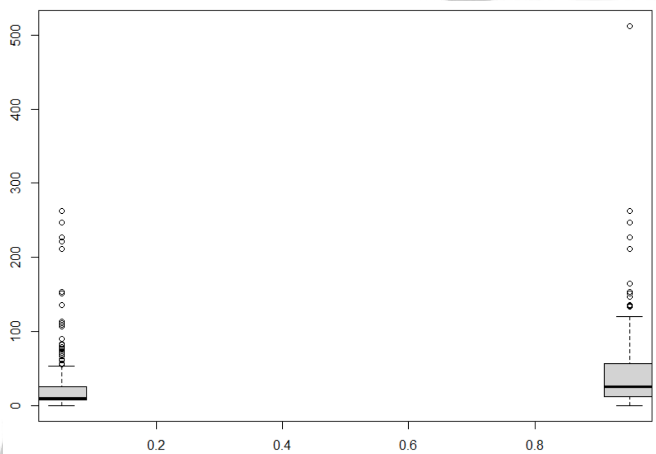
#if you summarize it, there are lots of NA’s. So, let’s exclude the variable Age, because it doesn’t have a big impact on Survived, and because the NA’s make it hard to work with.

summary(titanic_train$Age)

```
> summary(titanic_train$Age)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
   0.42   20.12   28.00   29.70   38.00   80.00     177 
```

#In the below boxplot, the boxplot for Fares are much different for those who survived and those who didn’t. Again, the y-axis is Fare and the x-axis is Survived.

bplot.xy(titanic_train$Survived, titanic_train$Fare)



#On summarizing you’ll find that there are no NA’s for Fare. So, let’s include this variable.

summary(titanic_train$Fare)

```
> summary(titanic_train$Fare)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    7.91   14.45   32.20   31.00  512.33
```

Converting ‘Survived’ to a factor

train$Survived <- factor(train$Survived)

Set a random seed

set.seed(51)

Training using ‘random forest’ algorithm

model <- train(Survived ~ Pclass + Sex + SibSp +

Embarked + Parch + Fare, # Survived is a function of the variables we decided to include

data = train, # Use the train data frame as the training data

method = 'rf',# Use the 'random forest' algorithm

trControl = trainControl(method = 'cv', # Use cross-validation

number = 5) # Use 5 folds for cross-validation

```
model
Random Forest

891 samples
  6 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 712, 713, 713, 712, 714
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
  2     0.8047116  0.5640887
  5     0.8070094  0.5818153
  8     0.8002236  0.5704306

Accuracy was used to select the optimal model using
The final value used for the model was mtry = 5.
```

summary(titanic_test)

```
> summary(titanic_test)
  PassengerId        Pclass          Name               Sex                 Age             SibSp            Parch             Ticket                Fare
 Min.   : 892.0   Min.   :1.000   Length:418         Length:418         Min.   : 0.17   Min.   :0.0000   Min.   :0.0000   Length:418         Min.   :  0.000
 1st Qu.: 996.2   1st Qu.:1.000   Class :character   Class :character   1st Qu.:21.00   1st Qu.:0.0000   1st Qu.:0.0000   Class :character   1st Qu.:  7.896
 Median :1100.5   Median :3.000   Mode  :character   Mode  :character   Median :27.00   Median :0.0000   Median :0.0000   Mode  :character   Median : 14.454
 Mean   :1100.5   Mean   :2.266                                         Mean   :30.27   Mean   :0.4474   Mean   :0.3923                      Mean   : 35.627
 3rd Qu.:1204.8   3rd Qu.:3.000                                         3rd Qu.:39.00   3rd Qu.:1.0000   3rd Qu.:0.0000                      3rd Qu.: 31.500
 Max.   :1309.0   Max.   :3.000                                         Max.   :76.00   Max.   :8.0000   Max.   :9.0000                      Max.   :512.329
                                                                        NA's   :86                                                           NA's   :1
    Cabin             Embarked
 Length:418         Length:418
 Class :character   Class :character
 Mode  :character   Mode  :character
```

titanic_test$Fare<-ifelse(is.na(titanic_test$Fare), mean(titanic_test$Fare, na.rm=TRUE), titanic_test$Fare)

titanic_test$Survived<-predict(model, newdata=titanic_test)

titanic_test$Survived

```
  [1] 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 1 0 0 0 1 1 0 1 0 1
 [55] 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 0 1 0
[109] 0 0 0 1 1 1 1 0 0 1 1 1 1 0 1 0 0 1 0 1 0 0 0 0 0
[163] 1 0 0 1 0 0 1 0 0 0 0 0 0 1 1 1 1 1 0 1 1 0 1 0 1
[217] 1 0 1 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 1 1 1
[271] 1 0 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0
[325] 1 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0
```

Program #32

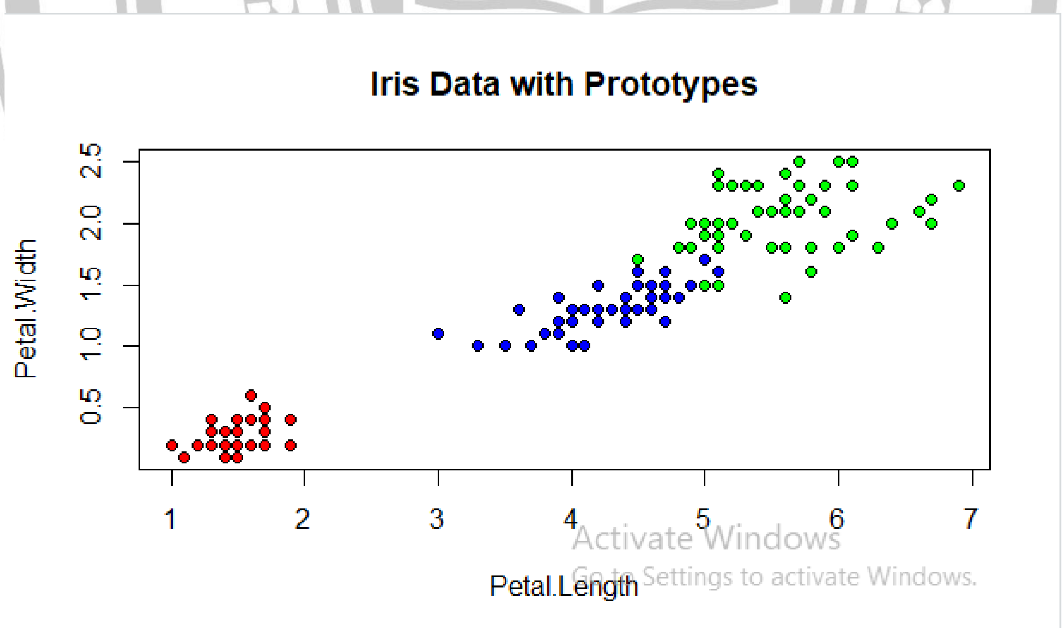
Random Forest Classification using Gini Index in iris dataset

SOURCE CODE

/*Alan Payyappilly*/

```
library(randomForest)
data(iris)
iris.rf <- randomForest(iris[,-5], iris[,5], prox=TRUE)
print(iris.rf)
iris.p <- classCenter(iris[,-5], iris[,5], iris.rf$prox)
plot(iris[,3], iris[,4], pch=21, xlab=names(iris)[3], ylab=names(iris)[4],
    bg=c("red", "blue", "green")[as.numeric(factor(iris$Species))],
    main="Iris Data with Prototypes")
points(iris.p[,3], iris.p[,4], pch=21, cex=2, bg=c("red", "blue", "green"))
```

OUTPUT



Program #33

Simple Linear Regression using R using mtcars dataset

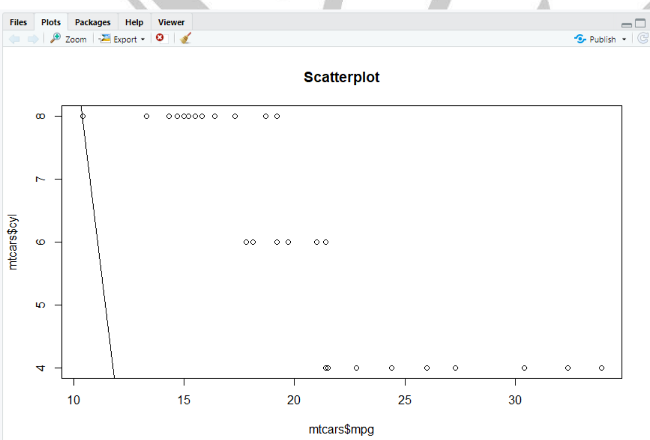
SOURCE CODE

/*Alan Payyappilly*/

Using mtcars dataset
> model <- lm(mtcars$mpg ~ mtcars$cyl)
> summary(model)
Using visualization for mtcars dataset
> plot(mtcars$mpg,mtcars$cyl,main="Scatterplot")
> abline(model)

OUTPUT

```
> model <- lm(mtcars$mpg ~ mtcars$cyl)
> summary(model)

Call:
lm(formula = mtcars$mpg ~ mtcars$cyl)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.9814 -2.1185  0.2217  1.0717  7.5186 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  37.8846     2.0738   18.27  < 2e-16 ***
mtcars$cyl   -2.8758     0.3224   -8.92 6.11e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.206 on 30 degrees of freedom
Multiple R-squared:  0.7262,	Adjusted R-squared:  0.7171 
F-statistic: 79.56 on 1 and 30 DF,  p-value: 6.113e-10

> plot(mtcars$mpg,mtcars$cyl,main="Scatterplot")
> abline(model)
> |
```



Program #34

Binary Logistic Regression using R

SOURCE CODE

/*Alan Payyappilly*/

```
install.packages('mlbench')
install.packages('MASS')
install.packages('pROC')
```

This dataset has a binary response (outcome, dependent) variable called admit. There are three predictor variables: gre, gpa and rank. We will treat the variables gre and gpa as continuous. The variable rank takes on the values 1 through 4. Institutions with a rank of 1 have the highest prestige, while those with a rank of 4 have the lowest. GRE (Graduate Record Exam scores), GPA (grade point average) and prestige of the undergraduate institution, effect admission into graduate school. The response variable, admit/don't admit, is a binary variable.

OUTPUT

```
mydata <- read.csv("https://stats.idre.ucla.edu/stat/data/binary.csv")
> head(mydata)
  admit gre  gpa rank
1     0 380 3.61    3
2     1 660 3.67    3
3     1 800 4.00    1
4     1 640 3.19    4
5     0 520 2.93    4
6     1 760 3.00    2
> summary(mydata)
     admit             gre             gpa             rank      
 Min.   :0.0000   Min.   :220.0   Min.   :2.260   Min.   :1.000  
 1st Qu.:0.0000   1st Qu.:520.0   1st Qu.:3.130   1st Qu.:2.000  
 Median :0.0000   Median :580.0   Median :3.395   Median :2.000  
 Mean   :0.3175   Mean   :587.7   Mean   :3.390   Mean   :2.485  
 3rd Qu.:1.0000   3rd Qu.:660.0   3rd Qu.:3.670   3rd Qu.:3.000  
 Max.   :1.0000   Max.   :800.0   Max.   :4.000   Max.   :4.000  
> supply(mydata)
Error in supply(mydata) : could not find function "supply"
> sapply(mydata)
```

```
Error in match.fun(FUN) : argument "FUN" is missing, with no default
> sapply(mydata,sd)
     admit         gre         gpa        rank
  0.4660867 115.5165364   0.3805668   0.9444602
> xtabs(~admit + rank, data = mydata)
     rank
admit  1  2  3  4
    0 28 97 93 55
    1 33 54 28 12
> mydata$rank <- factor(mydata$rank)
> mylogit <- glm(admit ~ gre + gpa + rank, data = mydata, family = "binomial")
> summary(mylogit)

Call:
glm(formula = admit ~ gre + gpa + rank, family = "binomial",
    data = mydata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6268  -0.8662  -0.6388   1.1490   2.0790

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.989979   1.139951  -3.500 0.000465 ***
gre          0.002264   0.001094   2.070 0.038465 *
gpa          0.804038   0.331819   2.423 0.015388 *
rank2       -0.675443   0.316490  -2.134 0.032829 *
rank3       -1.340204   0.345306  -3.881 0.000104 ***
rank4       -1.551464   0.417832  -3.713 0.000205 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 499.98  on 399  degrees of freedom
Residual deviance: 458.52  on 394  degrees of freedom
AIC: 470.52

Number of Fisher Scoring iterations: 4
```

Program #35

K Means Algorithm in R using wholesale customers

SOURCE CODE

/*Alan Payyappilly*/

// loading data

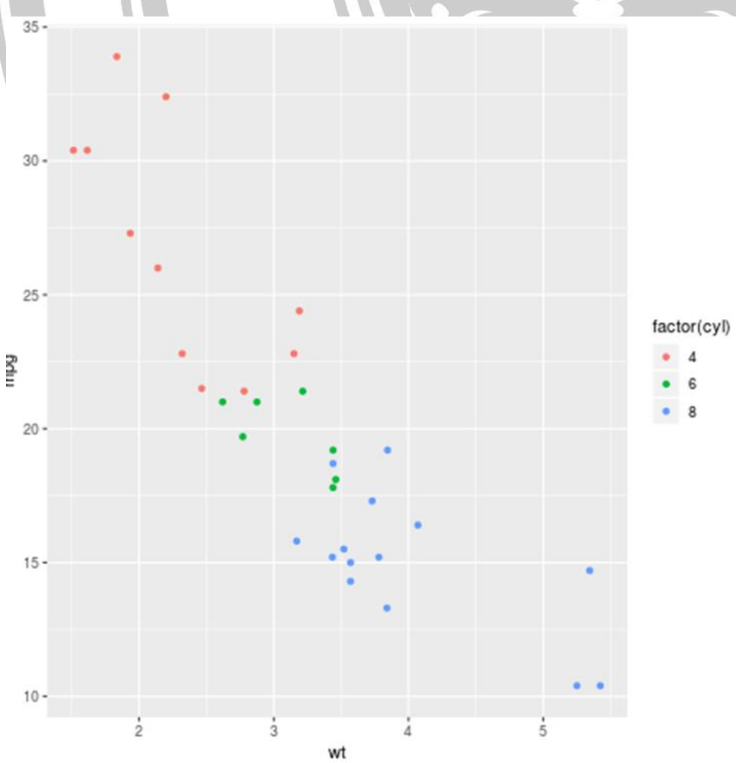
```
 data <-read.csv("Wholesale customers data.csv",header=T)
summary(data)
```

```
> data<-read.csv("wholesale customers data.csv",header=T)
> summary(data)
    Channel          Region          Fresh             Milk           Grocery          Frozen          Detergents_Paper    Delicassen
 Min.   :1.000   Min.   :1.000   Min.   :     3   Min.   :   55   Min.   :    3   Min.   :   25.0   Min.   :    3.0   Min.   :    3.0
 1st Qu.:1.000   1st Qu.:2.000   1st Qu.:  3128   1st Qu.: 1533   1st Qu.: 2153   1st Qu.:  742.2   1st Qu.:  256.8   1st Qu.:  408.2
 Median :1.000   Median :3.000   Median :  8504   Median : 3627   Median : 4756   Median : 1526.0   Median :  816.5   Median :  965.5
 Mean   :1.323   Mean   :2.543   Mean   : 12000   Mean   : 5796   Mean   : 7951   Mean   : 3071.9   Mean   : 2881.5   Mean   : 1524.9
 3rd Qu.:2.000   3rd Qu.:3.000   3rd Qu.: 16934   3rd Qu.: 7190   3rd Qu.:10656   3rd Qu.: 3554.2   3rd Qu.: 3922.0   3rd Qu.: 1820.2
 Max.   :2.000   Max.   :3.000   Max.   :112151   Max.   :73498   Max.   :92780   Max.   :60869.0   Max.   :40827.0   Max.   :47943.0
```

```
library(ggplot2)
# Use plots...
plot(cars)

# Even ggplot!
qplot(wt, mpg, data = mtcars, colour = factor(cyl))

data <-read.csv("Wholesale customers data.csv",header=T)
summary(data)
top.n.custs <- function (data,cols,n=5) {
idx.to.remove <-integer(0)
for (c in cols){
col.order <-order(data[,c],decreasing=T)
idx <-head(col.order, n) #Take the first n of the sorted column C to
idx.to.remove <-union(idx.to.remove,idx)
}
return(idx.to.remove)
}
top.custs <-top.n.custs(data,cols=3:8,n=5)
length(top.custs)
```

```
data[top.custs,]
data.rm.top <-data[-c(top.custs),]
set.seed(76964057)
k <-kmeans(data.rm.top[,-c(1,2)], centers=5)
k$centers
table(k$cluster)
rng<-2:20 #K from 2 to 20
tries<-100
avg.totw.ss<-integer(length(rng))
for(v in rng){
v.totw.ss<-integer(tries)
for(i in 1:tries){
k.temp<-kmeans(data.rm.top,centers=v)
v.totw.ss[i]<-k.temp$tot.withinss
}
avg.totw.ss[v-1]<-mean(v.totw.ss)
}
plot(rng,avg.totw.ss,type="b", main="Total Within SS by Various K",
ylab="Average Total Within Sum of Squares",
xlab="Value of K")
```

OUTPUT



Program #36

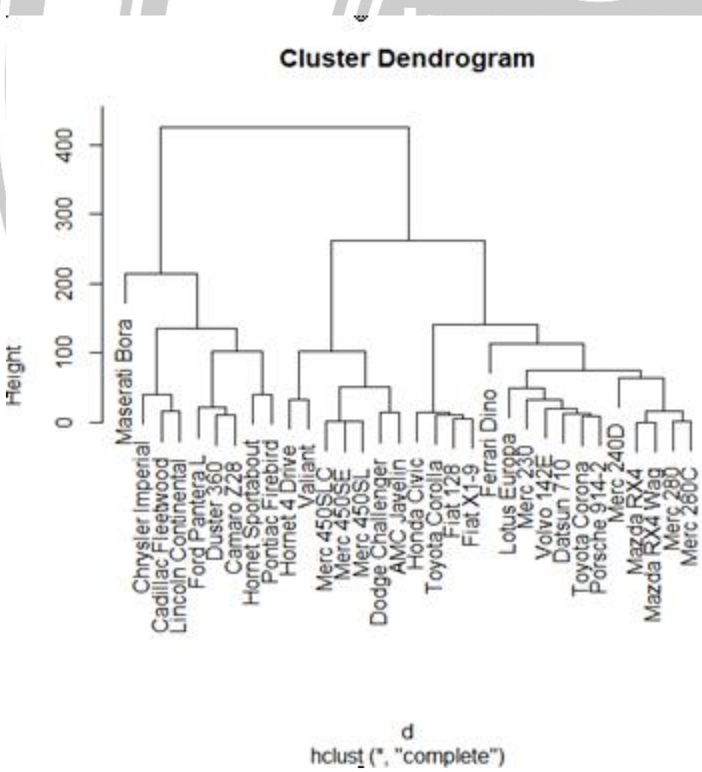
Hierarchical clustering using R using mtcars / iris

SOURCE CODE

/*Alan Payyappilly*/

```
> d <-
dist(as.matrix(mtcars
))
> hc <- hclust(d)
> plot(hc)
```

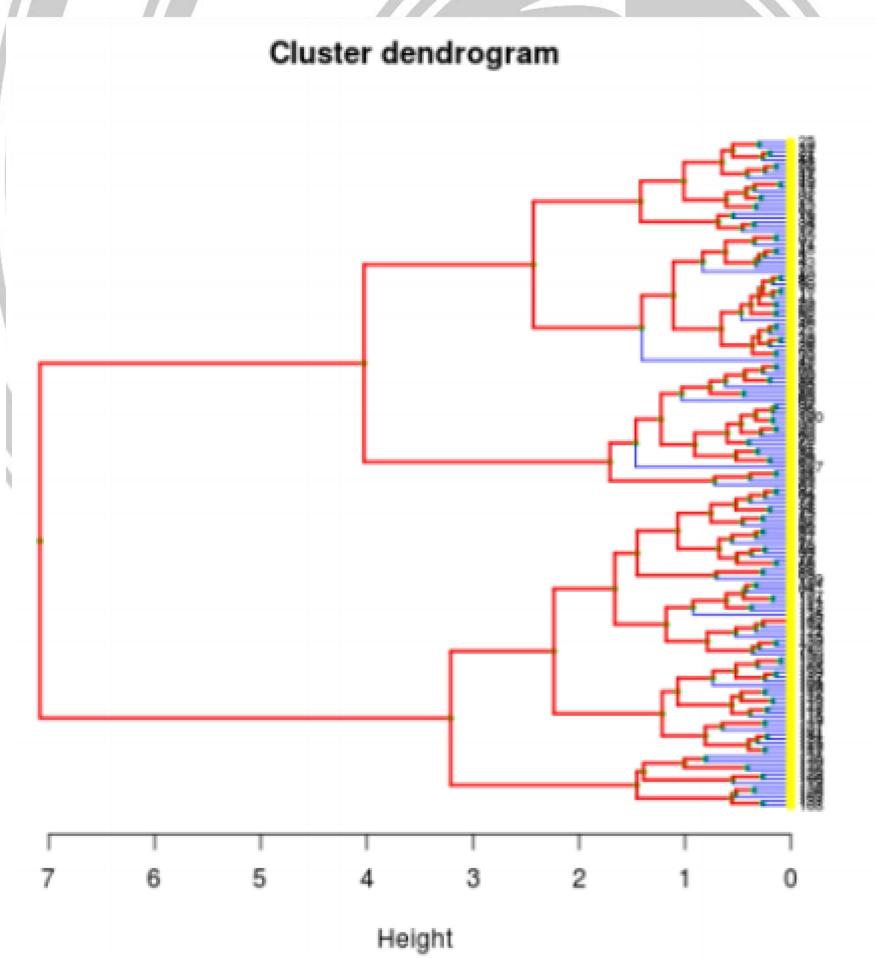
OUTPUT



SOURCE CODE

```
data <- dist(iris[,1:4])
hcd <-
as.dendrogram(hclust(data
)) # Define nodePar
nodePar <- list(lab.cex = 0.6, pch =
            c(20, 19), cex = 0.7, col =
            c("green","yellow"))
plot(hcd, xlab = "Height", nodePar = nodePar, main = "Cluster
    dendrogram", edgePar = list(col = c("red","blue"), lwd = 2:1), horiz =
    TRUE)
```

OUTPUT



Program #37

Simple Calculator in R using Shiny package

SOURCE CODE

```
/*Alan Payyappilly*/

library(shiny)
ui <- fluidPage(
# Application title
titlePanel("Simple Calculator"),
sidebarLayout(
sidebarPanel(
numericInput("num1","enter the first number",0),
numericInput("num2","enter the second number",0),
selectInput("operator","select the operator",
choices=c("+","-","*","/"))
),
mainPanel(
h2("Result:"),
textOutput("output")
)
)
)
server <- function(input, output) {
output$output <- renderText({
switch(input$operator,
"+"=input$num1 + input$num2,
"-"=input$num1 - input$num2,
"*"=input$num1 * input$num2,
"/"=input$num1 / input$num2)
```

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

OUTPUT