

**PES UNIVERSITY EC Campus**

**Electronic City, Hosur Road,**

**Bangalore-100**

A Project Report on

**Study of Diamonds**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

For the Academic year 2020

by

**A R MANYATHA - PES2UG19CS002**

**DEEPALI SURAJ ATTAVAR - PES2UG19CS106**

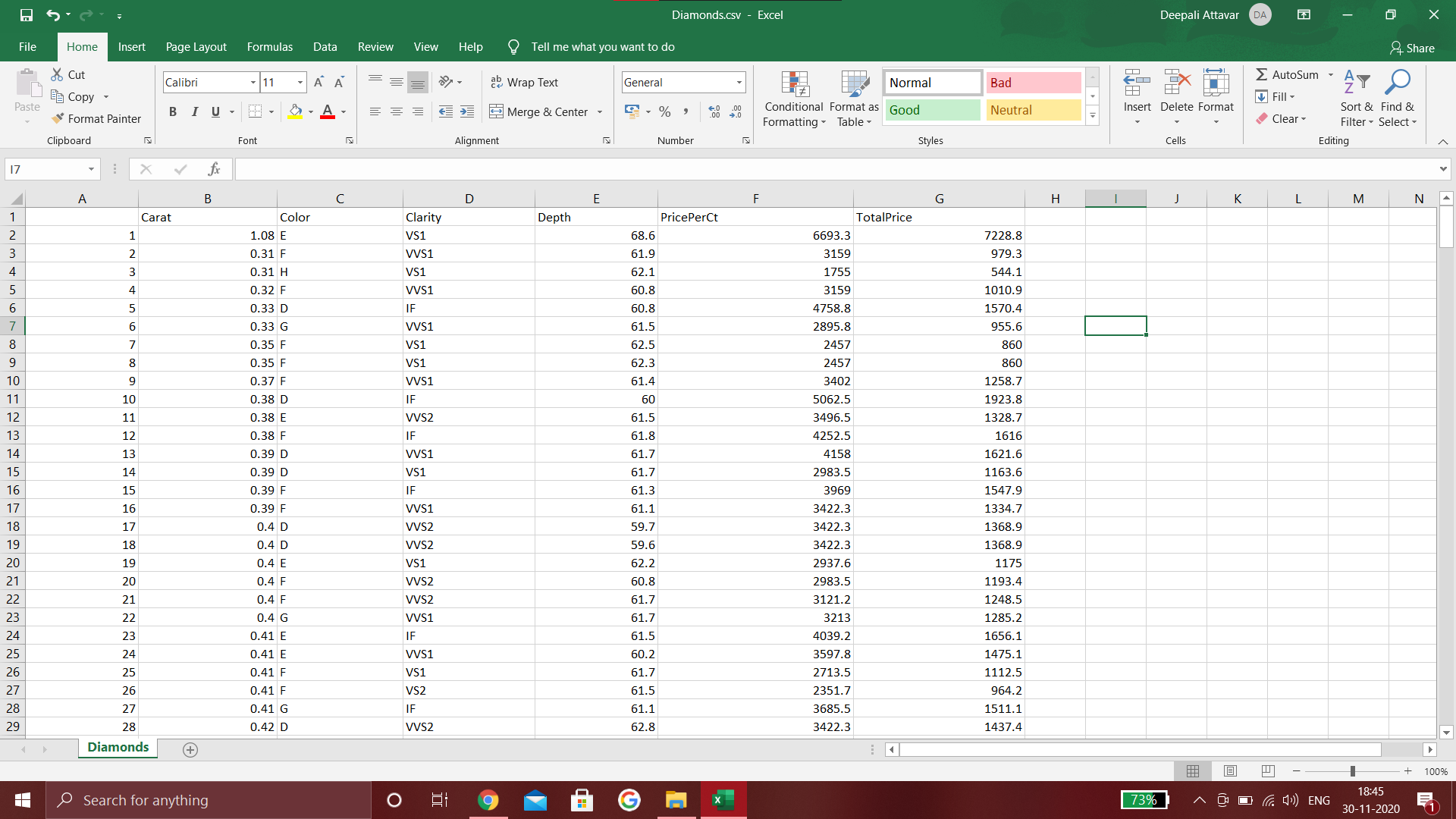
**MANASA K RAYACHOTI - PES2UG19CS215**

**ANANYA MUNGARA - PES2UG19CS040**

**TABLE OF CONTENTS**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Chapter No.** | **Title** | **Pg.No** | |  | **INTRODUCTION** | **4-5** | |  | **All Data Structures, along with access methods of each data structure.** | **6-13** | |  | **Operations on Date** | **14** | |  | **Conditional Statements, on Vector, Dataframe, Array, Matrix** | **15-17** | |  | **Loop on vector, Dataframe,array,matrics** | **15-17** | |  | **String Operations , Comparison, GREP,** | **18** | |  | **Functions to show recursion** | **19** | |  | **Usage of functions, all,any, apply, which,match, order, rank,sub, gsub,aggregate** | **20-23** | |  | **Plotting , PIE, Line, Scatter, BAR using R functions** | **24-29** | | **10.** | **Plotting , PIE, Line, Scatter, BAR using GGPLOT** | **30-34** | | **11.** | **Statistics: mean, median, summary, boxplot, histogram, factor,** | **35-42** | | **12.** | **Probability Distribution Functions: Any one** | **43-48** | | **13.** | **Hypothesis Testing using qqnorm and mean** | **49-51** | |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  | **Sample Screenshot of the csv file:** |  |



**CHAPTER-1**

**INTRODUCTION**

R is a language and environment for statistical computing and graphics. It is GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, …) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

One of R’s strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

R is available as Free Software under the terms of the [**Free Software Foundation**](http://www.gnu.org/)’s [**GNU General Public License**](https://www.r-project.org/COPYING) in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

**R Installation Procedure:**

In the link: **https://cran.r-project.org/mirrors.html**

find the link for India and open it.

Based on the operating system in your device, install it along with base and Rtools library based on the compiler.

On completing the download, install the rStudio for the device and begin with r programming

**Libraries Used:**

**GGplot**: is a plotting package that makes it simple to create complex plots from data in a data frame. It provides a more programmatic interface for specifying what variables to plot, how they are displayed, and general visual properties.

**Dataset:**

This classic dataset contains the prices and other attributes of almost 400 diamonds. It's a great dataset for beginners learning to work with data analysis and visualization. The dataset consists of 352 rows and 7 unique columns.

**-> Columns:**

ID: stands for a unique number for each diamond

Carat: stands for unit of measurement of physical weight of the diamond

Color: establishes the value of the diamond

Clarity: refers to the degree to which these imperfections are present in the diamond

Depth: refers to its measurement from top to bottom, from the table on the top of the diamond to the culet at its base.

PricePerCt: Diamonds are all priced per carat. It depends on factors such as the diamond’s cut quality, clarity, color and shape.

TotalPrice: stands for total price of the diamond listed.

**Analysis done in the project:**

The dataset which we chose from the Internet includes a wide range of approximately 400 different diamonds along with their properties which include carat, colour, clarity, depth, pricing per carat and finally the total price. A wide collection of tests and studies have been done on the dataset chosen. We have pictorially depicted the data for easier understanding using pie charts, line, scatter and bar graphs using basic functions of R and GGPLOT. We have found out the required results and the mean price per carat diamonds by applying statistics and verified the hypothesis testing using qqnorm and mean.

**CHAPTER-2**

**Data Structure**

**1. Data Structures, along with access methods of each data structure.**

To make the best of the R language, a strong understanding of the basic data types and data structures is important and how to operate on them.

Data structures are very important to understand because these are the objects you will manipulate on a day-to-day basis in R.

Everything in R is an object.

R has 6 basic data types.

* character
* numeric (real or decimal)
* integer
* logical
* complex

R’s base data structures can be organized by their dimensionality (1d, 2d, or nd) and whether they’re homogeneous (all contents must be of the same type) or heterogeneous (the contents can be of different types). This gives rise to the five data types most often used in data analysis:

|  |  |  |
| --- | --- | --- |
|  | Homogeneous | Heterogeneous |
| 1d | Atomic vector | List |
| 2d | Matrix | Data frame |
| nd | Array |  |

>>>First began with reading the csv file

> data=read.csv("C:/Users/Deepali Attavar/Downloads/Diamonds.csv")

> data

ID Carat Color Clarity Depth PricePerCt TotalPrice

1 1 1.08 E VS1 68.6 6693.3 7228.8

2 2 0.31 F VVS1 61.9 3159.0 979.3

3 3 0.31 H VS1 62.1 1755.0 544.1

4 4 0.32 F VVS1 60.8 3159.0 1010.9

5 5 0.33 D IF 60.8 4758.8 1570.4

6 6 0.33 G VVS1 61.5 2895.8 955.6

7 7 0.35 F VS1 62.5 2457.0 860.0

8 8 0.35 F VS1 62.3 2457.0 860.0

9 9 0.37 F VVS1 61.4 3402.0 1258.7

10 10 0.38 D IF 60.0 5062.5 1923.8

11 11 0.38 E VVS2 61.5 3496.5 1328.7

12 12 0.38 F IF 61.8 4252.5 1616.0

13 13 0.39 D VVS1 61.7 4158.0 1621.6

14 14 0.39 D VS1 61.7 2983.5 1163.6

15 15 0.39 F IF 61.3 3969.0 1547.9

16 16 0.39 F VVS1 61.1 3422.3 1334.7

17 17 0.40 D VVS2 59.7 3422.3 1368.9

18 18 0.40 D VVS2 59.6 3422.3 1368.9

19 19 0.40 E VS1 62.2 2937.6 1175.0

20 20 0.40 F VVS2 60.8 2983.5 1193.4

21 21 0.40 F VVS2 61.7 3121.2 1248.5

22 22 0.40 G VVS1 61.7 3213.0 1285.2

23 23 0.41 E IF 61.5 4039.2 1656.1

.

.

.

.

[ reached 'max' / getOption("max.print") -- omitted 209 rows

**->1.1) Some of the Operations done:**

>>> The function **class** prints the vector of names of classes an object inherits from.

> class(data)

[1] "data.frame"

> class(data$ID)

[1] "integer"

> class(data$Carat)

[1] "numeric"

> class(data$Color)

[1] "character"

> class(data$Color)

[1] "character"

> class(data$Clarity)

[1] "character"

> class(data$PricePerCt)

[1] "numeric"

**1.2) Vectors**

>>>**Vector** is a basic data structure in R. It contains element of the same type. The data types can be logical, integer, double, character, complex or raw.

> price=c(data$PricePerCt)

>>>**Accessing** **element in vector.** Vector elements are accessed using indexing vectors, which can be numeric, character or logical vectors. To get the 9th element of the price vector: price[9].

> a=price[9]

> a

[1] 3402

>>>**Indexing using a colon**

> a=price[2:5]

> a

[1] 3159.0 1755.0 3159.0 4758.8

>>> **Logical indexing**

> a=price[price>15000]

> a

[1] 15525.0 17367.8 16023.2 16409.3 17671.5 17313.8 16852.1

>>> **head()** function in R takes argument “n” and returns the first n rows of a dataframe or matrix, by default it returns first 6 rows.

> head(price,10)

[1] 6693.3 3159.0 1755.0 3159.0 4758.8

[6] 2895.8 2457.0 2457.0 3402.0 5062.5

>>**sort()** (or order()) a vector or factor (partially) into ascending or descending order.

> sort(head(price,10))

[1] 1755.0 2457.0 2457.0 2895.8 3159.0

[6] 3159.0 3402.0 4758.8 5062.5 6693.3

> sort(head(price,10),decreasing=TRUE)

[1] 6693.3 5062.5 4758.8 3402.0 3159.0

[6] 3159.0 2895.8 2457.0 2457.0 1755.0

>>>**Is Something A Vector**

> is.vector(price)

[1] TRUE

> is.matrix(price)

[1] FALSE

> is.array(price)

[1] FALSE

**1.3) Arrays**

>>> **Arrays** are the R data objects which can store data in more than two dimensions.

> b=array(head(price,10),dim=c(2,5))

> b

[,1] [,2] [,3] [,4] [,5]

[1,] 6693.3 1755 4758.8 2457 3402.0

[2,] 3159.0 3159 2895.8 2457 5062.5

> is.array(b)

[1] TRUE

>>>**Finding the dimension,** length of rows, columns and array

> dim(b)

[1] 2 5

> nrow(b)

[1] 2

> ncol(b)

[1] 5

> length(b)

[1] 10

>>>**Accessing elements** in the array

> b[,1]

[1] 6693.3 3159.0

> b[2,]

[1] 3159.0 3159.0 2895.8 2457.0 5062.5

> b[2,1]

[1] 3159

>>>**Adding the rows and columns** of the array

> apply(b, c(1), sum)

[1] 19066.1 16733.3

> apply(b, 2 , sum)

[1] 9852.3 4914.0 7654.6 4914.0 8464.5

**1.4) Matrix**

>>> **Matrix** is a two dimensional data structure in R programming. Matrix is similar to vector but additionally contains the dimension attribute.

>x=matrix(head(price,12),nrow=4,ncol=3,byrow=TRUE)

> x

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

[1,] 6693.3 3159.0 1755.0

[2,] 3159.0 4758.8 2895.8

[3,] 2457.0 2457.0 3402.0

[4,] 5062.5 3496.5 4252.5

>>>**finding the sum of the column and row**

> colSums(x)

[1] 17371.8 13871.3 12305.3

> rowSums(x)

[1] 11607.3 10813.6 8316.0 12811.5

>>>**naming the column and row** using using colnames() and rownames()

> colnames(x)=c("COL1","COL2","COL3")

> rownames(x)=c("ROW1","ROW2","ROW3","ROW4")

> x

COL1 COL2 COL3

ROW1 6693.3 3159.0 1755.0

ROW2 3159.0 4758.8 2895.8

ROW3 2457.0 2457.0 3402.0

ROW4 5062.5 3496.5 4252.5

**>>>1.5)** **Some more operations.....**

>>>**max** returns the position of the element with the maximal value in a vector.

> max(data$PricePerCt)

[1] 17671.5

>>>**is. element()** function in R Language is used to check if elements of first Objects are present in second Object or not.

> is.element(60,data$Depth)

[1] TRUE

>>>**Identical function** in R is one of the best ways to test whether two objects are exactly equal. R Identical function, returns TRUE when two objects are equal and it returns FALSE when two objects are not equal.

> identical(data$PricePerCt,data$TotalPrice)

[1] FALSE

>>>**setequal() function** in R Language is used to check if two objects are equal. This function takes two objects like Vectors, dataframes, etc. as arguments and results in TRUE or FALSE, if the Objects are equal or not.

> setequal(data$PricePerCt,data$TotalPrice)

[1] FALSE

Difference between identical and setequal comes in when:

a) there are duplicated values;

b) there is different ordering.

>>>**chartr() function** in R Language is used to do string substitutions. It replaces all the matches of the existing characters of a string with the new characters specified as argument.

> x=chartr('V','A',data$Clarity[1])

> x

[1] "AS1"

>>>**casefold()** converts a character vector to upper or lower case, depending on the value of the parameter upper .

> x=casefold(data$Clarity[1])

> x

[1] "vs1"

**2. Operations on Date**

R provides several options for dealing with date and date/time data. The builtin as.Date function handles dates (without times);

as.Date('2020-11-30')

[1] "2020-11-30"

> as.Date('2020/12/1')

[1] "2020-12-01"

|  |  |
| --- | --- |
| Code | Value |
| %d | Day of the month (decimal number) |
| %m | Month (decimal number) |
| %b | Month (abbreviated) |
| %B | Month (full name) |
| %y | Year (2 digit) |
| %Y | Year (4 digit) |

> as.Date('12/1/2001',format='%m/%d/%Y')

[1] "2020-12-01"

>as.Date('May 26, 2001',format='%B %d, %Y')

[1] "2001-05-26"

>as.Date('24NOV20',format='%d%b%y')

[1] "2020-11-24"

>>>We can even access today’s date:

> today=Sys.Date()

> format(today, format="%B %d %Y")

[1] "November 29 2020"

**3&4. Conditional Statements and Loops on Vector, Dataframe, Array, Matrix**

a) to find the number of diamonds of the best grade(ie. D) in the dataset

code:

> j=0

> for(i in data$Color)

+ {

+ if(i=='D'||i=='E'||i=='F')

+ {

+ j=j+1

+ }

+ }

> print(paste(j," number of colorless diamonds are available in the dataset"))

[1] "221 number of colorless diamonds are available in the dataset"

b) to find the cheapest best grade diamond

code:

> for(k in 1:nrow(data))

+ {

+ i=data[k,"Color"]

+ j=data[k,"TotalPrice"]

+ id=data[k,"ID"]

+ if(j<2000&& i=="D")

+ {

+ print(paste("no.",id,"is cheap as its price is ",j,"and is has the best color->",i))

+ }

+ }

[1] "no. 5 is cheap as its price is 1570.4 and is has the best color-> D"

[1] "no. 10 is cheap as its price is 1923.8 and is has the best color-> D"

[1] "no. 13 is cheap as its price is 1621.6 and is has the best color-> D"

[1] "no. 14 is cheap as its price is 1163.6 and is has the best color-> D"

[1] "no. 17 is cheap as its price is 1368.9 and is has the best color-> D"

[1] "no. 18 is cheap as its price is 1368.9 and is has the best color-> D"

[1] "no. 28 is cheap as its price is 1437.4 and is has the best color-> D"

[1] "no. 33 is cheap as its price is 1856.8 and is has the best color-> D"

[1] "no. 40 is cheap as its price is 1776.6 and is has the best color-> D"

C)different colors of diamond

code:

> d=vector()

> e=vector()

> f=vector()

> g=vector()

> h=vector()

> I=vector()

> for(i in data$Color)

+ {

+ if(i=='D')

+ {

+ d=append(d,i)

+ }

+ if(i=='E')

+ {

+ e=append(e,i)

+ }

+ if(i=='F')

+ {

+ f=append(f,i)

+ }

+ if(i=='G')

+ {

+ g=append(g,i)

+ }

+ if(i=='H')

+ {

+ h=append(g,i)

+ }

+ if(i=='I')

+ {

+ I=append(I,i)

+ }

+ }

> print(paste("no. of Color D:",length(d), "E:" ,length(e), "F:",length(f),"G:",length(g),"H:",length(h),"I:",length(I)))

[1] "no. of Color D: 52 E: 82 F: 87 G: 86 H: 84 I: 12"

**5.String operations, comparisons, grep**

|  |  |  |
| --- | --- | --- |
| **Quantifier** |  | **descriptions** |
| . |  | matches everything |
| **?** |  | item to it’s left is optional |
| **\*** |  | item to its left is matched 0 or more times |
| **+** |  | item to it’s left is matched 1 or more times |
| **{n}** |  | matched exactly n times |
| **{ ,n}** |  | matched at most n times |
| **{n, }** |  | matched atleast n times |
| **{n, m}** |  | matched atleast n times but not more than m times |

**grep:**

Code:

a=grep("S",head(data$Clarity))

b=grep("S",head(data$Clarity),value=TRUE)

c=grepl("S",head(data$Clarity))

d=grep("V\*",head(data$Clarity),value=TRUE)

e=grep("V+",head(data$Clarity),value=TRUE)

f=grep("V{2}",head(data$Clarity),value=TRUE)

g=grep("V{2,}",head(data$Clarity),value=TRUE)

print(a)

print(b)

print(c)

print(d)

print(e)

print(f)

print(g)

Output:

[1] 1 2 3 4 6

[1] "VS1" "VVS1" "VS1" "VVS1" "VVS1"

[1] TRUE TRUE TRUE TRUE FALSE TRUE

[1] "VS1" "VVS1" "VS1" "VVS1" "IF" "VVS1"

[1] "VS1" "VVS1" "VS1" "VVS1" "VVS1"

[1] "VVS1" "VVS1" "VVS1"

[1] "VVS1" "VVS1" "VVS1"

**6. Functions to show recursion**

A function that supports the concept of recursion, i.e., a function that calls itself considered as a recursive function in R. They are like loops, however more efficient than loops.

Let us use the concept of recursion to sort the Carat column of our sample of diamonds.

Code:

v=head(data$Carat)

sort\_vect<-function(vect)

if(length(vect)<=1){

return(vect)

}else{

initial\_element<-vect[1]

prev\_element<-vect[-1]

greater<-prev\_element[prev\_element>initial\_element]

smaller<-prev\_element[prev\_element<=initial\_element]

greater<-sort\_vect(greater)

smaller<-sort\_vect(smaller)

return(c(smaller,initial\_element,greater))

}

print(v)

print(sort\_vect(v))

Output:

[1] 1.08 0.31 0.31 0.32 0.33 0.33

[1] 0.31 0.31 0.32 0.33 0.33 1.08

**7. Usage of functions like all, any, apply, which,match, order, rank, sub, gsub, aggregate.**

**7.1) ‘all’ function:**

The all R function checks in a logical vector, if all values are TRUE.

Code:

v1=data$PricePerCt

v2=data$TotalPrice

x=all(v1 %in% v2)

print(x)

Output:

[1] FALSE

Code:

y1=c(61.7,61.5,62.3)

y2=data$Depth

x=all(y1 %in% y2)

print(x)

Output:

[1] TRUE

**7.2) ‘Any’ Function:**

The any R function checks in a logical vector, if any values are TRUE.

Code:

v1=data$PricePerCt

v2=data$TotalPrice

x=any(v1 %in% v2)

print(x)

Output:

[1] TRUE

**7.3) ‘Apply’ Function:**

Code:

x=data$PricePerCt

y=data$Carat

df=data.frame(x,y)

total=apply(df,2,sum)

Output:

x y

2191071.80 350.83

Code:

low=lapply(head(data$Clarity),tolower)

print(low)

Output:

[[1]]

[1] "vs1"

[[2]]

[1] "vvs1"

[[3]]

[1] "vs1"

[[4]]

[1] "vvs1"

[[5]]

[1] "if"

[[6]]

[1] "vvs1"

**7.4) ‘Which’ Function:**

Code:

a1=which(data$Color=='E')

print(a1)

Output:

[1] 1 11 19 23 24 29 34 35 36 37 42 45 46 47 50

[16] 52 53 57 60 61 62 66 67 68 72 73 74 75 77 81

[31] 82 83 84 86 87 88 94 95 103 108 114 128 129 130 138

[46] 139 140 141 142 143 147 159 168 169 171 176 191 192 210 216

[61] 224 239 241 258 259 260 265 266 267 268 271 272 274 285 286

[76] 287 297 298 299 303 315 332

a2=which(data$Clarity=='VS1')

print(a2)

Output:

[1] 1 3 7 8 14 19 25 34 35 36 37 40 44 45 47

[16] 51 52 56 60 66 72 75 77 81 83 86 89 90 91 92

[31] 93 94 95 97 99 105 117 120 125 128 131 133 135 148 149

[46] 151 152 153 161 165 169 172 173 174 175 177 184 186 188 189

[61] 190 191 193 194 197 198 201 202 203 204 206 207 209 210 212

[76] 213 215 219 220 226 228 230 232 238 245 248 250 253 254 264

[91] 270 271 273 274 275 276 280 285 286 287 289 290 291 294 295

[106] 297 298 300 301 304 308 310 311 312 315 316 323 326 335 343

[121] 350 351

**7.5) ‘Match’ Function:**

Code:

b1=data$PricePerCt

b2=data$TotalPrice

m=match(b1,b2)

print(m)

Output



**7.6) Rank,Sort,Order functions**

Code:

r=rank(head(data$Depth))

s=sort(head(data$Depth))

o=order(head(data$Depth))

print(r)

print(s)

print(o)

Output:

[1] 6.0 4.0 5.0 1.5 1.5 3.0

[1] 60.8 60.8 61.5 61.9 62.1 68.6

[1] 4 5 6 2 3 1

**7.7) Sub. gsub functions:**

Code:

p=head(data$Clarity)

q=sub("V","R",p)

print(p)

print(q)

Output:

[1] "VS1" "VVS1" "VS1" "VVS1" "IF" "VVS1"

[1] "RS1" "RVS1" "RS1" "RVS1" "IF" "RVS1"

p1=head(data$Clarity)

q1=gsub("V","R",p1)

print(p1)

print(q1)

Output:

[1] "VS1" "VVS1" "VS1" "VVS1" "IF" "VVS1"

[1] "RS1" "RRS1" "RS1" "RRS1" "IF" "RRS1"

**7.8) Aggregate function**

Code:

dataframe=data.frame(head(data$Carat),head(data$Color),head(data$PricePerCt))

print(dataframe)

agg=aggregate(head(data$Color),list(head(data$PricePerCt)),max)

print(agg)

Output:

head.data.Carat. head.data.Color. head.data.PricePerCt.

1 1.08 E 6693.3

2 0.31 F 3159.0

3 0.31 H 1755.0

4 0.32 F 3159.0

5 0.33 D 4758.8

6 0.33 G 2895.8

Group.1 x

1 1755.0 H

2 2895.8 G

3 3159.0 F

4 4758.8 D

5 6693.3 E

**8. Plotting PIE, Line, Scatter, Bar using basic R function.**

**8.1) PIE chart**

A pie chart is a type of graph in which a circle is divided into sectors that each represents a proportion of the whole. Pie charts are a useful way to organize data in order to see the size of components relative to the whole, and are particularly good at showing percentage or proportional data.

Code:

data <- read.csv("C:/diamonds.csv")

print(data)

y=table(data$Clarity)

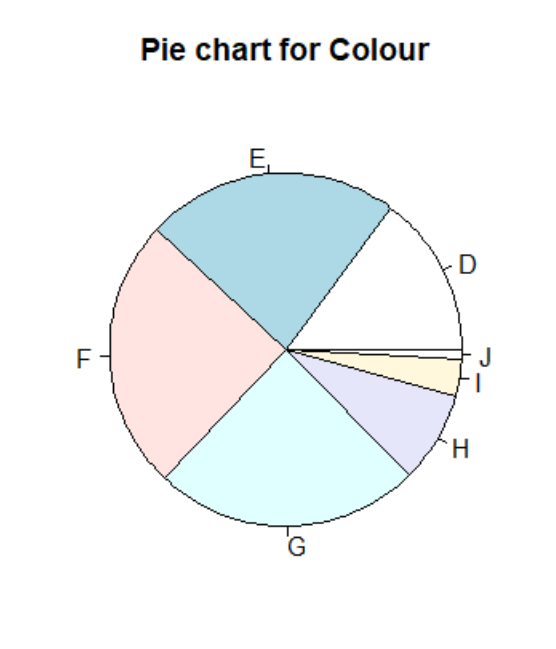
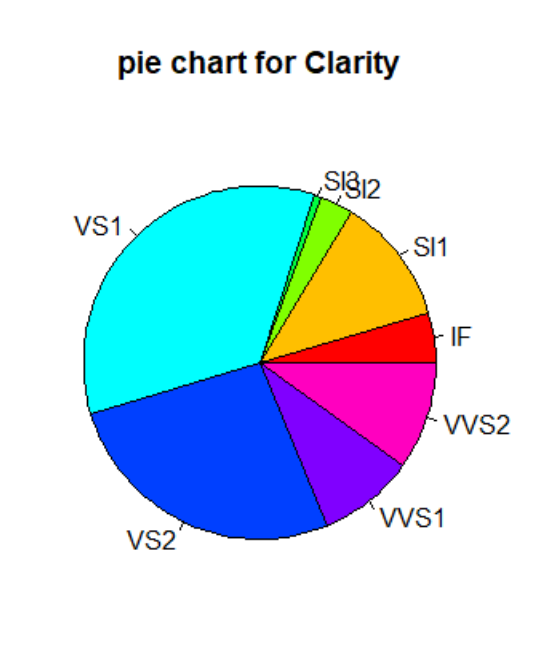
pie(y,main = "pie chart for Clarity",

col = rainbow(length(y)))

x=table(data$Color)

pie(x,main="Pie chart for Colour",

col = rainbow(length(x)))



Here it’s seen that VS1 occurs the

most in our sample followed by

VS2.

And E,F,G are the most dominating colours.

**8.2) LINE PLOT**

A Line plot can be defined as a graph that displays data as points or check marks above a number line, showing the frequency of each value.

Code:

plot(head(data$Carat), type="o",col="blue",

main="frequency of Carat",

xlab="carat",ylab="count")

plot(head(data$Depth),main="frequency of Depth",

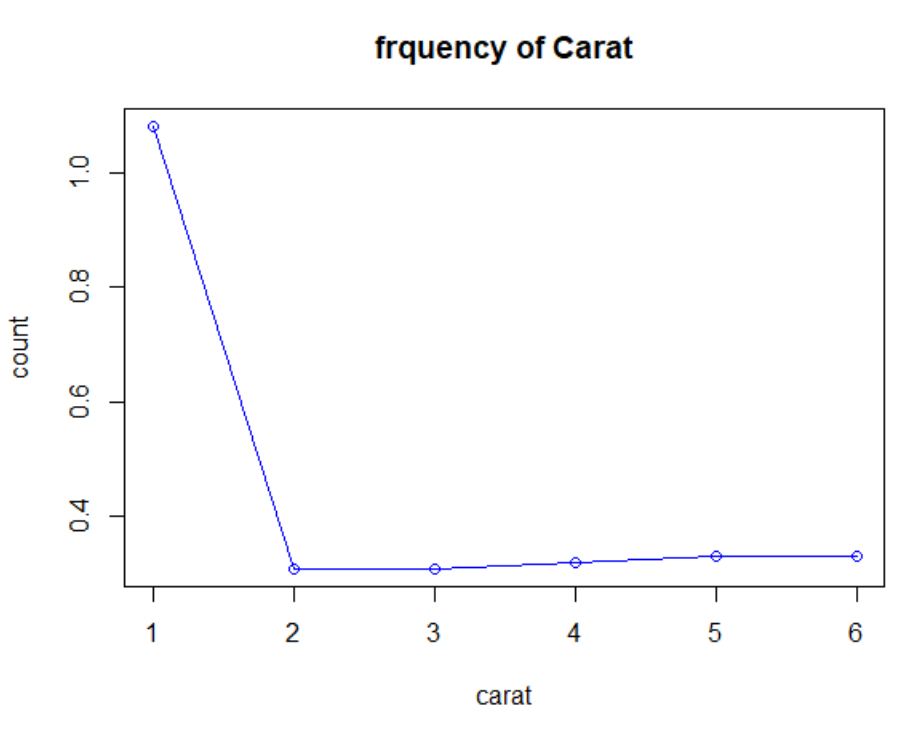
type="o",col="blue"

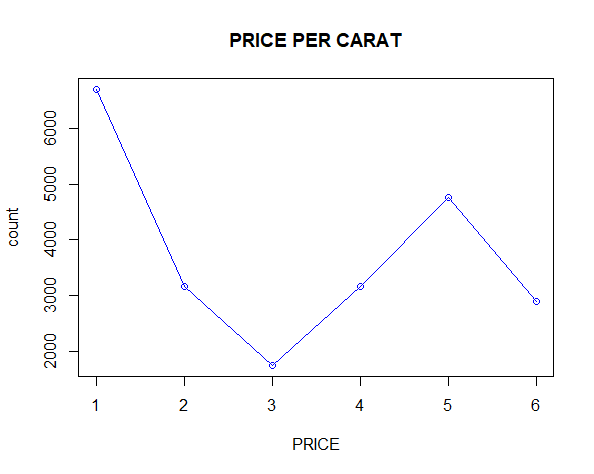
,xlab="Depth",ylab="count")

plot(head(data$PricePerCt),type="o",col="blue",

main="PRICE PER CARAT"

,xlab="PRICE",ylab="count")





**8.3) SCATTER PLOT**

A scatter plot (aka scatter chart, scatter graph) uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables.

Code:

# Relation between carat and pricepercarat

df=data[c('Carat','PricePerCt')]

print(df)

plot(df$Carat,df$PricePerCt,

ylab = "price",

xlab = "Carat",

main="Variation of price of diamonds")

#Relation between depth and total price

df1=data[c('Depth','TotalPrice')]

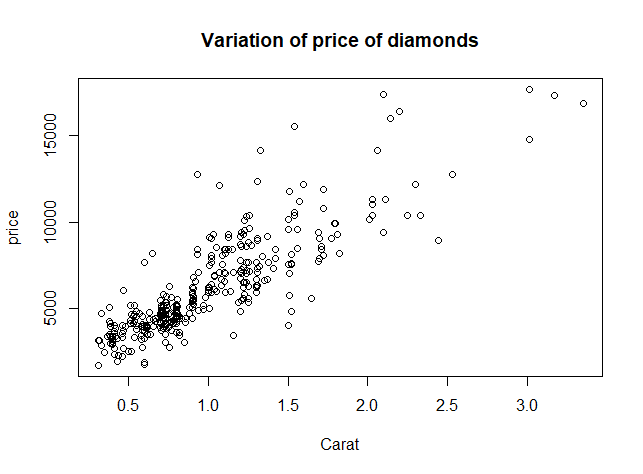
print(df1)

plot(df1$Depth,df1$TotalPrice,

ylab = "TotalPrice",

xlab = "Depth",

main="Depth Vs Price")



From the above scatter plot we can conclude that carat and price per carat are positively correlated.



From the above scatter plot, we can conclude that Total price and Depth are not correlated.

**8.4) BAR CHART**

A bar chart is a way of summarizing a set of categorical data. The bar chart displays data using a number of bars, each representing a particular category. The height of each bar is proportional to a specific aggregation (for example the sum of the values in the category it represents).

Code

x=table(data$Color)

print(x)

barplot(x,

main="Colors",

xlab="Color",ylab="Count",

border="red",col="blue",

density=50

)

y=table(data$Clarity)

print(y)

barplot(y,

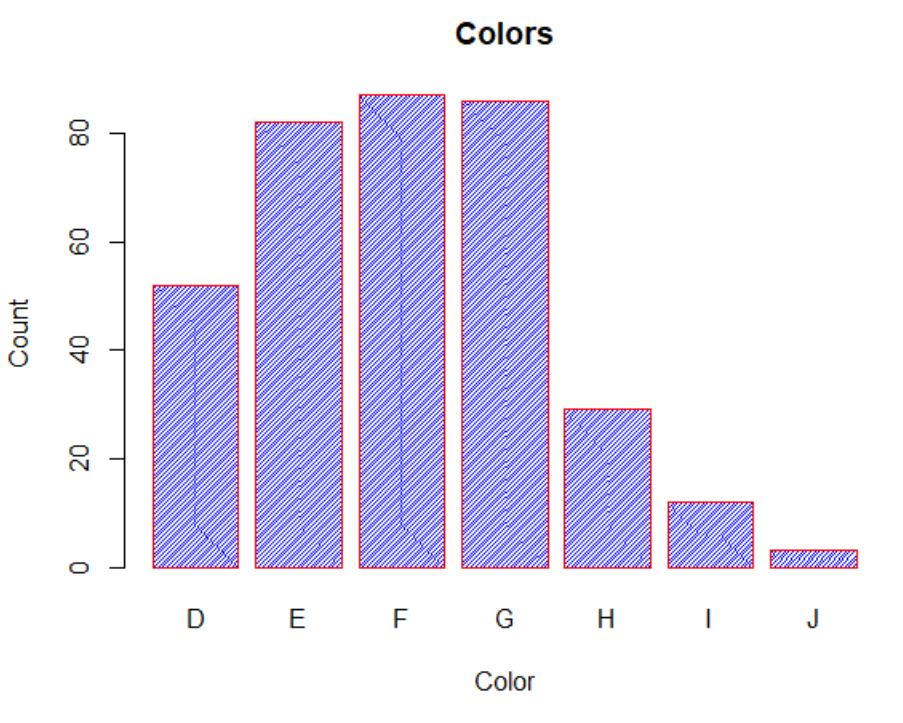
main="Clarity",

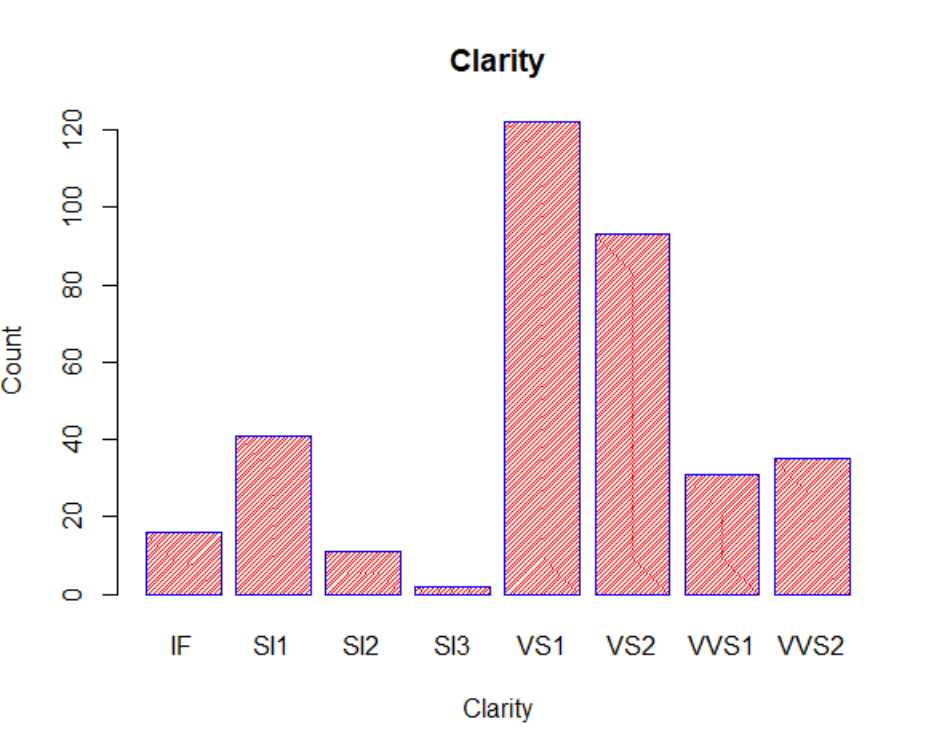
xlab="Clarity",ylab="Count",

border="blue",col="red",

density=50

)





**9.Plotting the graphs using GGPLOT**

* 1. **) Pie chart**

Code:

library(ggplot2)

data=read.csv("C:/Users/Manyatha/Desktop/R/Diamonds.csv")

data

ggplot(data, aes(x="", fill=Color))+

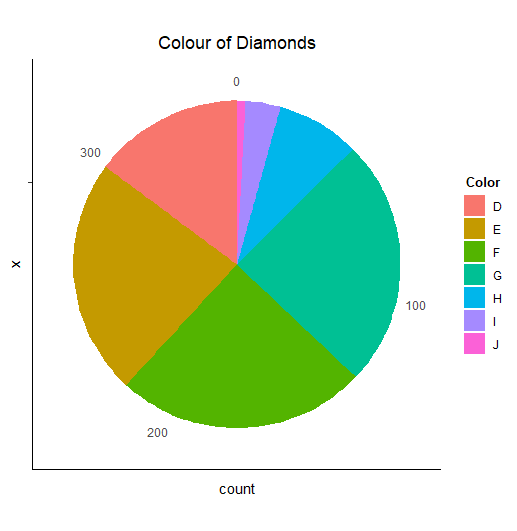
geom\_bar(width = 1)+coord\_polar(theta="y")+

labs(title="Colour of Diamonds")+

theme(plot.title = element\_text(hjust = 0.5),

legend.title = element\_text(hjust = 0.5, face="bold", size = 10))

Output:



From the above pie chart, we can depict that E,F and G are equally dominant whereas J is least dominant.

**9.2) Pie chart along with percentage**

code:

library(dplyr)

x <- data.frame(a=data$Clarity,b=1:length("Clarity"))

x <- x %>%

group\_by(a) %>%

count() %>%

ungroup() %>%

mutate(per=`n`/sum(`n`)) %>%

arrange(desc(a))

x$label <- scales::percent(x$per)

ggplot(data=x)+

geom\_bar(aes(x="", y=per, fill=a), stat="identity", width = 1)+

coord\_polar("y", start=0)+

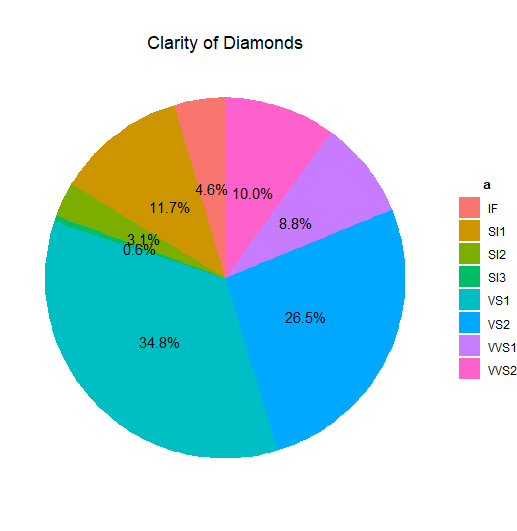
theme\_void()+

geom\_text(aes(x=1, y = cumsum(per) - per/2, label=label))

labs(title="Clarity of Diamonds")+

theme(plot.title = element\_text(hjust = 0.5),

legend.title = element\_text(hjust = 0.5, face="bold", size = 10))



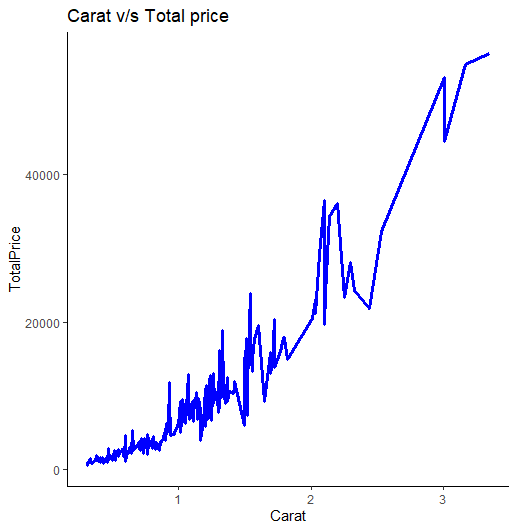
From the above pie chart, we can depict that the clarity of VS1 is the highest ( 34.8%) whereas SI3 is the least (0.6%).

* 1. **Line chart**

ggplot(data, aes(x=Carat, y=TotalPrice)) +

geom\_line(linetype="solid", color="blue", size=1.2)+

labs(title="Carat v/s Total price")



From the above line chart we can tell that the carat increases with the total price of diamonds along with some rising and falling peaks.

* 1. **) Bar plot**

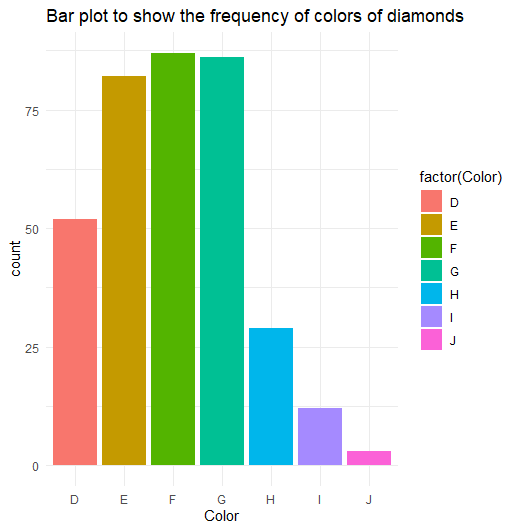
Code:

ggplot(data,aes(Color))+

geom\_bar(aes(fill = factor(Color)))+

theme\_minimal()+

labs(title = "Bar plot to show the frequency of colors of diamonds")



From the above bar plot we can conclude that F is the most occurring colour followed by G and E . The least occurring one is color J.

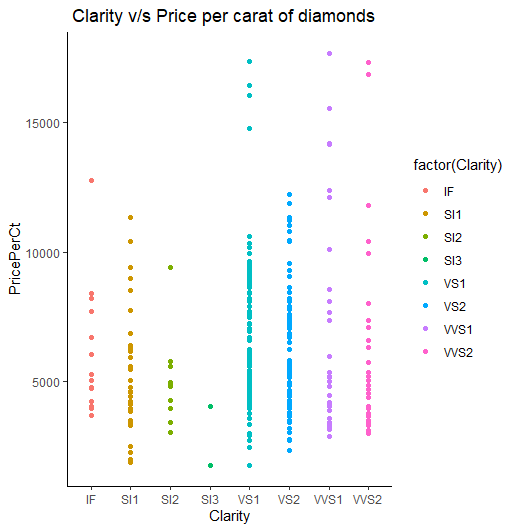
* 1. **Scatter plot**

Code:

ggplot(data, aes(x = Clarity, y = PricePerCt)) +

geom\_point(aes(color = factor(Clarity)))+

labs(title = " Clarity v/s Price per carat of diamonds ")



From the above scatter plot we can depict that the clarity of VS1 is more prominent when compared to others and SI3 is least prominent.

**10.Statistics**

**10.1) Mean**

The **mean** (average) of a data set is found by adding all numbers in the data set and then dividing by the number of values in the set.

Code:

mean(data$TotalPrice)

mean(data$Depth)

mean(data$Carat)

Output:

> mean(data$TotalPrice)

[1] 7450.012

> mean(data$Depth)

[1] 64.38889

> mean(data$Carat)

[1] 0.9995157

**10.2) Median**

The median is the middle value when a data set is ordered from least to greatest or greatest to least.

Code:

median(data$TotalPrice)

median(data$Depth)

median(data$Carat)

Output:

> median(data$TotalPrice)

[1] 4762.8

> median(data$Depth)

[1] 62

> median(data$Carat)

[1] 0.9

**10.3) Variance**

The term **variance** refers to a **statistical** measurement of the spread between numbers in a data set. More specifically, **variance** measures how far each number in the set is from the mean and thus from every other number in the set.

Code:

var(data$TotalPrice)

var(data$Depth)

var(data$Carat)

Output:

> var(data$TotalPrice)

[1] 60542313

> var(data$Depth)

[1] 26.28065

> var(data$Carat)

[1] 0.2439103

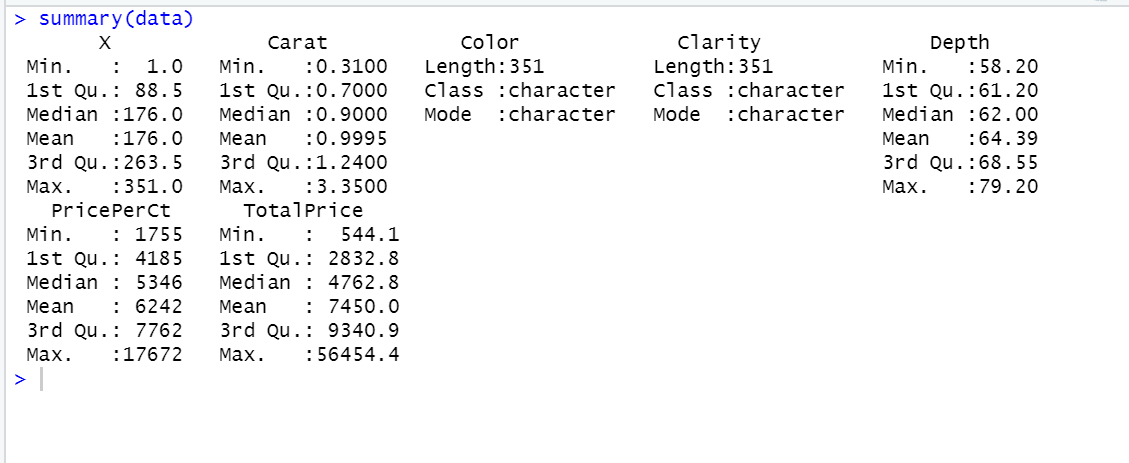
**10.4) Summary of the dataset**

It gives the minimum value, 1st quartile,median, 2nd quartile, maximum value and the mean of all columns in the dataset.

Code:

summary(data)

Output:



**10.5) Box plot**

A **box plot** (also known as **box and whisker plot**) is a type of chart often used in exploratory data analysis to visualise the shape of the distribution and also represent the spread in the data by showing the quartiles of the data along with the extreme values.

1. **Box plot for Depth**

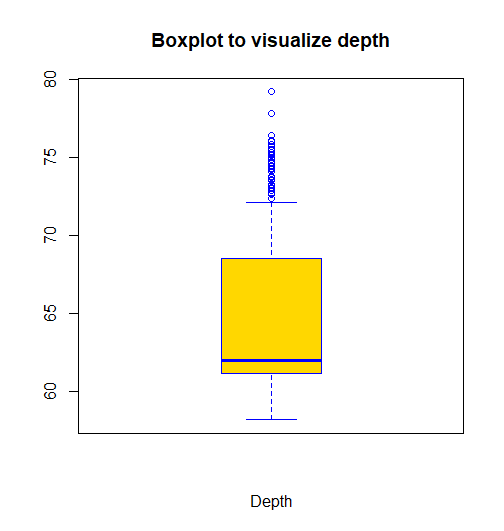
Code:

boxplot(data$Depth, horizontal=FALSE, varwidth=TRUE,

notch=FALSE, range=0.5,

outline=TRUE, boxwex=0.28, border=c("blue"), col=c("gold"),xlab="Depth",

main="Boxplot to visualize depth")



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Min. value | 1st Quartile | Median | Mean | 3rd Quartile | Max. value |
| 58.20 | 61.20 | 62.00 | 64.39 | 68.55 | 79.20 |

1. **Box plot for color v/s carat**

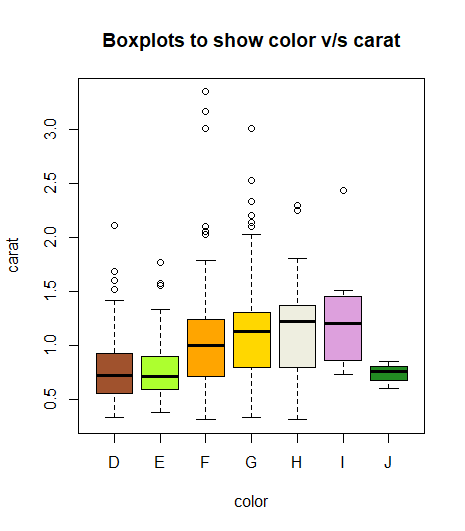
Code:

boxplot(data$Carat ~data$Color,data,notch=F,

col=c('sienna','greenyellow','orange','gold','ivory2','plum','forestgreen'),

main="Boxplots to show color v/s carat",

xlab="color",ylab="carat")



**10.6) Histogram**

A **histogram** is a plot that lets you discover and show the underlying frequency distribution (shape) of a set of continuous data. It helps us to graphically summarize the distribution of a univariate data set.

1. **Histogram for price per carat of diamonds**

Code:

hist(data$PricePerCt,col=c("lightgreen","pink"),

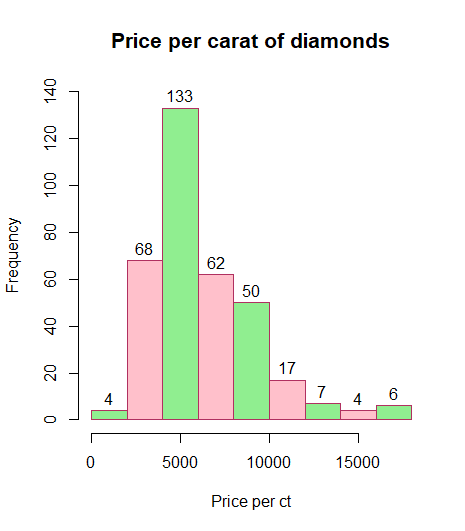
border = "maroon",

xlab = "Price per ct",

main="Price per carat of diamonds",

xlim=c(0,18000),ylim = c(0,140),cex.main=1.3,labels = T )

Output:



From the above histogram, we can conclude that the frequency of the price per carat of diamonds mostly lies between 4000 to 6000.

1. **Histogram for carat of diamonds**

Code:

hist(data$Carat,col=c("gold","skyblue"),

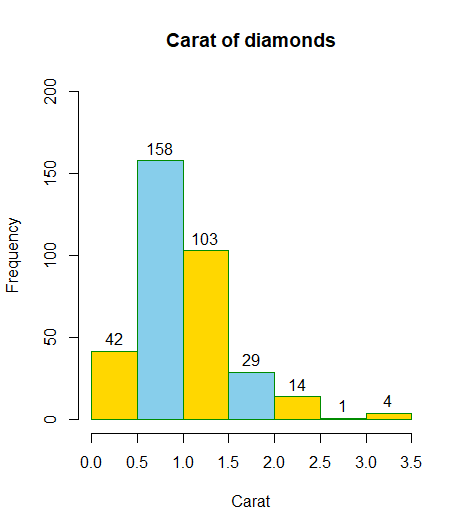
border = "green4",

xlab = "Carat",

main="Carat of diamonds",

xlim=c(0,3.5),ylim = c(0,200),labels = T)

Output:



From the above histogram, we can conclude that the frequency of the carat of diamonds mostly lies in the range of 0.5 to 1.0 and the least between 2.5 to 3.0.

**10.7) Factor**

Factors are the data objects which are used to categorize the data and store it as levels.They are useful in the columns which have a limited number of unique values.

If argument ordered is TRUE, the factor levels are assumed to be ordered.

1. **To Apply the factor function on the color column:**

# Test if color is a factor

print(is.factor(data$Color))

#Apply the factor function on color

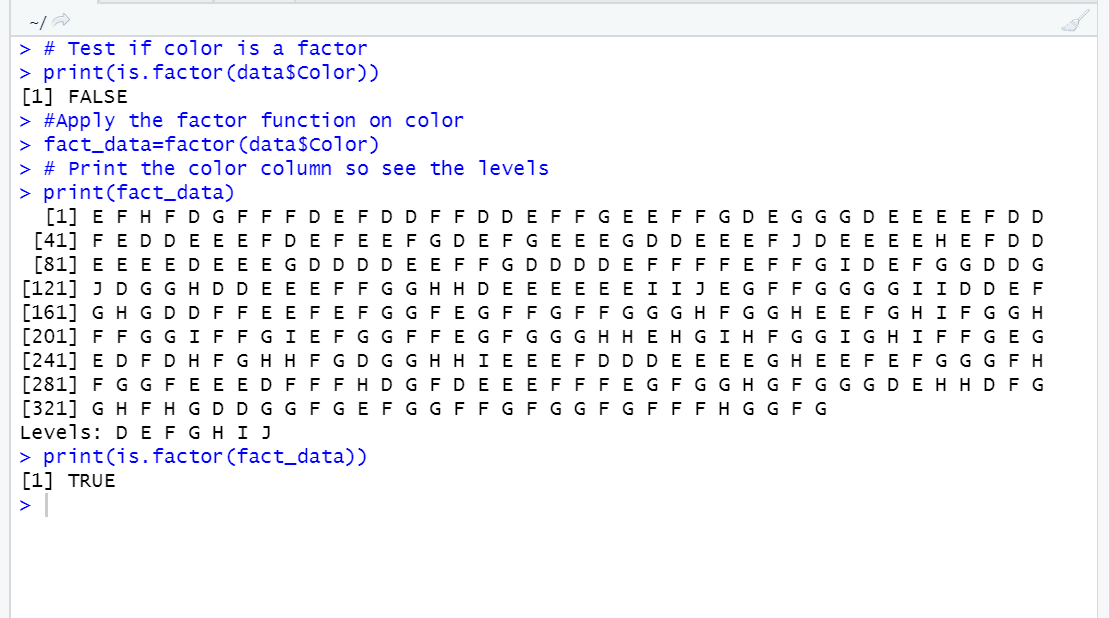
fact\_data=factor(data$Color)

# Print the color column so see the levels

print(fact\_data)

print(is.factor(fact\_data))

Output:



1. **To Apply the factor function on the Clarity column:**

# Test if clarity column is a factor

print(is.factor(data$Clarity))

#Apply the factor function on clarity column

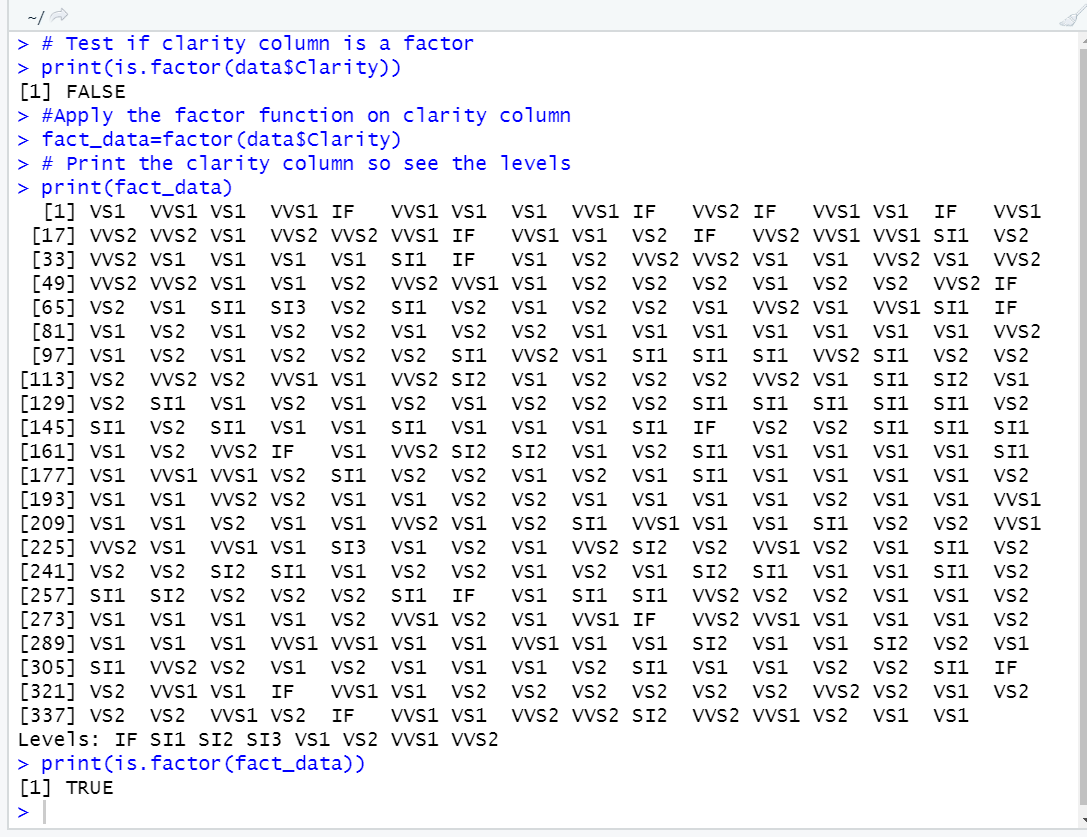
fact\_data=factor(data$Clarity)

# Print the clarity column so see the levels

print(fact\_data)

print(is.factor(fact\_data))

Output:



**11.PROBABILITY DISTRIBUTION FUNCTIONS**

**Normal Distribution:**

In a random collection of data from independent sources, it is generally observed that the distribution of data is normal. Which means, on plotting a graph with the value of the variable in the horizontal axis and the count of the values in the vertical axis we get a bell shaped curve. The center of the curve represents the mean of the data set and is said to be symmetrical about the mean. This is referred to as normal distribution in statistics.

**a) dnorm()**

This function gives height of the probability distribution at each point for a given mean and standard deviation.

Let us consider a sequence of numbers from the column Price per carat of diamonds ranging from 800 to 12,000 which is incremented by 2000 with mean ‘m’ and std. deviation ‘s’.

m=mean(data$PricePerCt)

s=sd(data$PricePerCt)

x=seq(800,12000,length=2000)

y=dnorm(x,mean=m,sd=s)

plot(x,y,type="l",lwd=2,col="red",main="Normal Distribution Curve")

head(y,10)

output:

> m=mean(data$PricePerCt)

> s=sd(data$PricePerCt)

> x=seq(800,12000,length=2000)

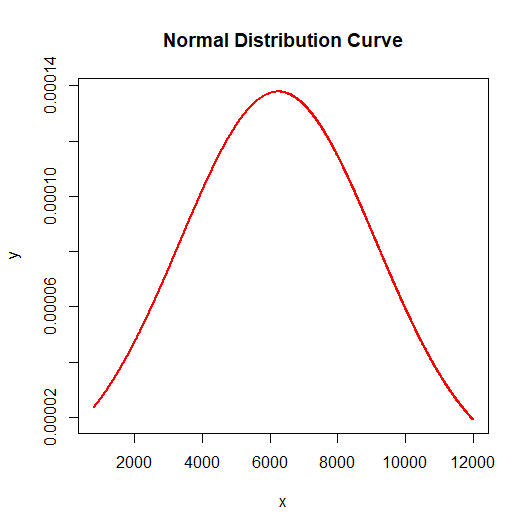
> y=dnorm(x,mean=m,sd=s)

> plot(x,y,type="l",lwd=2,col="red",main="Normal Distribution Curve")

> head(y,10)

[1] 2.355028e-05 2.363605e-05 2.372204e-05 2.380826e-05 2.389470e-05 2.398136e-05

[7] 2.406825e-05 2.415537e-05 2.424271e-05 2.433027e-05



**b) pnorm()**

This function gives the cumulative distribution function.

Let us consider a sequence of numbers from the column Price per carat of diamonds ranging from 800 to 16,000 which is incremented by 2000 with mean ‘m’ and std. deviation ‘s’.

Code:

m=mean(data$PricePerCt)

s=sd(data$PricePerCt)

x=seq(800,16000,length=2000)

y=pnorm(x,mean=m,sd=s)

plot(x,y,type="l",lwd=3,col="green4",main="CDF")

head(y,10)

Output:

> m=mean(data$PricePerCt)

> s=sd(data$PricePerCt)

> x=seq(800,16000,length=2000)

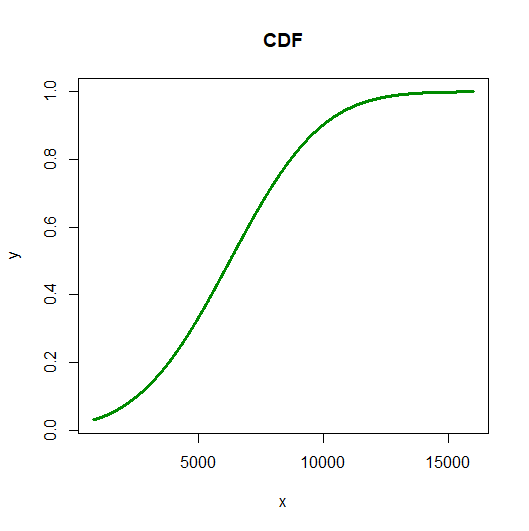
> y=pnorm(x,mean=m,sd=s)

> plot(x,y,type="l",lwd=3,col="green4",main="CDF")

> head(y,10)

[1] 0.03007749 0.03025700 0.03043740 0.03061869 0.03080088 0.03098396 0.03116794

[8] 0.03135282 0.03153861 0.03172531



1. **qnorm()**

It gives the pth quantile of the binomial distribution.

To compute the values of depth for 10th,60th,95th and 99th quantiles respectively having the mean ‘m’ and std.deviation ‘s’.

Code:

m=mean(data$Depth)

s=sd(data$Depth)

qnorm(c(0.10,0.60,0.95,0.99),mean=m,sd=s)

Output:

> m=mean(data$Depth)

> s=sd(data$Depth)

> qnorm(c(0.10,0.60,0.95,0.99),mean=m,sd=s)

[1] 57.81906 65.68766 72.82117 76.31483

rnorm()

This function is used to generate random numbers whose distribution is normal. It takes the sample size as input and generates that many random numbers.

Generate 150 random numbers which are normally distributed for the depth column. We draw a histogram to show the distribution of the generated numbers

Code:

m=mean(data$Depth)

s=sd(data$Depth)

y=rnorm(150,mean=m,sd=s)

hist(y,col=terrain.colors(6),main = "Normal Distribution",ylim=c(0,60),

border="blue",xlab = "Depth",labels = T)

tail(y,10)

> m=mean(data$Depth)

> s=sd(data$Depth)

> y=rnorm(150,mean=m,sd=s)

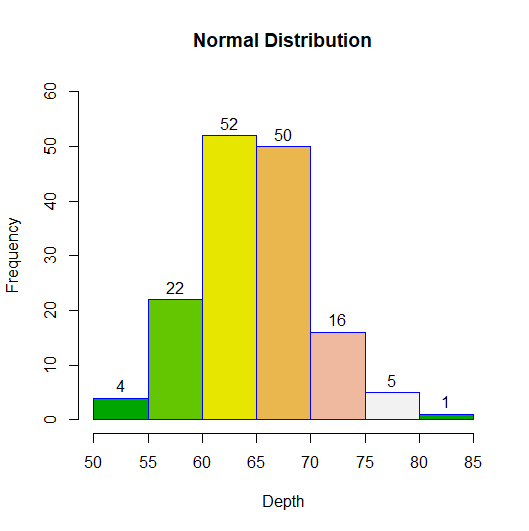
> hist(y,col=terrain.colors(6),main = "Normal Distribution",ylim=c(0,60),

+ border="blue",xlab = "Depth",labels = T)

> tail(y,10)

[1] 68.47808 59.72597 59.84184 64.34563 65.73505 62.16064 54.16319 64.11858 64.28350

[10] 65.84246



**12.Hypothesis testing**

**Hypothesis testing** in statistics is a way for you to test the results of a survey or experiment to see if you have meaningful results.

Suppose the diamond merchant claims that the mean price per carat of diamonds is more than $3250. In a sample of 30 diamonds, it was found that their average was $3472.4. Assume the population standard deviation is $2895.4. At 0.05 significance level, can we reject the claim by the diamond merchant?

Sol.

H0: μ>= 3250 (Null hypothesis)

H1: μ < 3250 (Alternate hypothesis)

x ̅ = 3472.4

μ\_0 = 3250

σ = 2895.4

Code:

xbar = 3472.4 # sample mean

mu0 = 3250 # hypothesized value

sigma = 2895.4 # population standard deviation

n = 30 # sample size

z = (xbar-mu0)/(sigma/sqrt(n))

z

alpha = 0.05

1. using Z test statistic approach

z.alpha = qnorm(1-alpha)

-z.alpha # critical value

if(z < -z.alpha){

print("Rejected")

} else {

print("Accepted")

}

1. using p value approach

pval = pnorm(z)

pval

if(pval < alpha){

print("Rejected")

} else {

print("Accepted")

}

Output:

> xbar = 3472.4 # sample mean

> mu0 = 3250 # hypothesized value

> sigma = 2895.4 # population standard deviation

> n = 30 # sample size

> z = (xbar-mu0)/(sigma/sqrt(n))

> z

[1] 0.4207139

> alpha = 0.05

> z.alpha = qnorm(1-alpha)

> -z.alpha # critical value

[1] -1.644854

> if(z < -z.alpha){

+ print("Rejected")

+ } else {

+ print("Accepted")

+ }

[1] "Accepted"

> pval = pnorm(z)

> pval

[1] 0.663018

> if(pval < alpha){

+ print("Rejected")

+ } else {

+ print("Accepted")

+ }

[1] "Accepted"

From the z-test, we can tell that the test statistic 0.4207 is greater than the critical value of -1.6448 .

Hence, at 0.05 significance level, we accept the claim that the mean price per carat of diamonds is more than $3250.

From the p-value approach, we can tell that the value of p (0.663) is greater than 0.05 and hence the null hypothesis is accepted.

**Using qqnorm,**

#qqnorm(): produces a normal QQ (Quantile-Quantile) plot of the variable

#qqline(): adds a reference line

x=data$PricePerCt

m=mean(x)

s=sd(x)

x1=rnorm(500,m,s)

qqnorm(x1,xlim=c(-2,2))

qqline(x1,col="red",lwd=2)

