|  |
| --- |
| **Program #1** |
| List and explain the different variables, constants and operators in R. |

**SOURCE CODE**

**/\*Geo George \*/**

**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.

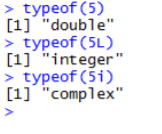
Numeric constants followed by L are regarded as integer and those followed by i are regarded as complex.

typeof(5)

typeof(5L)

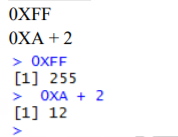
typeof(5i)

# OUTPUT



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

**Examples**



**Character Constants**

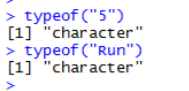
Character constants can be represented using either single quotes (') or double quotes (") as delimiters.

**Example**

typeof("5")

typeof("Run")

# OUTPUT



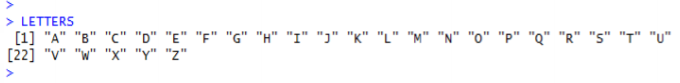
**Built-in Constants**

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

**Example**

# OUTPUT

LETTERS



letters



pi



month.name



month.abb



**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

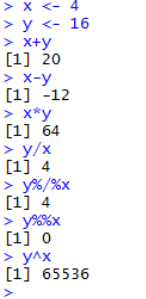
y/x

y%/%x

y%%x

y^x

# OUTPUT



**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

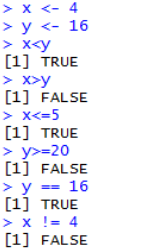
x<=5

y>=20

y == 16

x != 4

# OUTPUT



**Logical Operators**

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

|  |  |
| --- | --- |
| Operator | Description |
| ! | Logical NOT |
| & | Element-wise logical AND |
| && | Logical AND |
| | | Element-wise logical OR |
| || | Logical OR |

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

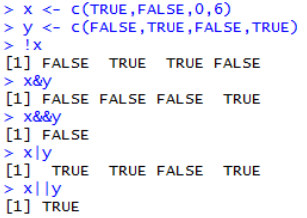
x&y

x&&y

x|y

x||y

# OUTPUT



**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

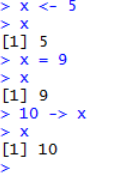
x = 9

x

10 -> x

X

# OUTPUT



|  |
| --- |
| **Program #2** |
| List and explain the Vector data type. |

**/\* Geo George \*/**

**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.

v <-6.6:12.6

print(v)

# 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**

**Using sequence (Seq.) operator**

# 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. |

**/\* Geo George \*/**

**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])

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

# OUTPUT

$`1st\_Quarter`

[1] "Jan" "Feb" "Mar"

|  |
| --- |
| **Program #4** |
| List and explain the Matrix data type. |

**/\* Geo George \*/**

**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.

print(P[1,3])

# OUTPUT

[1] 5

|  |
| --- |
| **Program #5** |
| List and explain the Arrays data type. |

**/\* Geo George \*/**

**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. |

**/\* Geo George \*/**

**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)

print(factor\_data)

print(is.factor(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. |

**/\* Geo George \*/**

**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). |

**/\* Geo George \*/**

* **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. |

**/\* Geo George \*/**

**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? |

**/\* Geo George \*/**

**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)

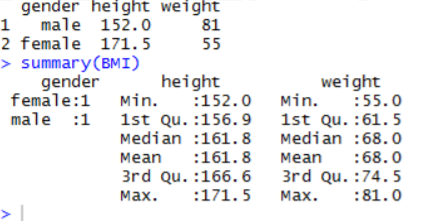
**create data frame**

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

BMI

summary(BMI)

# OUTPUT



**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

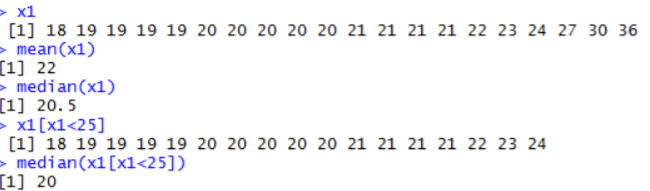
mean(x1)

median(x1)

x1[x1<25]

median(x1[x1<25])

# OUTPUT



**SOURCE CODE**

modex1 <-which(xt==max(xt))

modex1

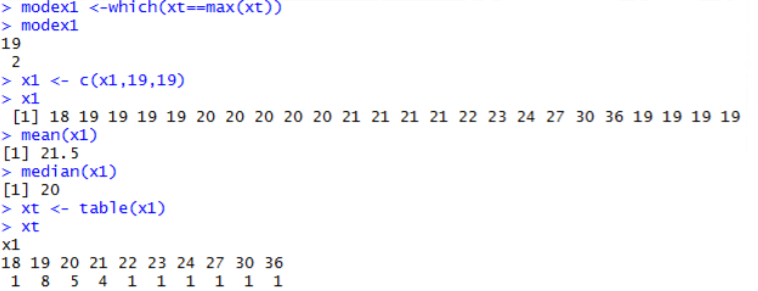
x1 <- c(x1,19,19)

x1 mean(x1) median(x1)

xt <- table(x1)

xt

# OUTPUT



**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)

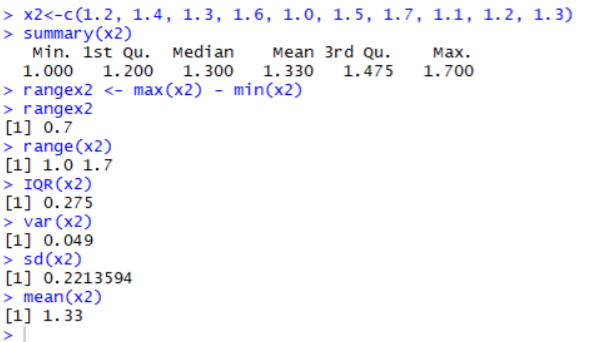
IQR(x2)

var(x2)

sd(x2)

mean(x2)

# OUTPUT



**SOURCE CODE**

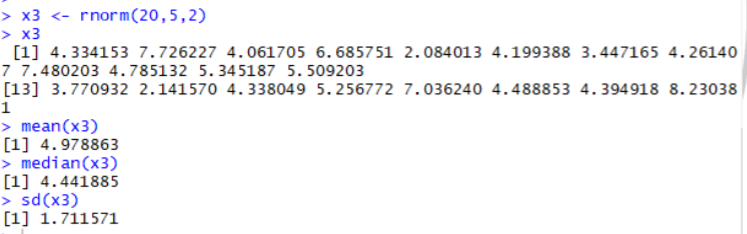
x3 <- rnorm(20,5,2) x3

mean(x3)

median(x3)

sd(x3)

# OUTPUT



**SOURCE CODE**

set.seed(100)

x<-rnorm(100, mean=0, sd=1)

mean(x)

median(x)

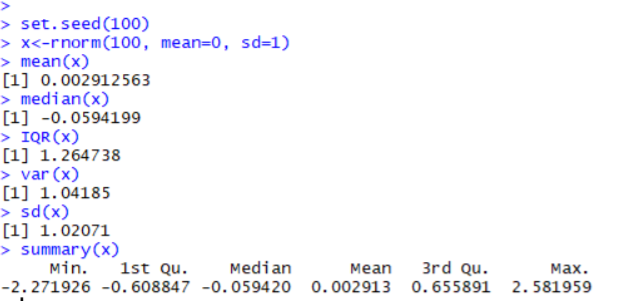
IQR(x)

var(x)

sd(x)

summary(x)

# OUTPUT



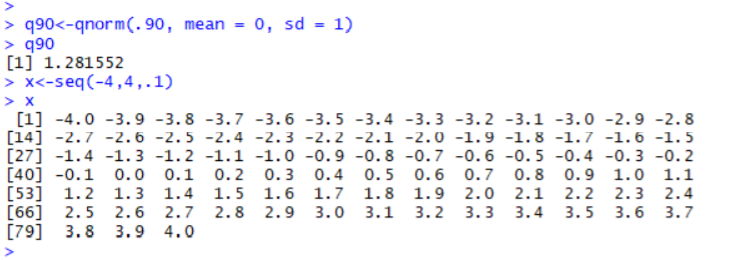
**SOURCE CODE**

q90<-qnorm(.90, mean = 0, sd = 1) q90

x<-seq(-4,4,.1)

x

# OUTPUT

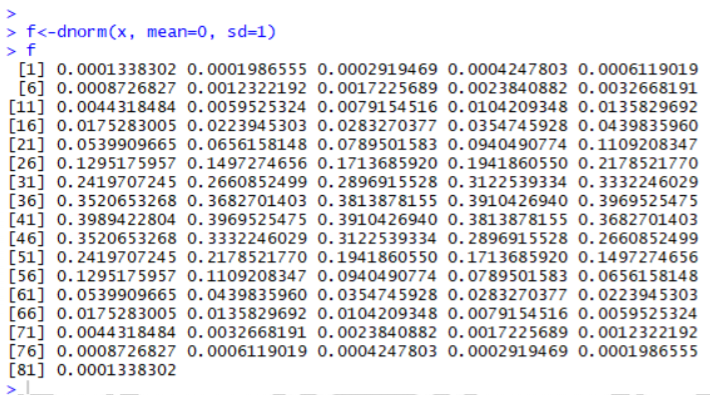


**SOURCE CODE**

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

f

# OUTPUT

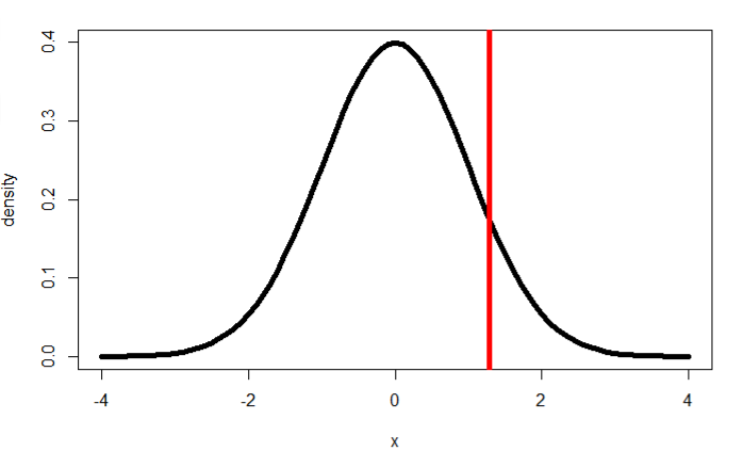


**SOURCE CODE**

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**

**/\* Geo George \*/**

**#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**

**/\* Geo George \*/**

**#Random Generation of Numbers**

runif(1)

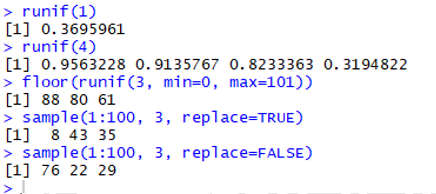
runif(4)

floor(runif(3, min=0, max=101))

sample(1:100, 3, replace=TRUE)

sample(1:100, 3, replace=FALSE)

# OUTPUT



|  |
| --- |
| **Program #13a** |
| "Generate the Cumulative Distribution Function and Probability Density Function of Normal distribution". |

**SOURCE CODE**

**/\* Geo George \*/**

**#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.

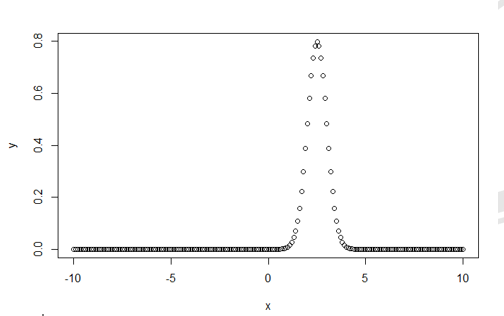
y <- dnorm(x, mean = 2.5, sd = 0.5)

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)

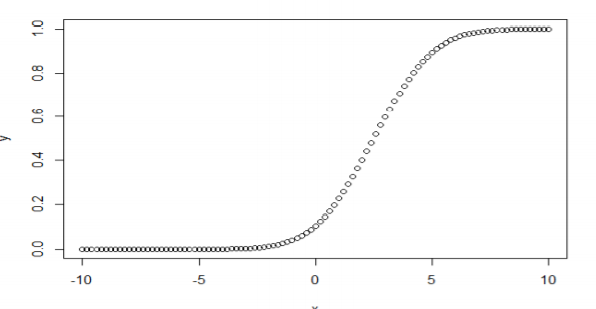
# Plot the graph.

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)

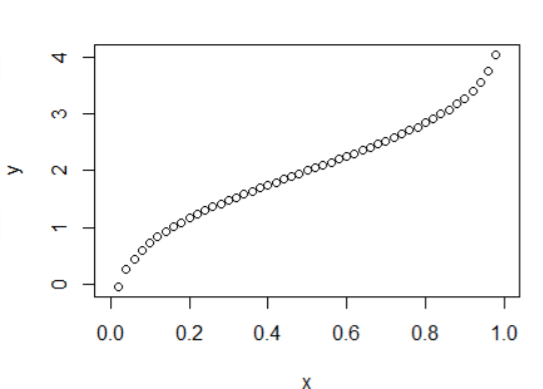
# Plot the graph.

plot(x,y)

# Save the file.

dev.off()

# OUTPUT



**SOURCE CODE**

**#rnorm**

# Create a sample of 50 numbers which are normally distributed.

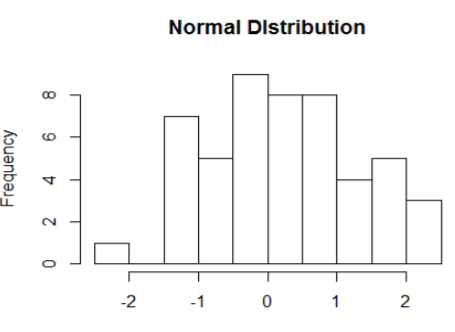
y <- rnorm(50) # Plot the histogram for this sample.

hist(y, main = "Normal DIstribution")

# Save the file.

dev.off()

# OUTPUT



|  |
| --- |
| **Program #13b** |
| 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? |

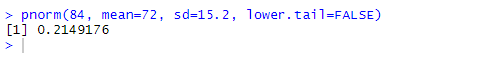
**SOURCE CODE**

**/\* Geo George \*/**

**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



**Answer**

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

|  |
| --- |
| **Program #14a** |
| "Generate the Cumulative Distribution Function and Probability Density Function of Binomial distribution". |

**SOURCE CODE**

**/\***  **Geo George** **\*/**

**#dbinom**

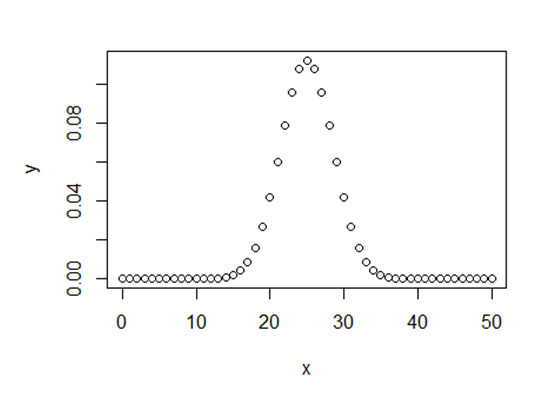
# Create a sample of 50 numbers which are incremented by 1. x <- seq(0,50,by = 1)

# 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 #14b** |
| 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**

**/\* Geo George \*/**

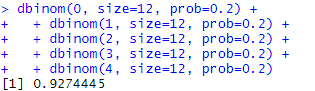
**Solution**

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.



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

# OUTPUT



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



**Answer**

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

|  |
| --- |
| **Program #15a** |
| "Generate the Cumulative Distribution Function and Probability Density Function of Poisson distribution". |

**SOURCE CODE**

**/\* Geo George \*/**

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)

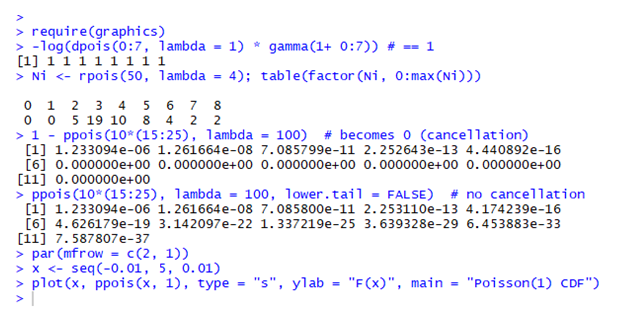
dPoisson25 <- data.frame(N=n,

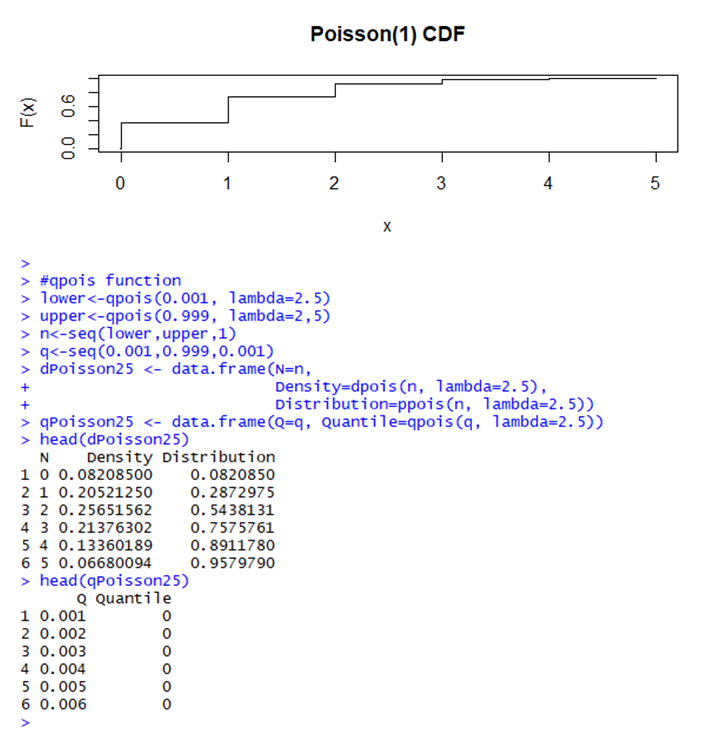
Density=dpois(n, lambda=2.5), Distribution=ppois(n, lambda=2.5))

qPoisson25 <- data.frame(Q=q, Quantile=qpois(q, lambda=2.5)) head(dPoisson25)

head(qPoisson25)

# OUTPUT





|  |
| --- |
| **Program #15b** |
| 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**

**/\* Geo George \*/**

**Solution**

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



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

#### 

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 #16** |
| Explain Data wrangling with "dplyr" package to transform, organize and summarize the given dataset. |

**SOURCE CODE**

**/\* Geo George \*/**

# OUTPUT

|  |
| --- |
| **Program #17** |
| Explain the Pie Chart, Bar Chart and Line Graph using the given dataset |

**/\* Geo George \*/**

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()

1. **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()

1. **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()

1. **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")

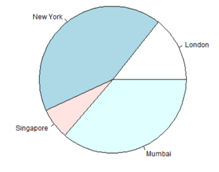
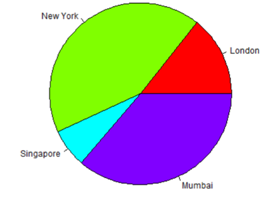
# Plot the chart.

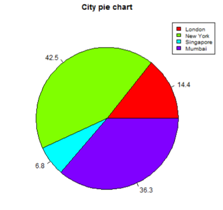
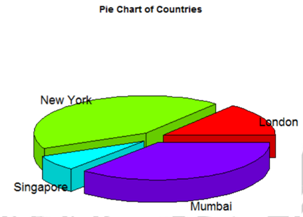
pie3D(x,labels = lbl,explode = 0.1, main = "Pie Chart of Countries ")

# 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()

1. **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)

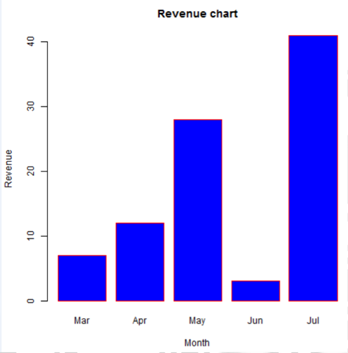
# Add the legend to the chart

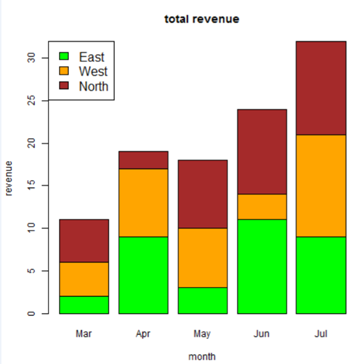
legend("topleft", regions, cex = 1.3, fill = colors)

#Save the file.

dev.off()

**OUTPUT**

****

****

**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.

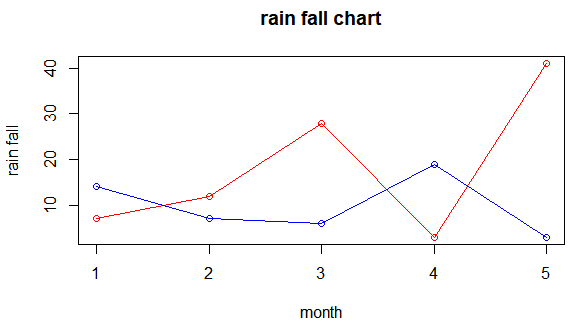
plot(x,type="o",col="red",xlab="month",ylab="rain fall", main="rain fall chart")

lines(y,type="o",col="blue")

#save the file.

dev.off()

**OUTPUT**

****

|  |
| --- |
| **Program #18** |
| Explain the plot and line functions by constructing the Sine and Cosine wave. |

**SOURCE CODE**

**/\* Geo George \*/**

x <-seq (0,8\*pi,length.out =100)

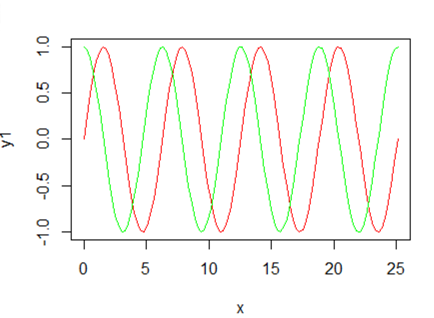
y1 <-sin(x)

y2 <-cos (x)

plot(x,y1,type="l",col="red")

lines(x,y2,col="green")

# OUTPUT



|  |
| --- |
| **Program #19** |
| Explain the different features of Histogram using the given dataset. |

**SOURCE CODE**

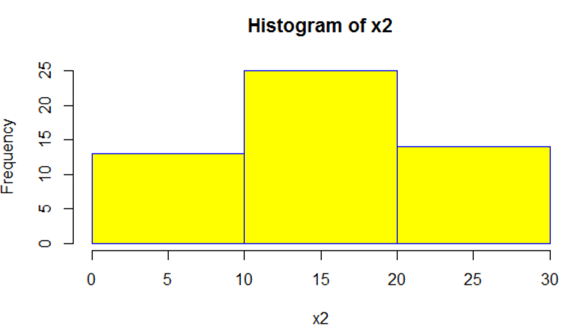
**/\* Geo George \*/**

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, 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 #20** |
| From a given data set plot a box plot, scatter plot and correl plot. |

**/\* Geo George \*/**

**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

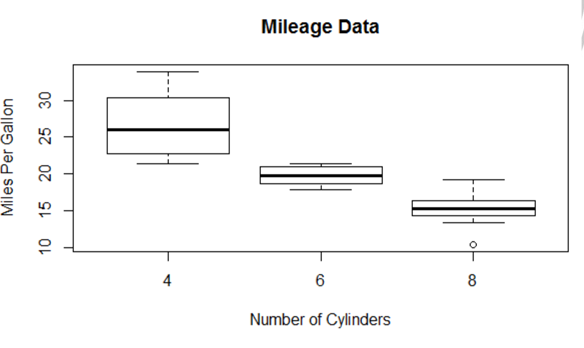
Valiant 18.1 6

**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')]

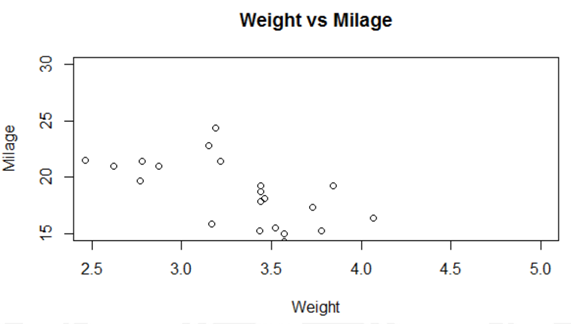
print(head(input))

#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))

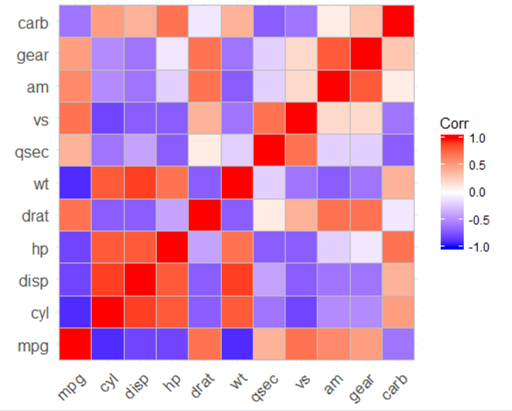
#Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.

plot(x = input$wt,y = input$mpg,

xlab = "Weight", ylab = "Milage", xlim = c(2.5,5), ylim = c(15,30),

main = "Weight vsMilage")

# OUTPUT



|  |
| --- |
| **Program #21** |
| Explain the power of ggplot2 using the given dataset |

**SOURCE CODE**

**/\* Geo George \*/**

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

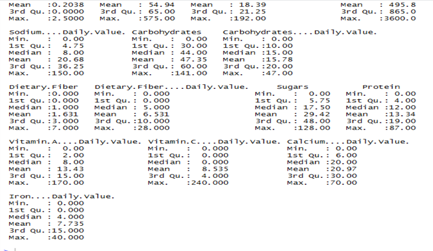
ggplot(menu, aes(Protein, Sugars, color=Category)) + geom\_point() + theme\_bw()

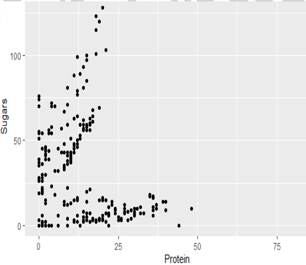
ggplot(menu, aes(Protein, Sugars, color=Category)) + geom\_point() + facet\_wrap(~Category) + theme\_bw()

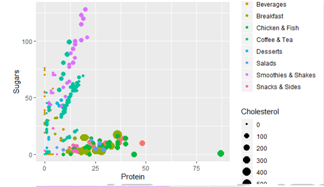
ggplot(menu, aes(Protein, Sugars)) + geom\_point() + facet\_wrap(~Category, scales="free")

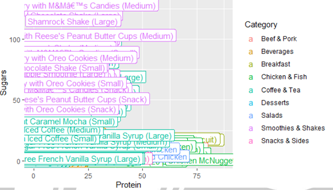
+ theme\_bw()

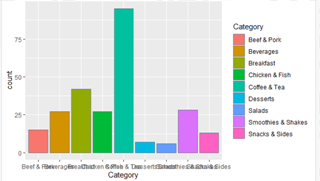
# OUTPUT

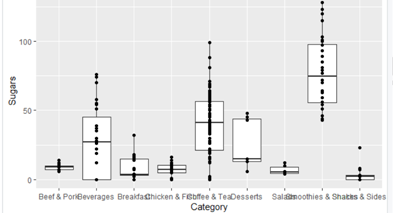


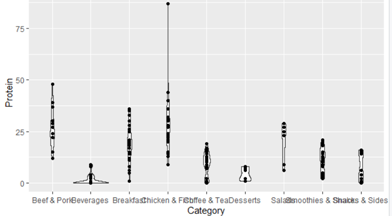


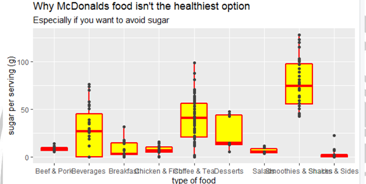


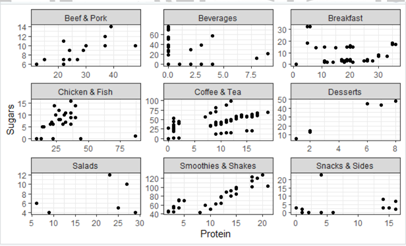












|  |
| --- |
| **Program #22** |
| Explain the power of shiny using the given dataset |

**SOURCE CODE**

**/\* Geo George \*/**

#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

