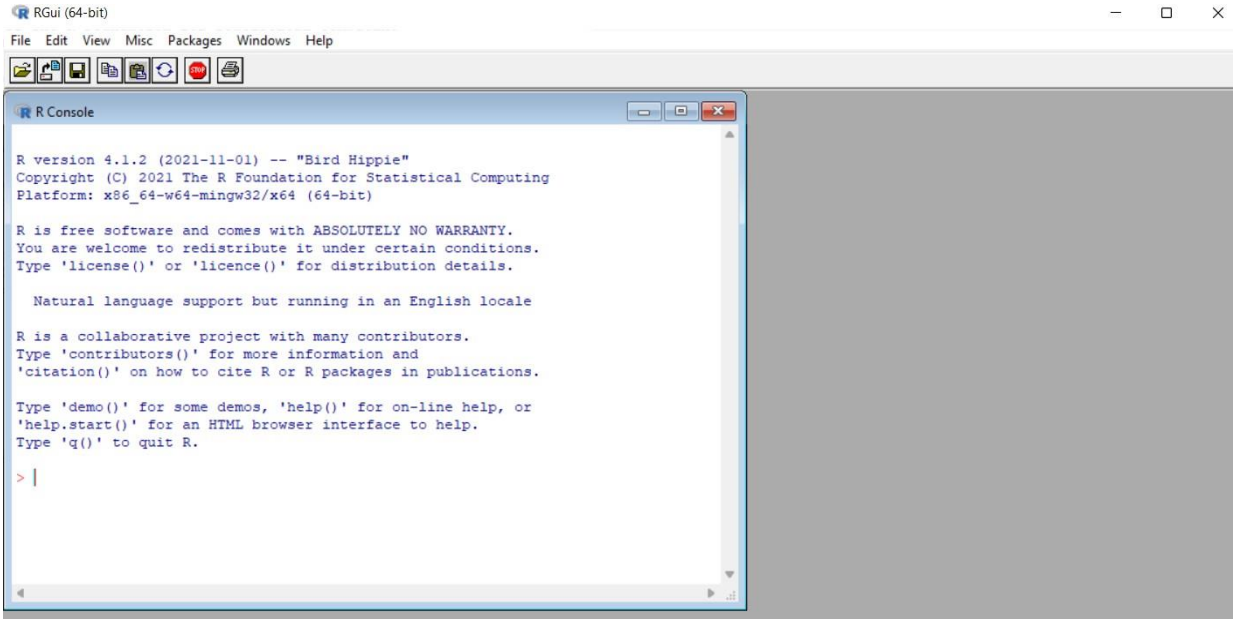
LABORATORY EXERCISE BOOK
SUBJECT: APL LAB (CS -215)
B.Tech CSE- 4th SEMESTER

Submitted By: Tushar Rathi
Roll No: 2012174
Brach and Section: CSE [Section-B]

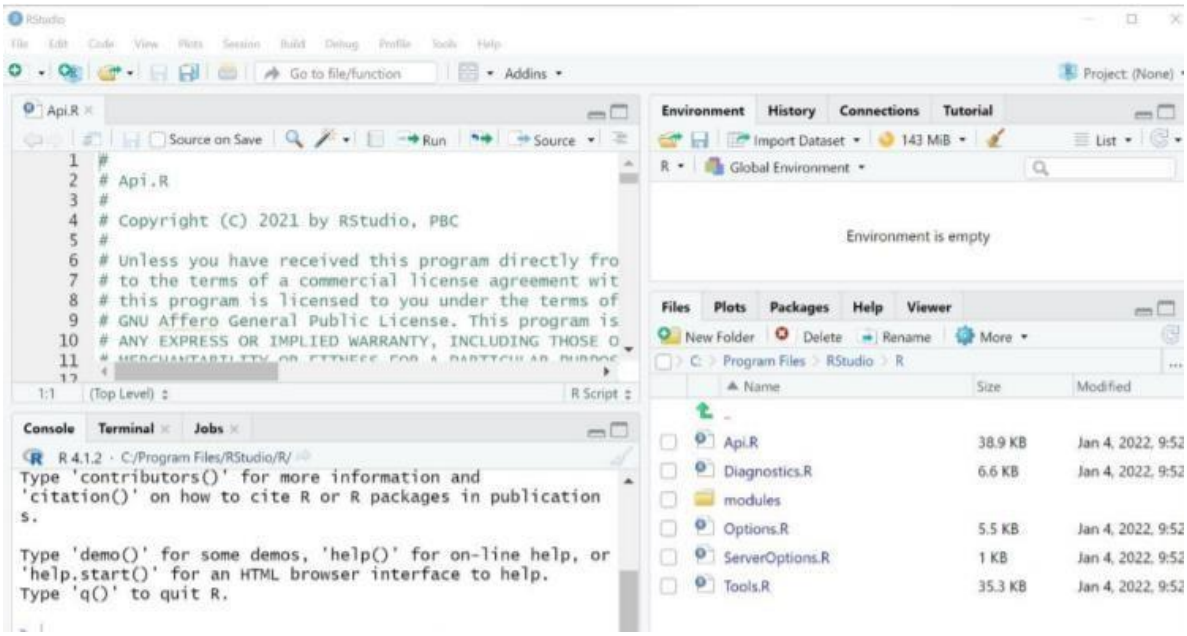
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,
NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR
ASSAM, PIN-788010

LAB ASSIGNMENT A1:

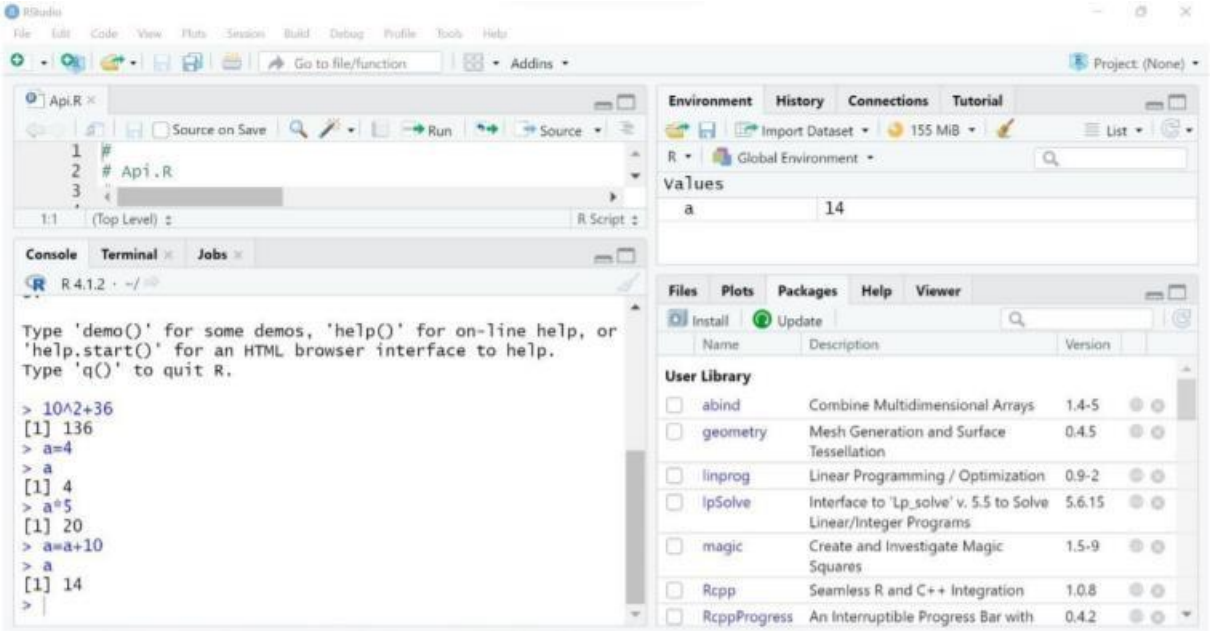
Task 1: INSTALL R AND R STUDIO



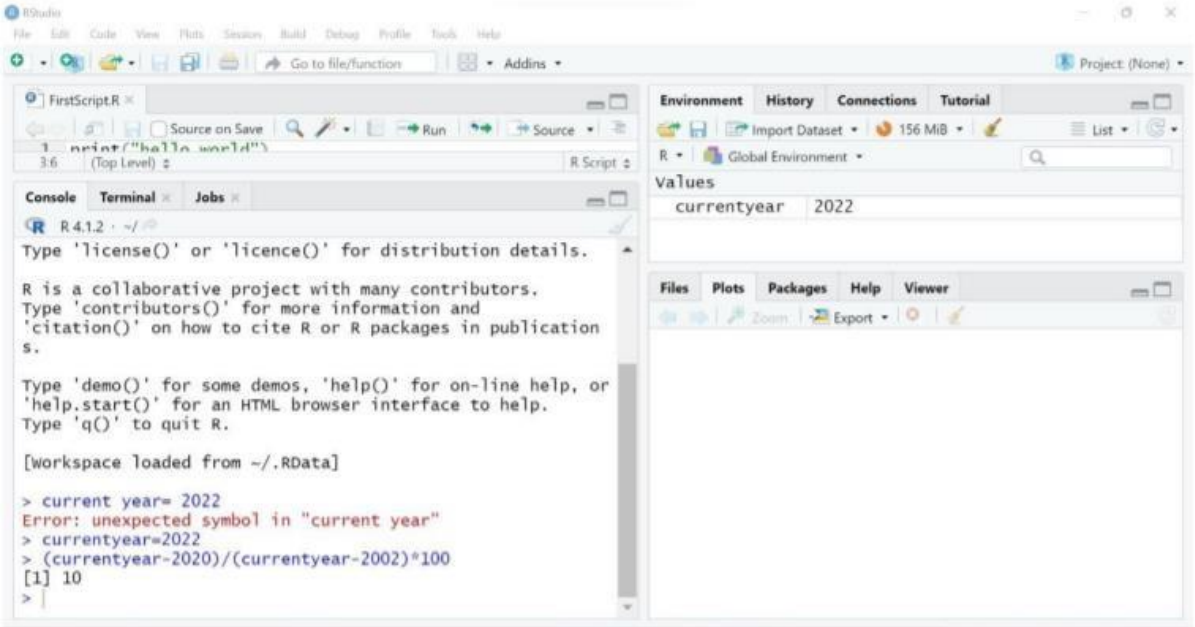
TASK 2: GET FAMILIAR WITH R STUDIO



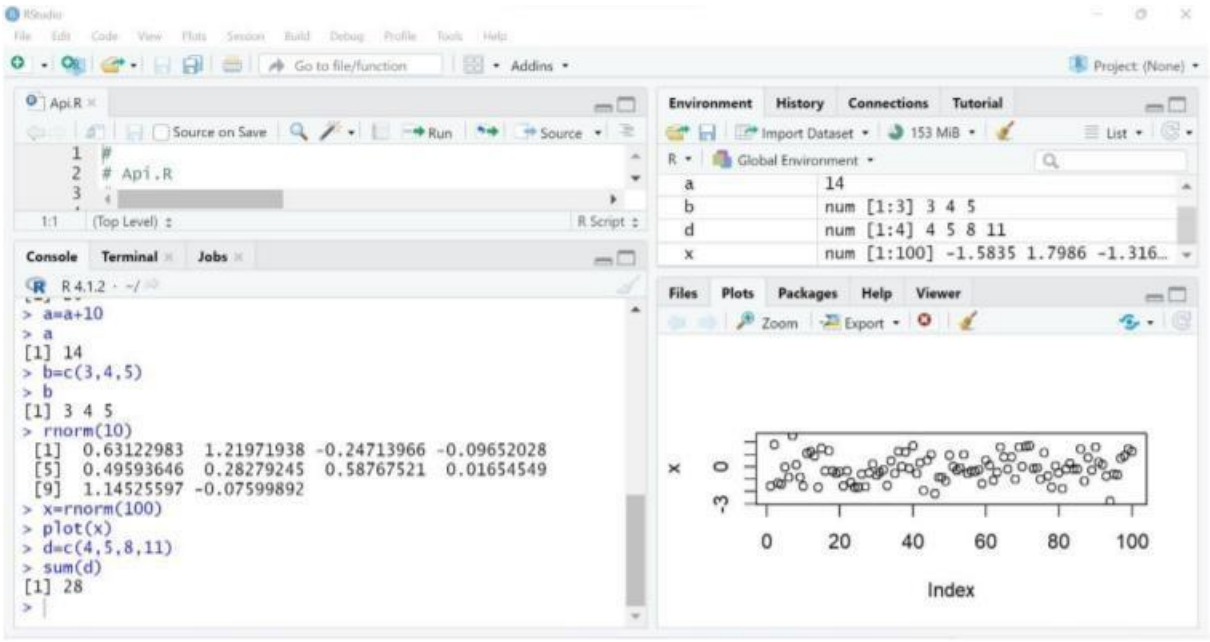
TASK 3: TRY SOME EXAMPLES OF R COMMANDS



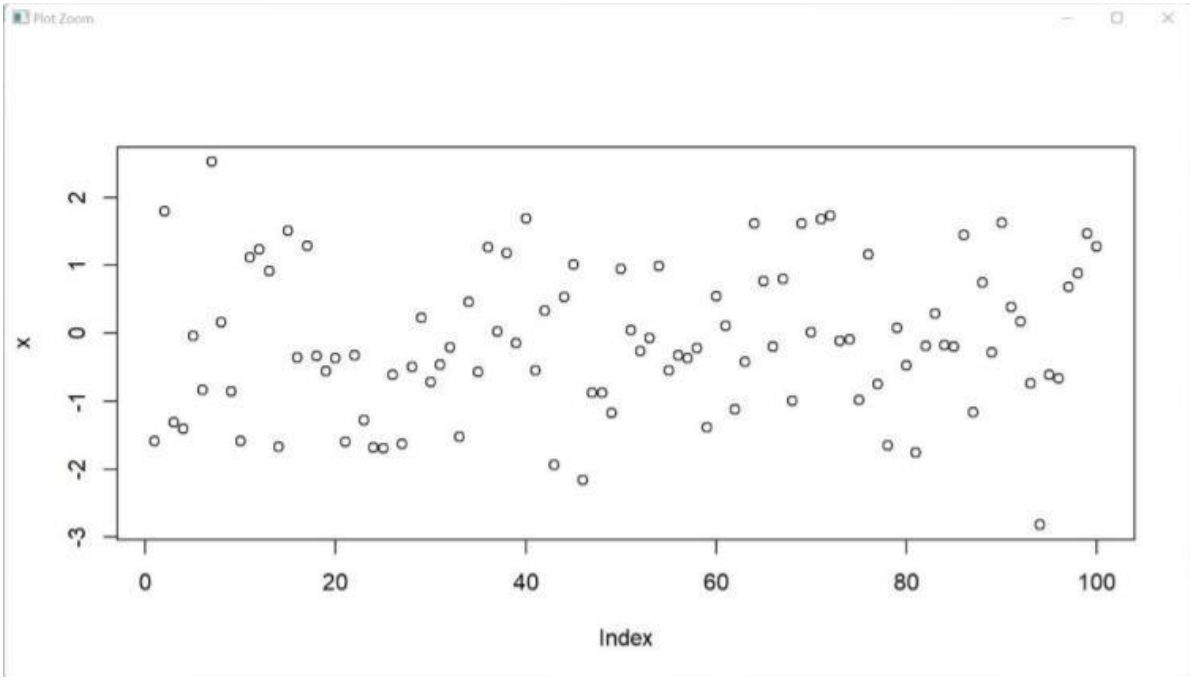
TASK 4: COMPUTING THE PERCENTAGE OF MY LIFE SPENT AT THE INSTITUTE



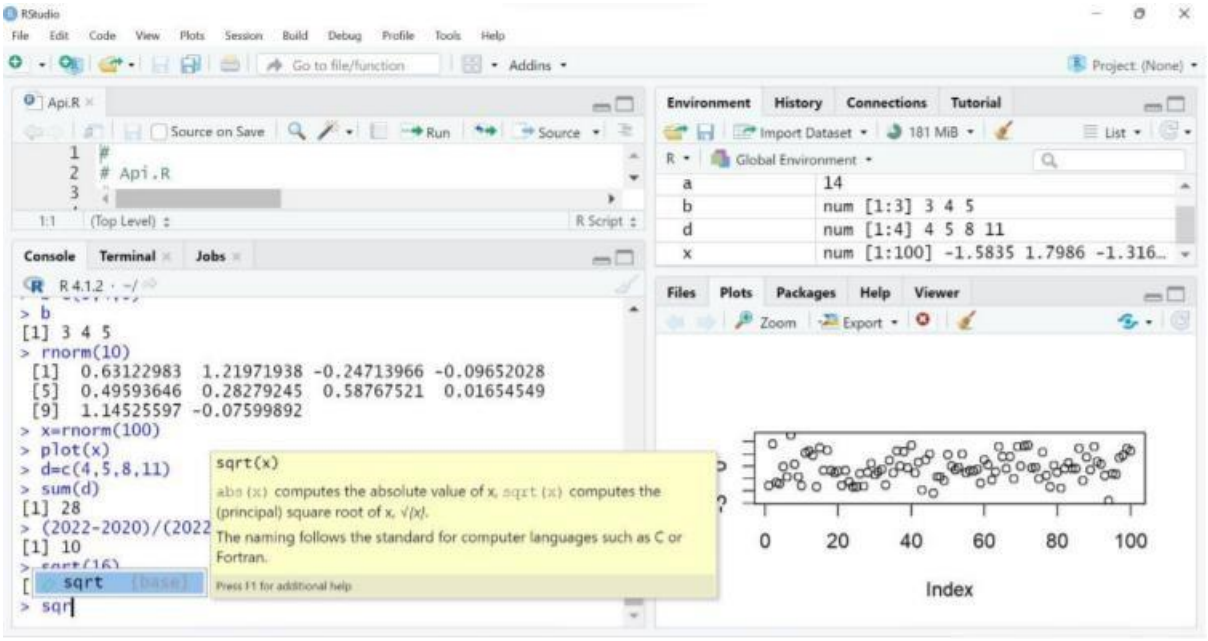
TASK 5: COMPUTING THE SUM OF 4, 5, 8 AND 11 BY COMBIING INTO VECTOR ANDUSING SUM() FUNCTION



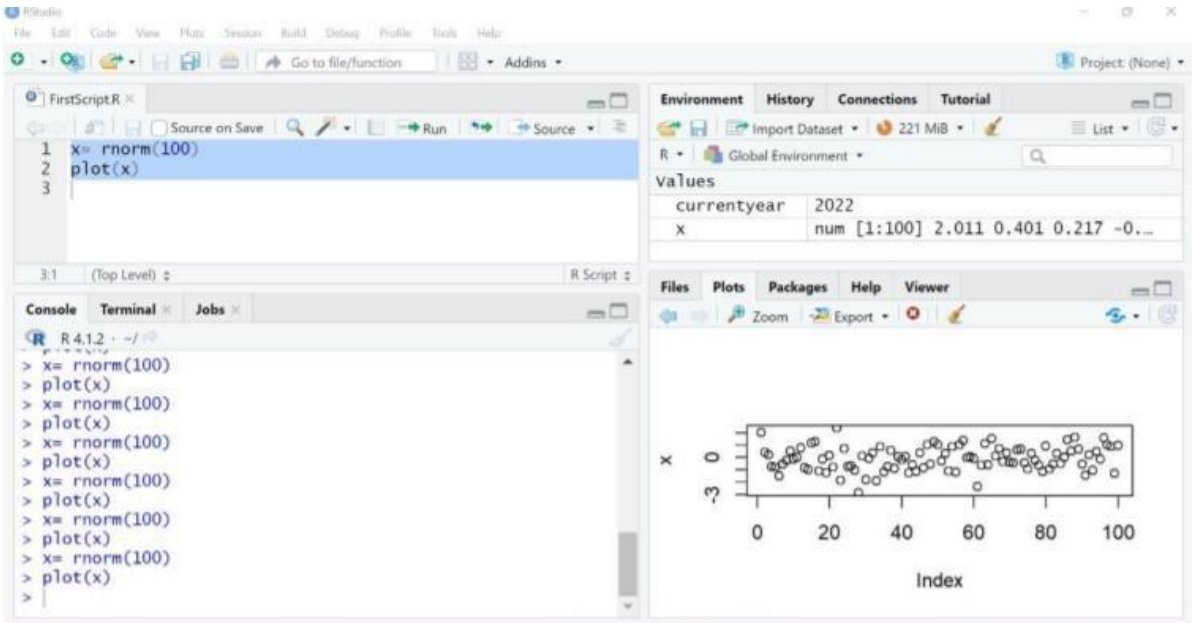
TASK 6: PLOTTING 100 RANDOM NUMBERS



TASK 7: USING SQRT() FUNCTION



TASK 8: FILE CONTAINING R CODE THAT GENERATES 100 RANDOM NUMBERS ANDPLOT THEM (FIRSTSCRIPT.R)



LAB ASSIGNMENT 2:

Q1: Short notes and sample R code snippet for the following constructs

i. Vectors
ii. Lists
iii. Matrices and Data frames
iv. Functions definition and function call
v. Associative arrays
vi. R data object

Ans:

i. Vectors

A vector is a data structure that stores a sequence of data elements of the same basic data types. A vector can have a single element or a sequence of elements that belongs to any of the basic data types like logical, integer, Numeric, etc. Therefore vector data structures are further classified into six types. They are:

Data type	Description	Example
Logical	Takes either a TRUE or FALSE value.	TRUE or FALSE
Integer	Take whole number values positive integers and 0	0, 56,9822,100
Double	Takes decimal numbers	5.67, 2.18
Complex	Takes values with real & imaginary parts.	1+2i, -3+4i
Character	Takes a single character or sequence of words	“A”, “Hello”
Raw	We can convert character value to its raw value using “charToRaw()” function.	charToRaw(“val”)

The function c() function is used to combine a sequence of data elements of the same basic data types in R.
Syntax to create vector= c(<value 1>,<value 2>… <value n>)
Arithmetic operations like addition, subtraction, multiplication and division on the vectors are done element wise.

ii. Lists

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 list=
list_data <- list(c("Red", "Green"), c(21,32,11), TRUE, c(51.23, 119.1))
We can name the elements in the list using this syntax names(list_data) <- c("colours”, “numbers”, “signals”, “decimals”)
We can add, delete and update list elements as shown below. We can add and delete elements only at the end of a list. But we can update any element.

iii. Matrices and Data frames

Matrices:
A matrix is a two-dimensional, homogeneous data structure in R. This means that it has two dimensions, rows and columns. A matrix can store data of a single basic type (numeric, logical, character, etc.). Therefore, a matrix can be a combination of two or more vectors.
We can create matrices using the matrix() function. The syntax of the matrix() function is= matrix(data, byrow, nrow, ncol, dimnames)
The arguments in the matrix function are the following:
1. data – data contains the elements in the R matrix.
2. byrow – byrow is a logical variable. Matrices are by default column-wise. By setting byrow as TRUE, we can arrange the data row-wise in the matrix.
3. nrow – defines the number of rows in the R matrix.

4. ncol – defines the number of columns in the R matrix.
5. dimnames – takes two character arrays as input for row names and column names. We can also name the rows and columns of an R matrix after its creation by using the rownames() or colnames() functions

It is not necessary to specify both nrow and ncol. Only one of the two is required. R calculated the other based on the length of the data.
Data Frame:
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.

Syntax of data frame = data.frame(df, stringsAsFactors = TRUE)
df: It can be a matrix to convert as a data frame or a collection of variables to join stringsAsFactors: Convert string to factor by default

The statistical summary and nature of the data can be obtained by applying summary() function.
We can expand data frame using rbind() and cbind() just like in matrices
iv. Function Definitions and Function Call

Function:
A function is a set of statements organized together to perform a specific task. R has a large number of in-built functions and the user can create their own functions. An R function is created by using the keyword function. The basic syntax of an R function definition is as follows:
Function body
}
The different parts of a function are −
- Function Name − This is the actual name of the function. It is stored in R environment as an object with this name.
- Arguments − An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.
- Function Body − The function body contains a collection of statements that defines what the function does.
- Return Value − The return value of a function is the last expression in the function body to be evaluated.

R has many in-built functions which can be directly called in the program without defining them first. We can also create and use our own functions referred as user defined functions. Simple examples of in-built functions are seq(), mean(), max(), sum(x) and paste(...) etc.

Function Call:

A function call is a request made by a program or script that performs a predetermined function. Most of the computations carried out in R involve the evaluation of functions. We will also refer to this as function invocation. Functions are invoked by name with a list of arguments separated by commas.

Example= mean(1:10)

In this example the function mean was called with one argument, the vector of integers from 1 to 10.

Function calls can have tagged (or named) arguments, as in plot(x, y, pch = 3).

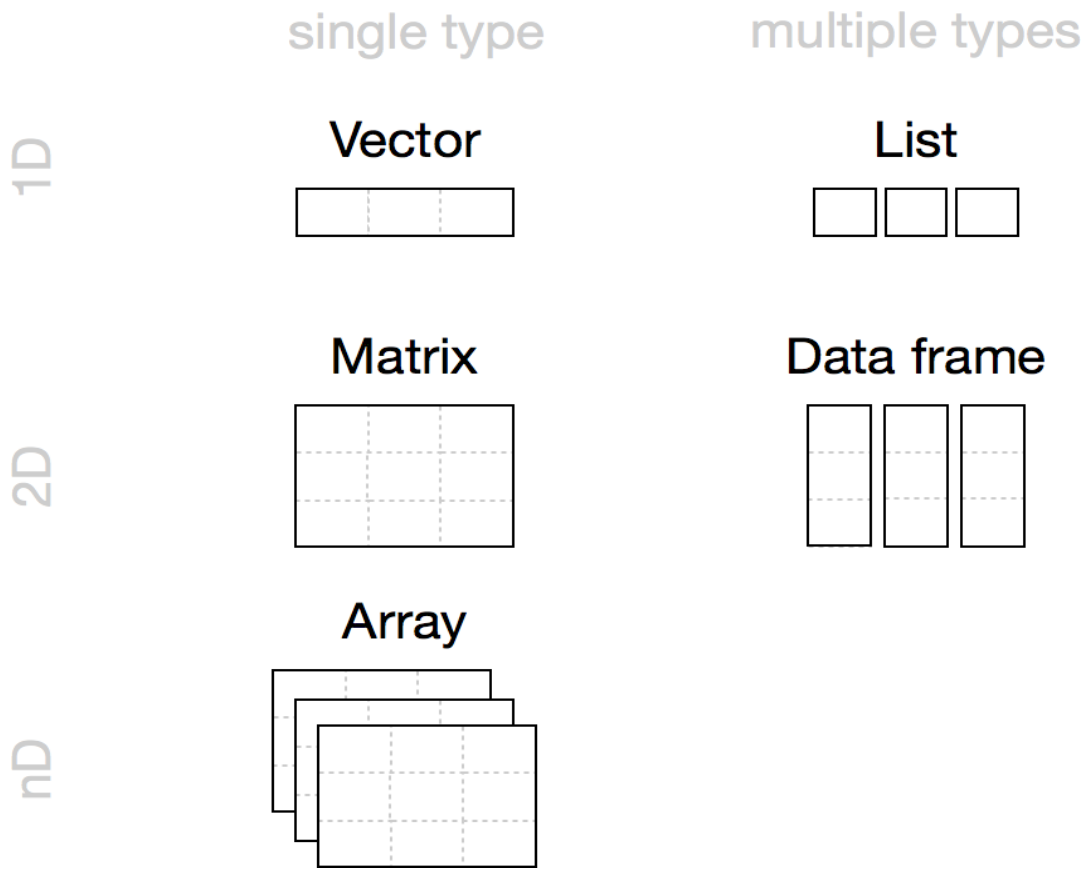
v. Associative arrays

Hash table or associative array, is a well-known key value data structure. To implement the equivalent R, we can use a vector of any type, a list, or an environment.

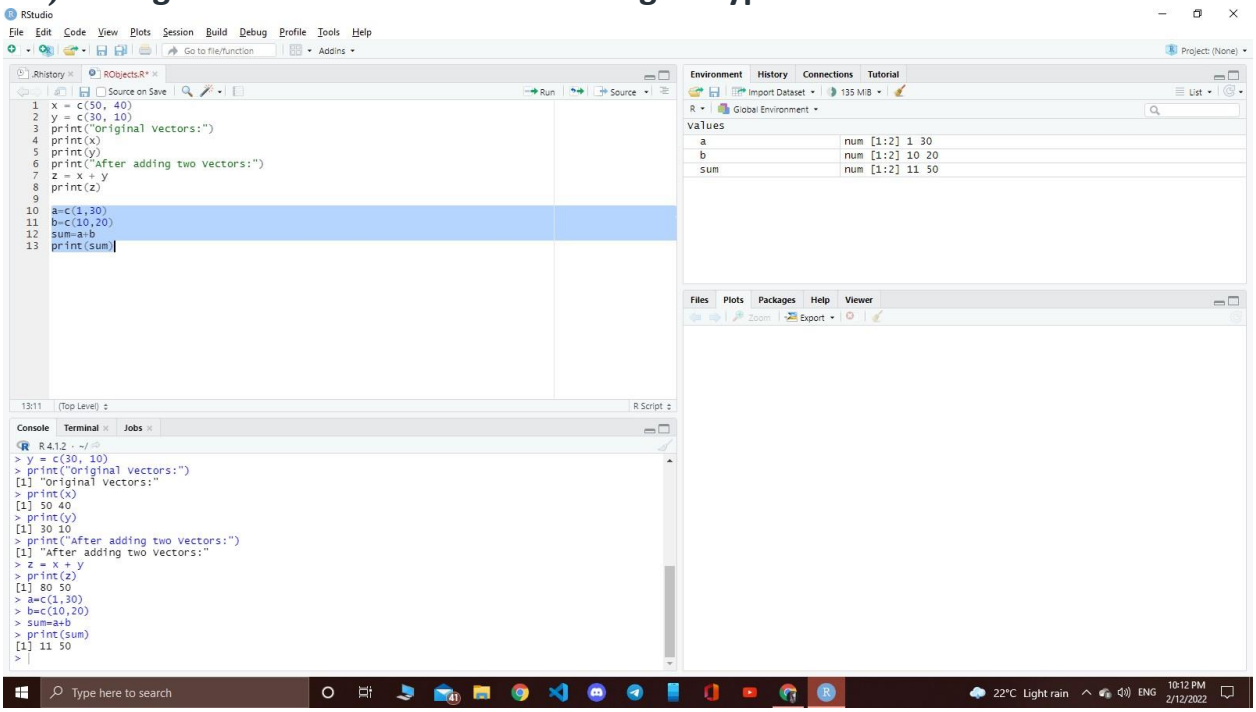
vi. R data objects

There are 5 basic types of objects in the R language:

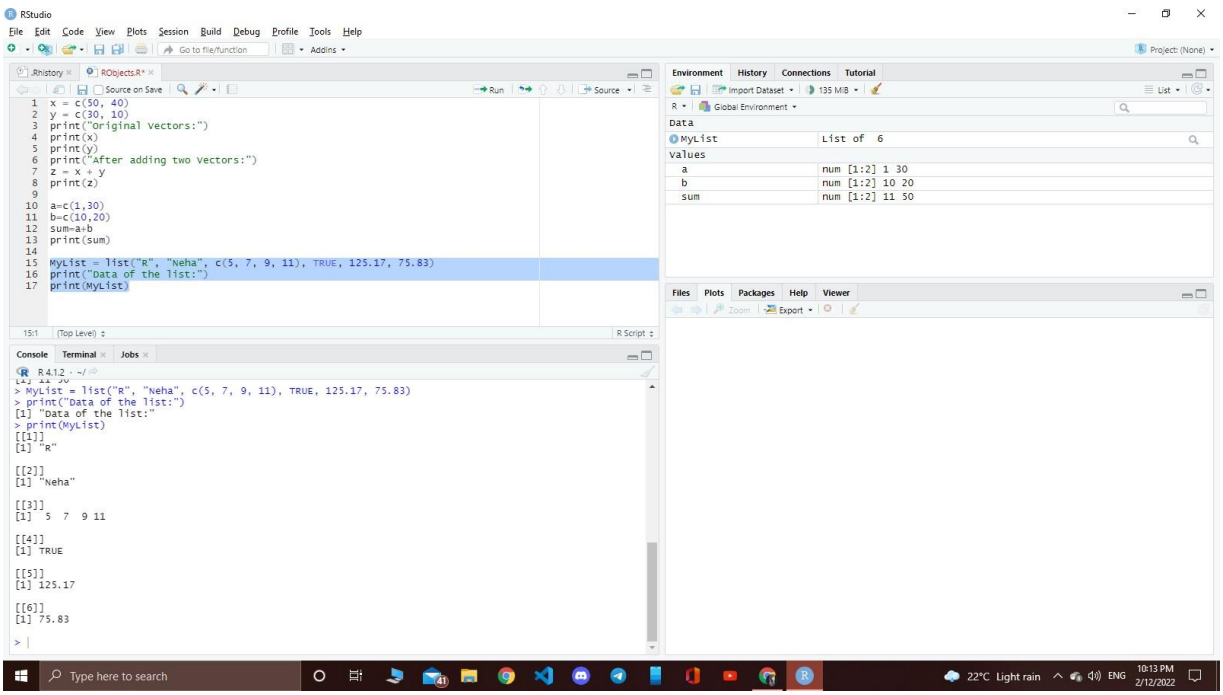
- Atomic Vectors: A vector is a data structure that stores a sequence of data elements of the same basic data types.
- Lists: Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it.
- Matrices: Matrices store values in a two-dimensional array, just like a matrix from linear algebra.
- Arrays: The array function creates an n-dimensional array. You could use array to sort values into a cube of three dimensions or a hypercube in 4, 5, or n dimensions.
- Data Frames: Data frames are the two-dimensional version of a list.



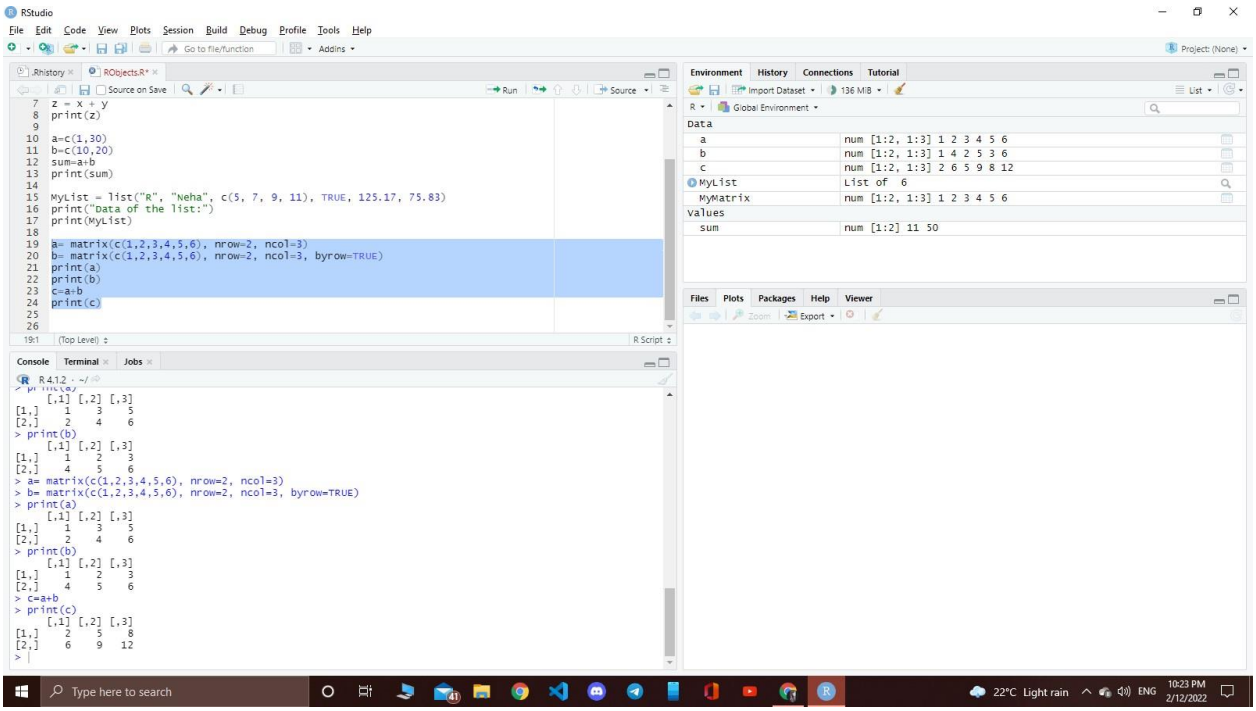
2.1) R Program to add two vectors of integers type.



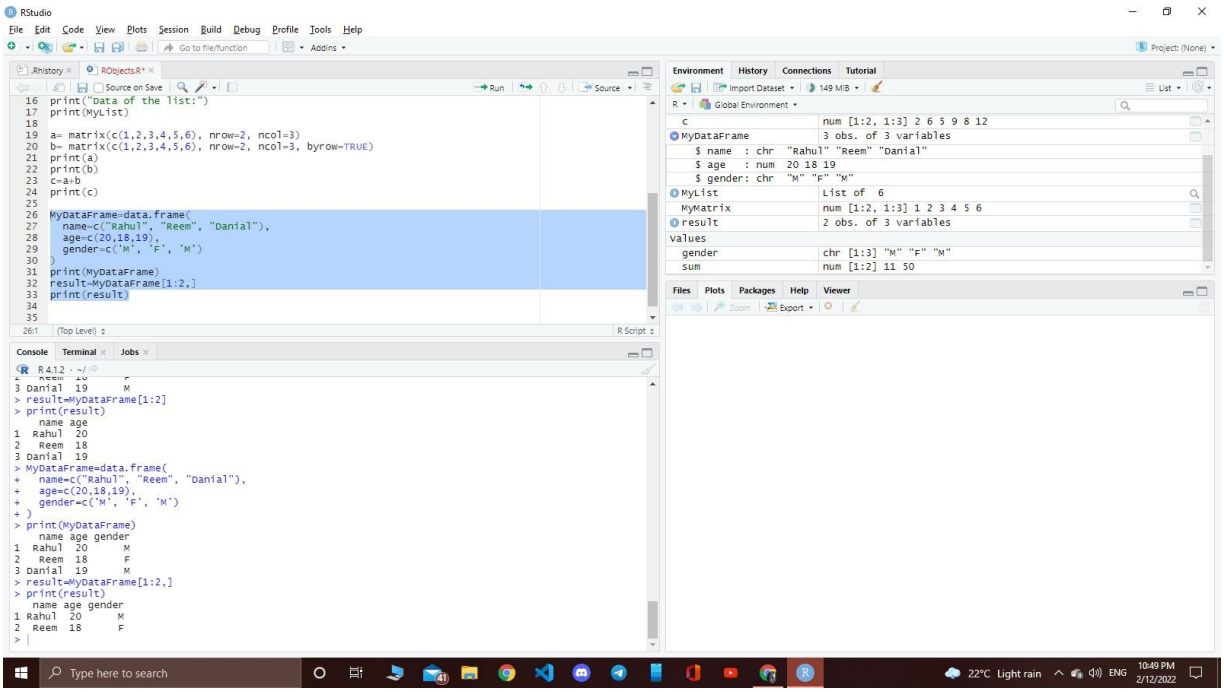
2.2) R program to create a list containing strings, numbers, vectors and a logical values.



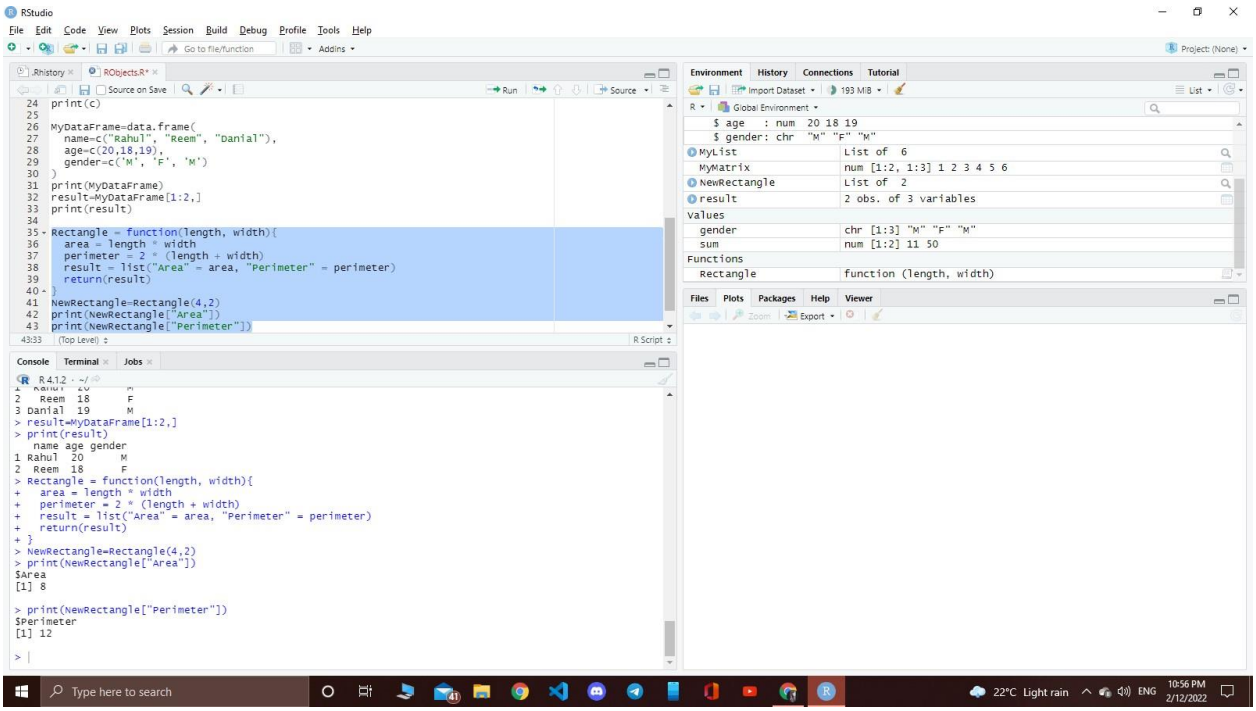
2.3.1) R program to add two matrixes.



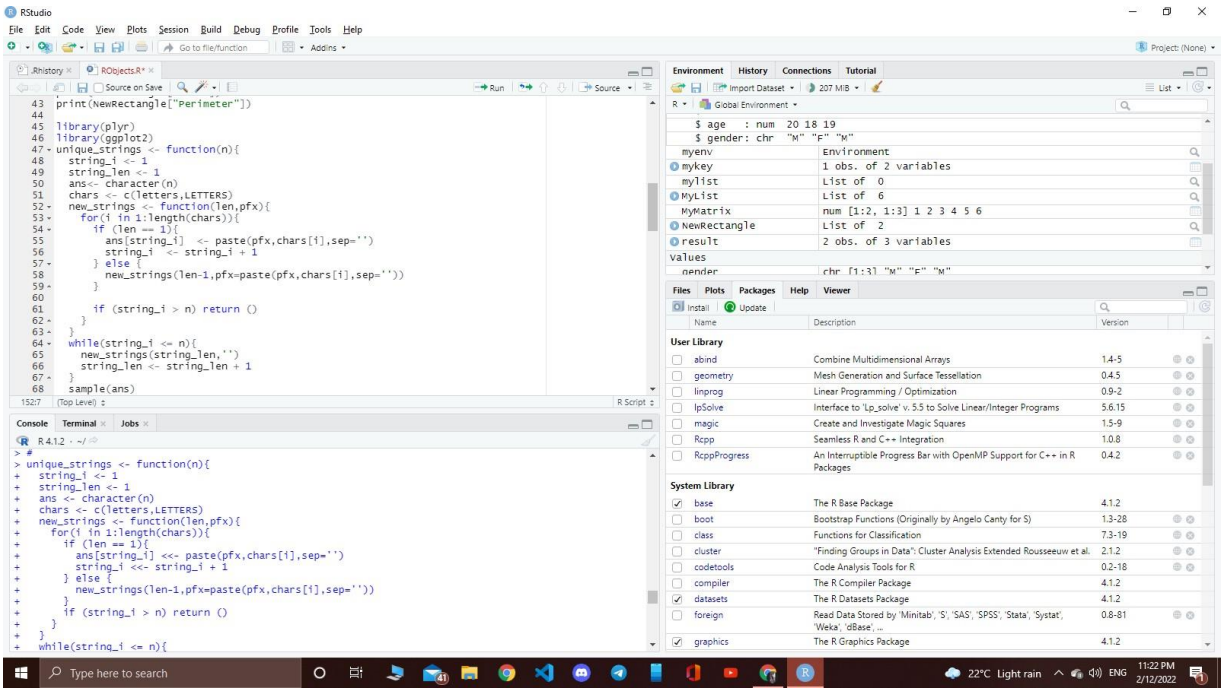
2.3.2) R program to extract first two rows from a given data frame.



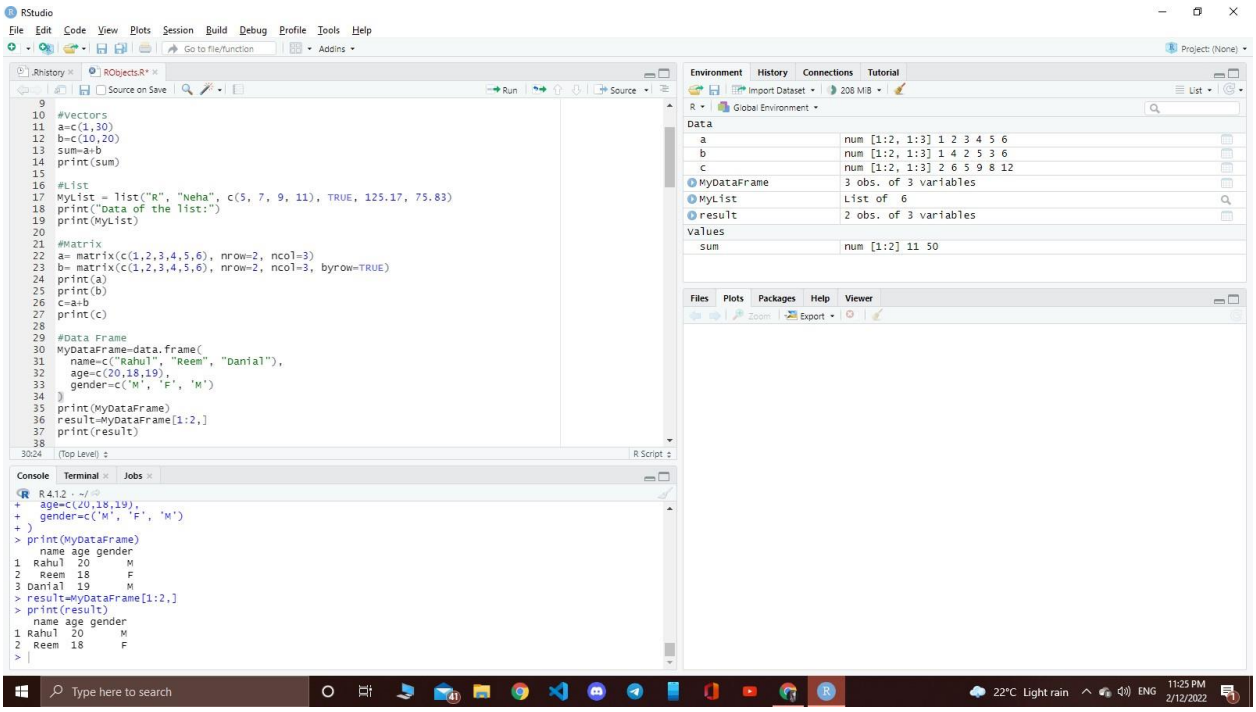
2.4) R program to calculate area and perimeter of a rectangle.

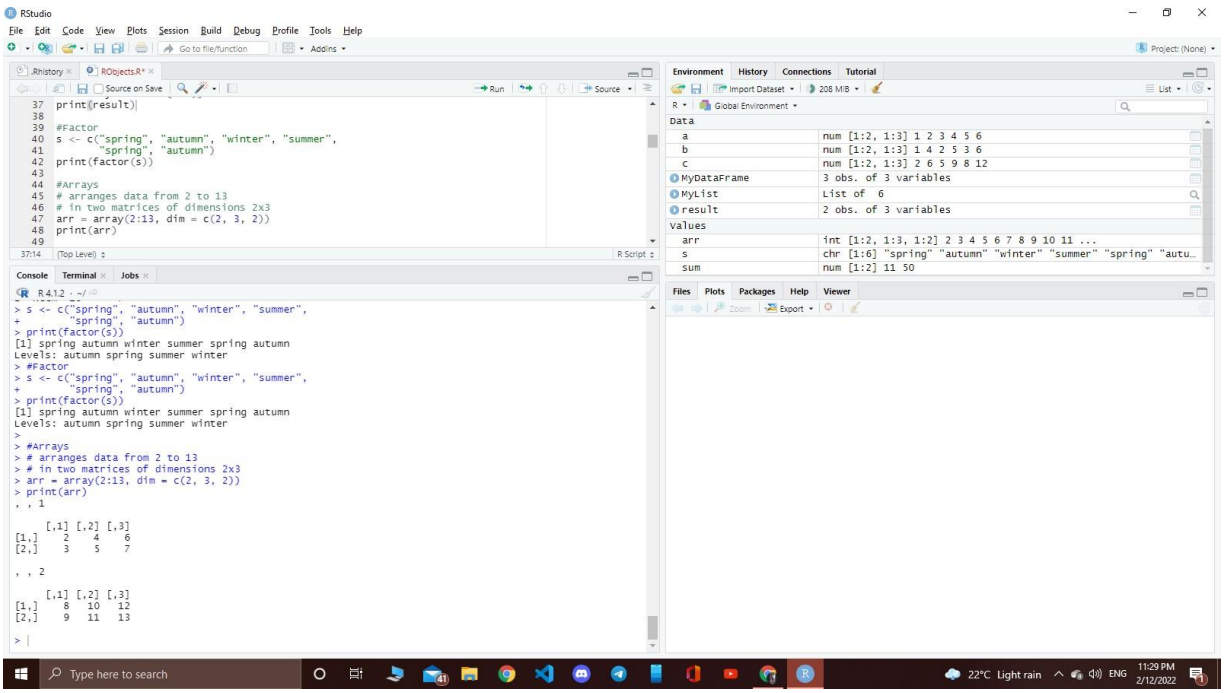


2.5) Create a hash table or Associative array using different R data structures.



2.6) R data objects:





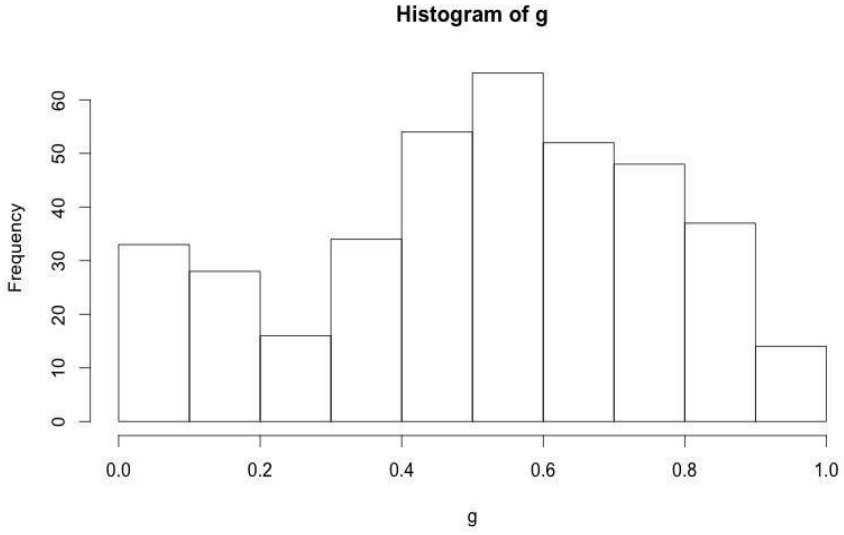
APL Assignment 3

Q1: Describe the types of tailed histograms such as long-tailed, short-tailed and skewed-tailed and provide a sample of R code snippets for each type.

Ans:

Histogram: A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range. The most common form of the histogram is obtained by splitting the range of the data into equal-sized bins (called classes).

R creates histogram using hist() function. This function takes a vector as an input and uses some more parameters to plot histograms.



The basic syntax for creating a histogram using R is − hist(v, main, xlab, xlim, ylim, breaks, col, border)

Following is the description of the parameters used −

- v is a vector containing numeric values used in histogram.
- main indicates title of the chart.
- col is used to set color of the bars.
- border is used to set border color of each bar.
- xlab is used to give description of x-axis.
- xlim is used to specify the range of values on the x-axis.
- ylim is used to specify the range of values on the y-axis.
- breaks is used to mention the width of each bar.

The histogram graphically shows the following :

- center (i.e. the location) of the data
- spread (i.e. the scale) of the data
- skewness of the data
- presence of outliers; and
- presence of multiple modes in the data.

Types of Histograms with Examples:

- Short tailed histograms
  - For a short-tailed distribution, the tails approach zero very fast.
  - Such distributions commonly have a truncated ("sawed-off") look.
  - The classical short-tailed distribution is the uniform (rectangular) distribution in which the probability is constant over a given range and then drops to zero everywhere else, we would speak of this as having no tails, or extremely short tails.

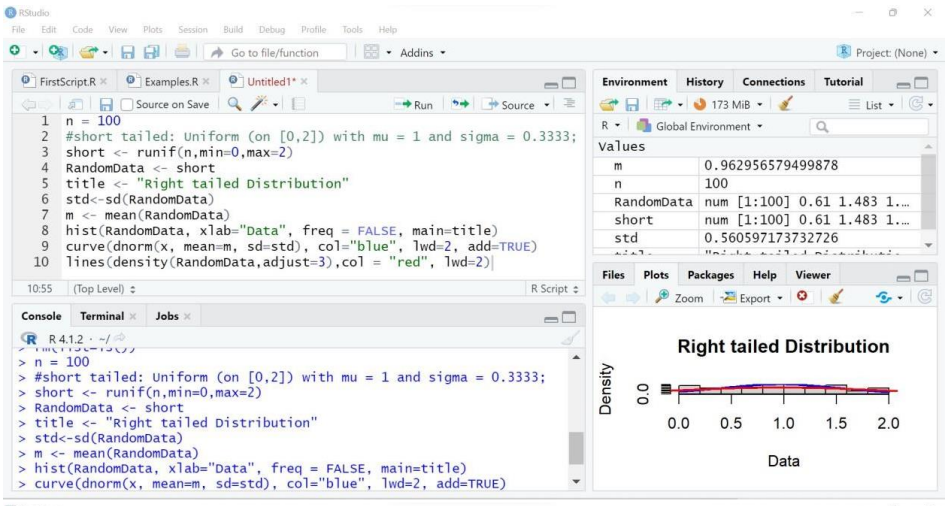
Example Code:

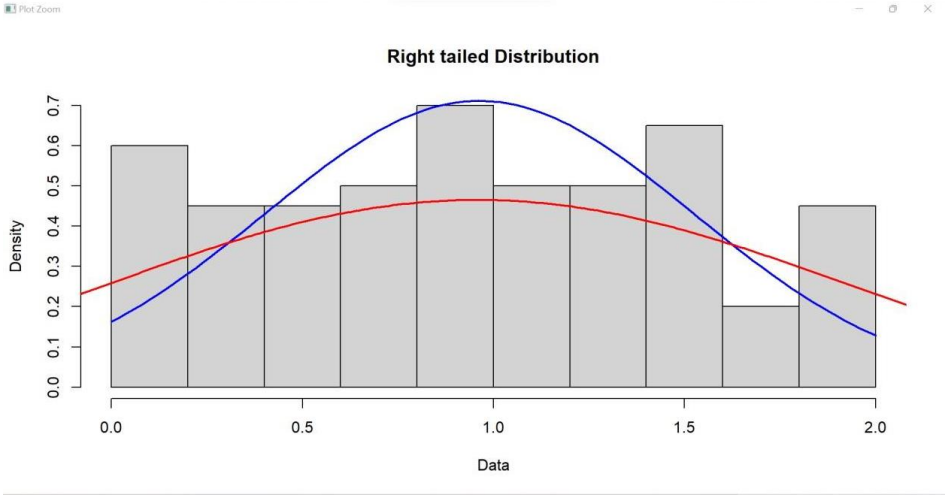
```
n = 100

#short tailed: Uniform (on [0,2]) with mu = 1 and sigma = 0.3333; short <-
runif(n,min=0,max=2) RandomData <- short

title <- "Right tailed Distribution"
std<-sd(RandomData) m <-
mean(RandomData)

hist(RandomData, xlab="Data", freq = FALSE, main=title) curve(dnorm(x,
mean=m, sd=std), col="blue", lwd=2, add=TRUE)
lines(density(RandomData,adjust=3),col = "red", lwd=2)
```





 Long tailed histograms

- For a long-tailed distribution, the tails decline to zero very slowly, and hence one is apt to see probability a long way from the body of the distribution. The classical long-tailed distribution is the Cauchy distribution.

Example Code 1:

N <- 10000 x <-

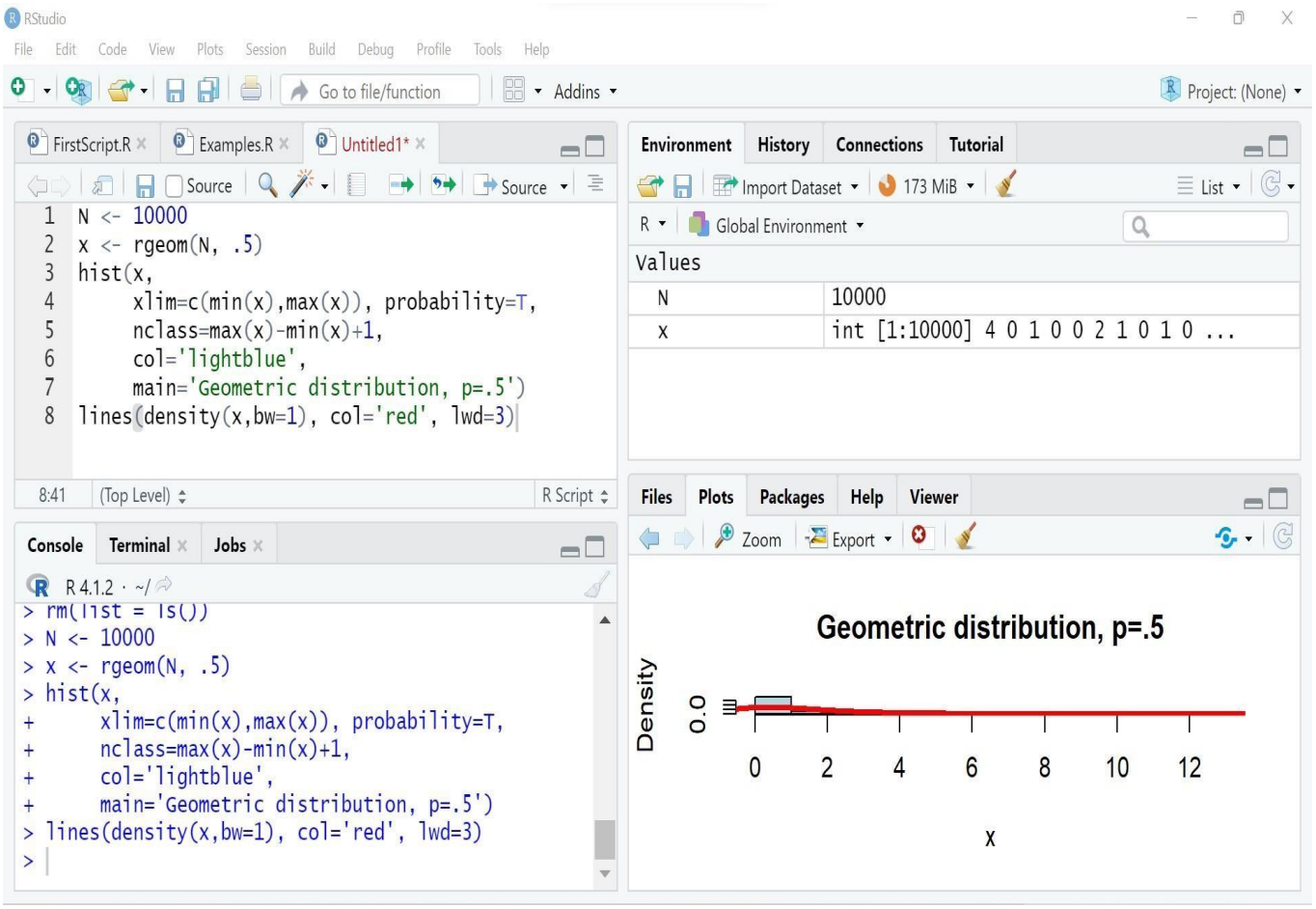
rgeom(N, .5) hist(x,

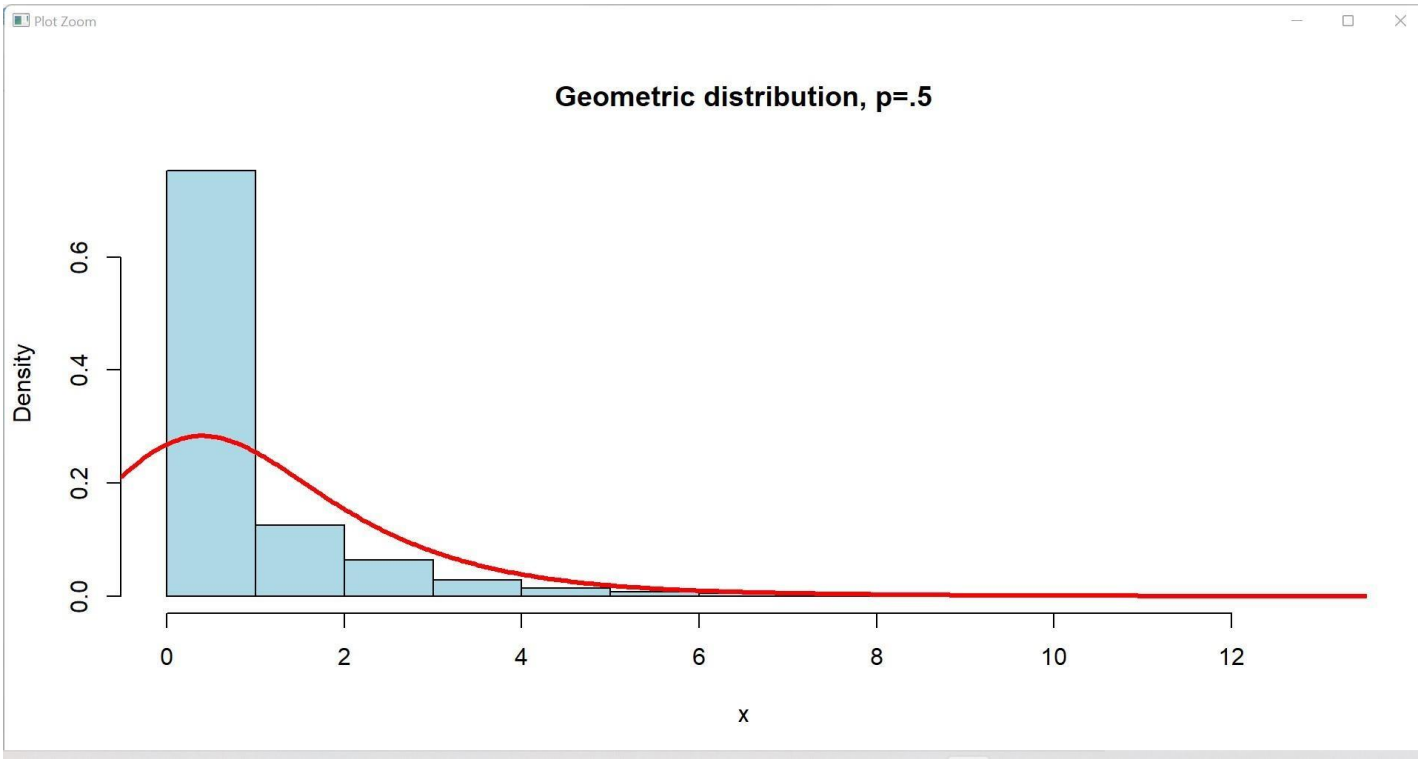
xlim=c(min(x),max(x)), probability=T, nclass=max(x)-min(x)+1,

col='lightblue',

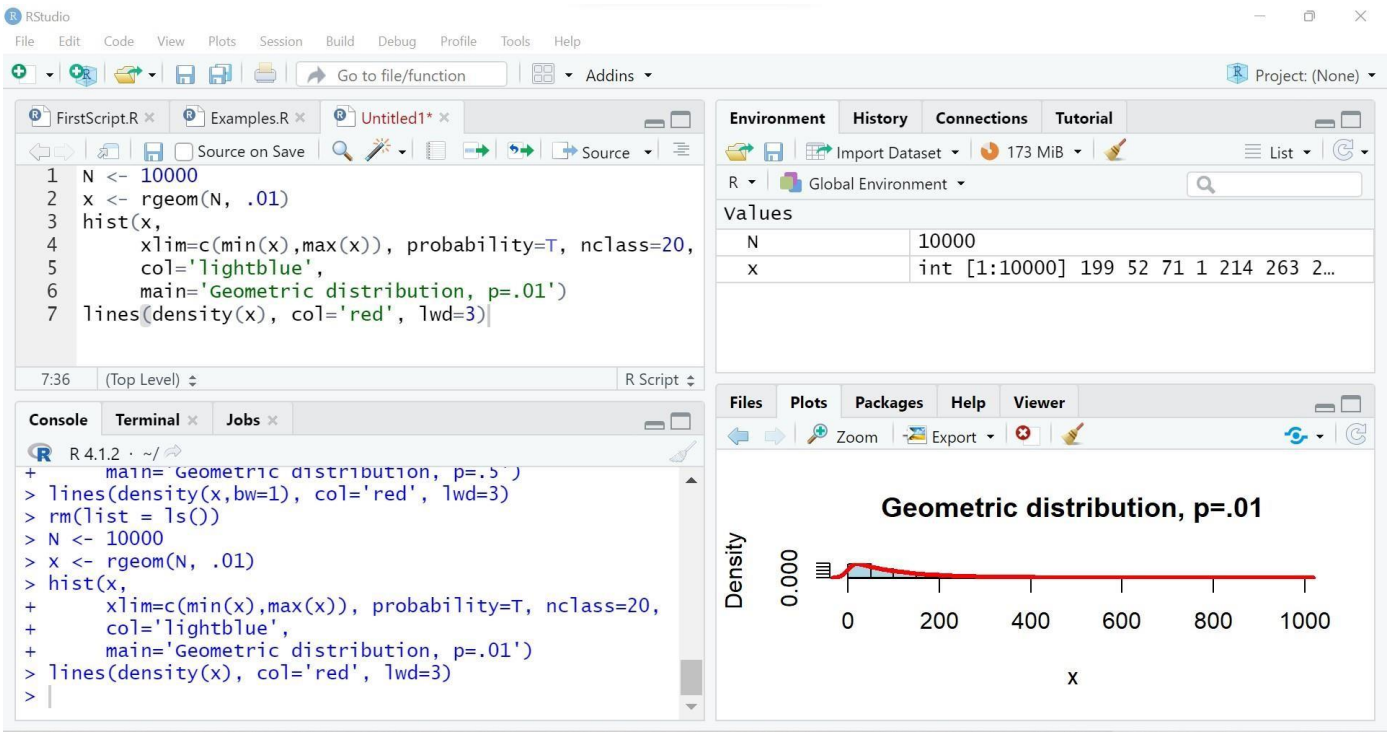
main='Geometric distribution, p=.5') lines(density(x,bw=1),

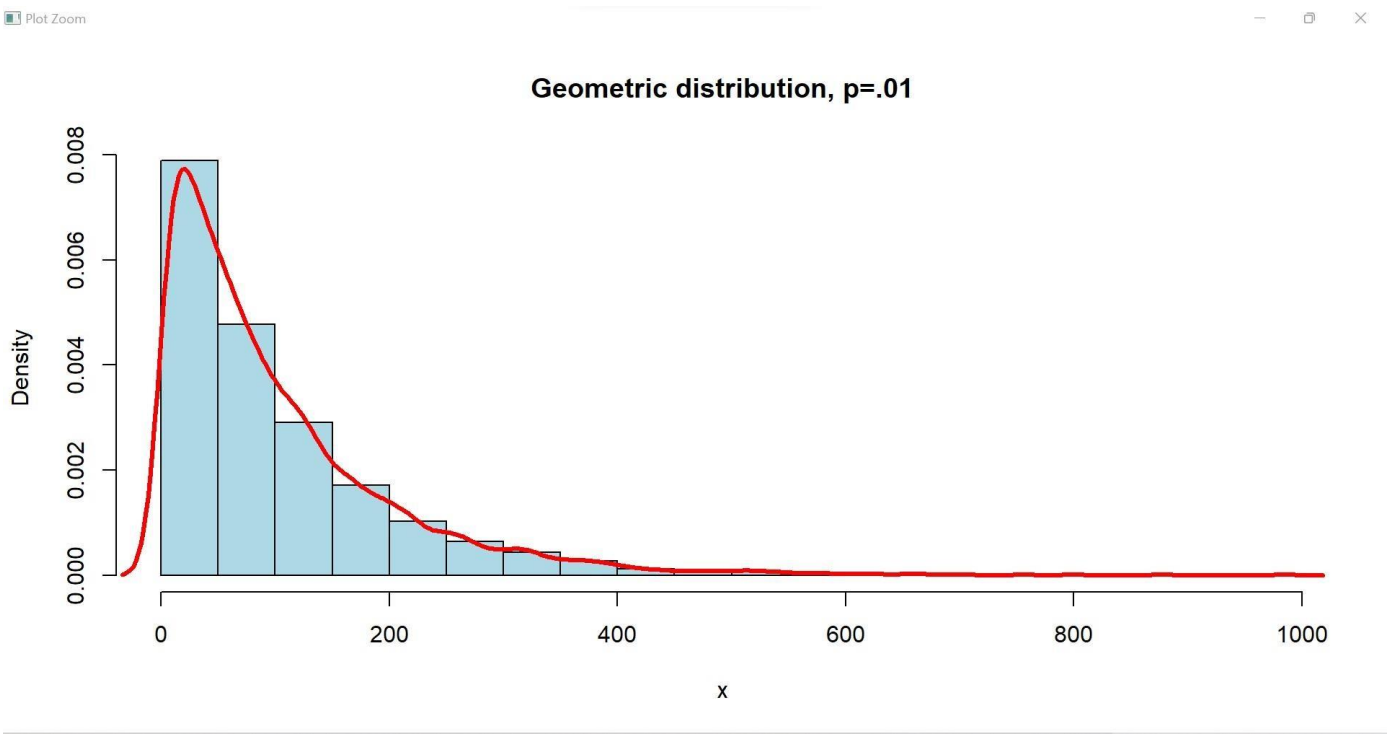
col='red', lwd=3)





Example Code 2: N <- 10000 x <- rgeom(N, .01) hist(x, xlim=c(min(x),max(x)), probability=T, nclass=20, col='lightblue', main='Geometric distribution, p=.01') lines(density(x), col='red', lwd=3)





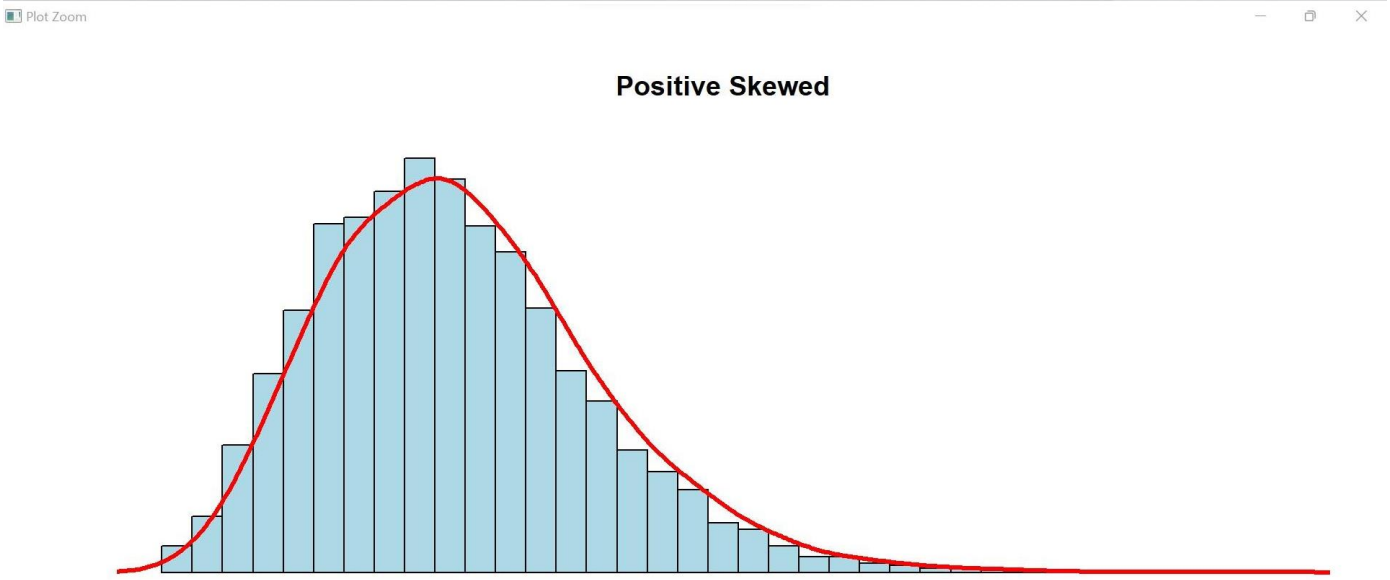
- Skewer tailed histograms
  - For skewed distributions, it is quite common to have one tail of the distribution considerably longer or drawn out relative to the other tail.
  - A "skewed right" distribution is one in which the tail is on the right side.
  - A "skewed left" distribution is one in which the tail is on the left side.

Example Code 1:

```
N <- 10000 x <-
rnbinom(N, 10, .5) hist(x)
    xlim=c(min(x),max(x)), probability=T,
nclass=max(x)-min(x)+1,      col='lightblue',
xlab=' ', ylab=' ',      axes=F,main='Positive
Skewed') lines(density(x,bw=1), col='red', lwd=3)
```

```
N <- 10000
x <- rnbinom(N, 10, .5)
hist(x,
     xlim=c(min(x),max(x)), probability=T,
     nclass=max(x)-min(x)+1,
     col='lightblue', xlab=' ', ylab=' ',
     axes=F,main='Positive Skewed')
lines(density(x,bw=1), col='red', lwd=3)
```

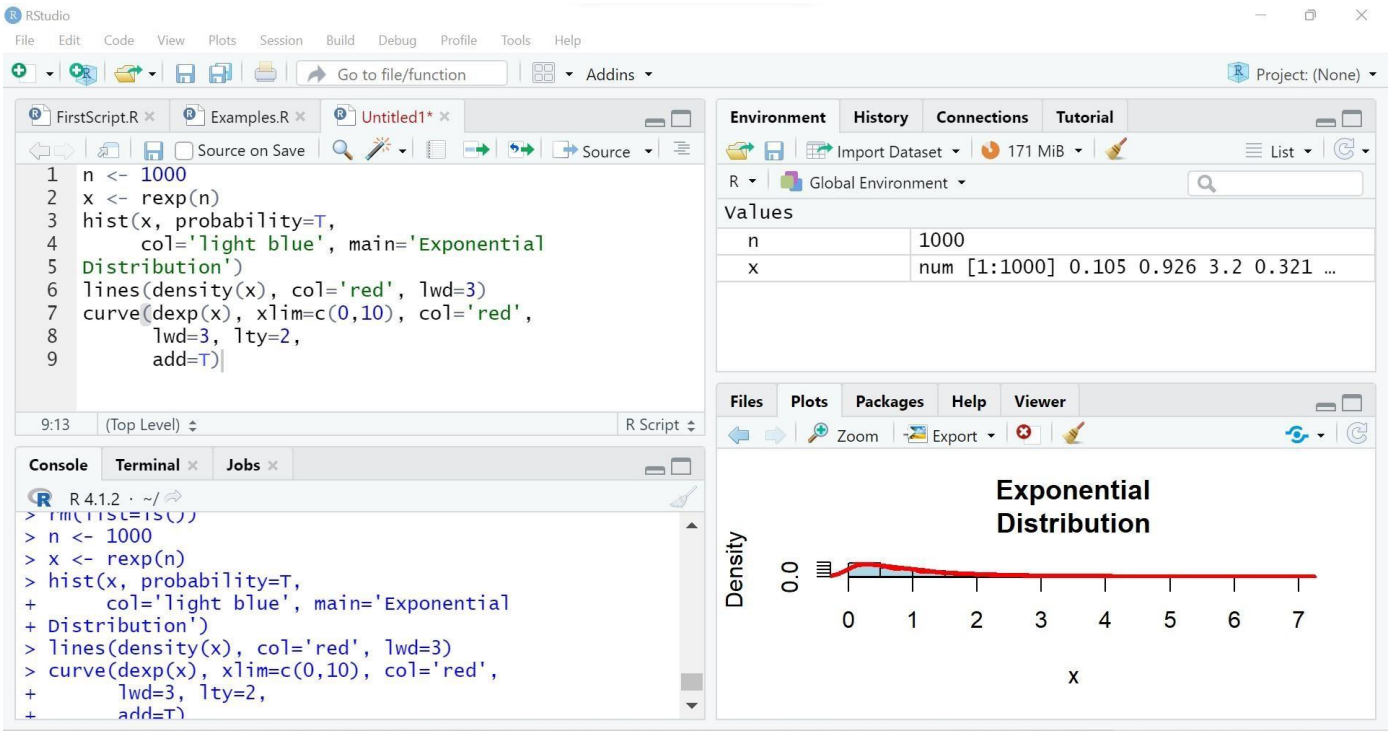
```
> rm(list = ls())
> N <- 10000
> x <- rnbinom(N, 10, .5)
> hist(x,
+      xlim=c(min(x),max(x)), probability=T,
+      nclass=max(x)-min(x)+1,
+      col='lightblue', xlab=' ', ylab=' ',
+      axes=F,main='Positive Skewed')
> lines(density(x,bw=1), col='red', lwd=3)
```

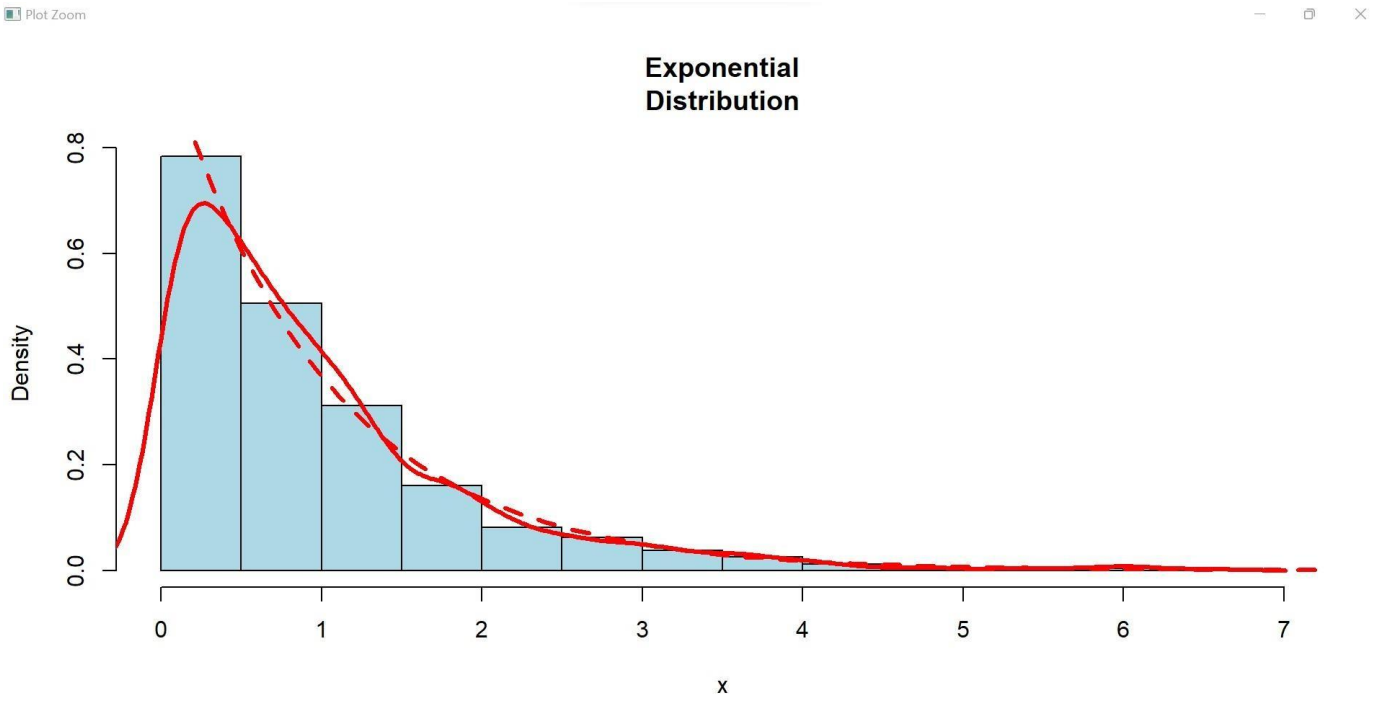


Example Code 2:

```
n <- 1000 x <- rexp(n)
hist(x, probability=T,
    col='light blue', main='Exponential Distribution')
```

```
lines(density(x), col='red', lwd=3)
curve(dexp(x), xlim=c(0,10), col='red',
lwd=3, lty=2,       add=T)
```





Q2: Write a R program to plot multiple time series using ggplot2 data visualization package.

Ans:

ggplot2: ggplot2 is an R package used for statistical computing and data representation using data visualization. It follows underlying graphics called Grammar of Graphics which includes

certain rules and independent components which can be used to represent data in various formats.

Program using ggplot2 (step wise)

- We first need to install and load the reshape2 package, if we want to use the functions thatare included in the add-on package:
- Now, we can reshape our data from wide to long format using the melt function:
- Now, we need to install and load the ggplot2 package to draw our time series plot usingggplot2:
- Finally, we can apply the ggplot and geom_line commands to draw multiple time series to aplot:

Example Code 1: library("reshape2")
library("ggplot2") set.seed(1023172)

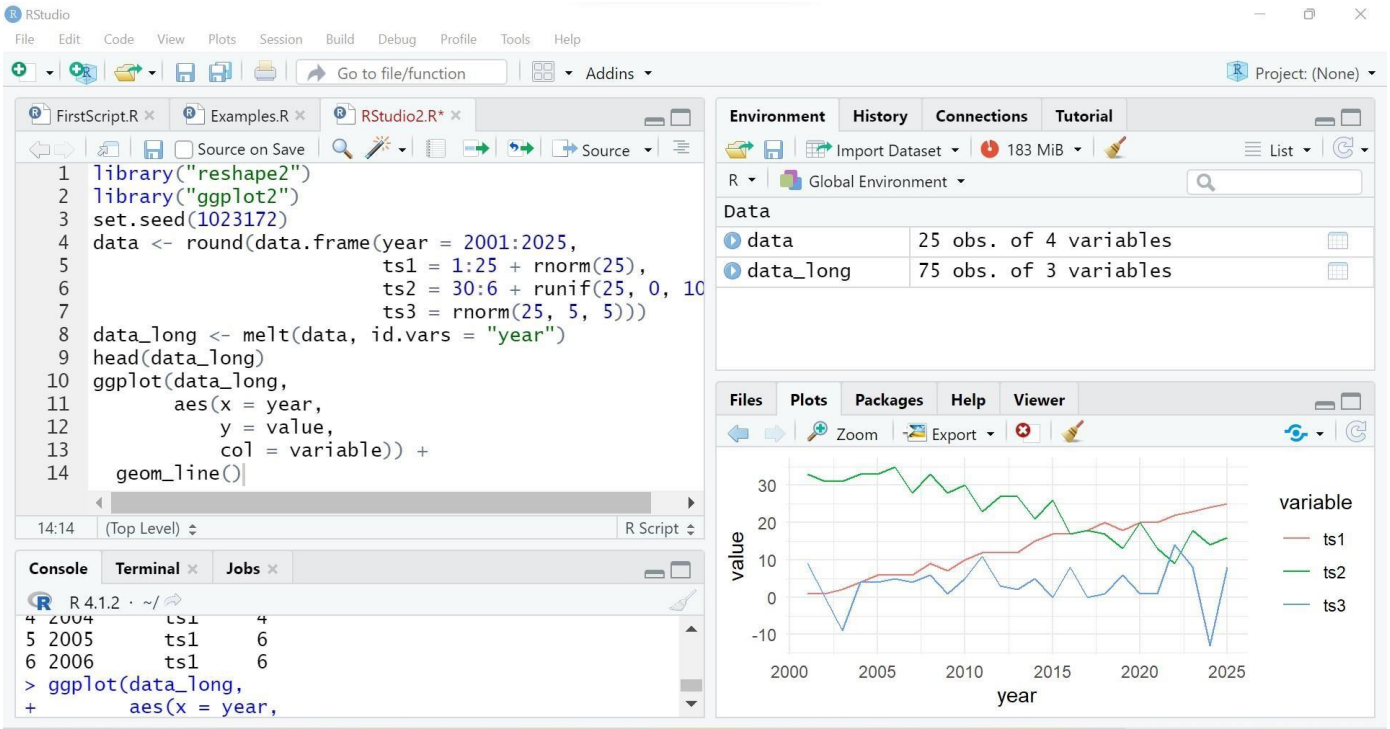
data <- round(data.frame(year = 2001:2025,

ts1 = 1:25 + rnorm(25),

ts2 = 30:6 + runif(25, 0, 10), ts3 = rnorm(25, 5, 5))) data_long <- melt(data, id.vars = "year") head(data_long)

ggplot(data_long,

aes(x = year, y = value, col = variable)) + geom_line()



Example code 2: library(ggplot2)
library(reshape2) library(dplyr)

covid1 =(read.csv(file="EUCOVIDdeaths.csv",header=TRUE)[,-c(2)])

head(covid1)

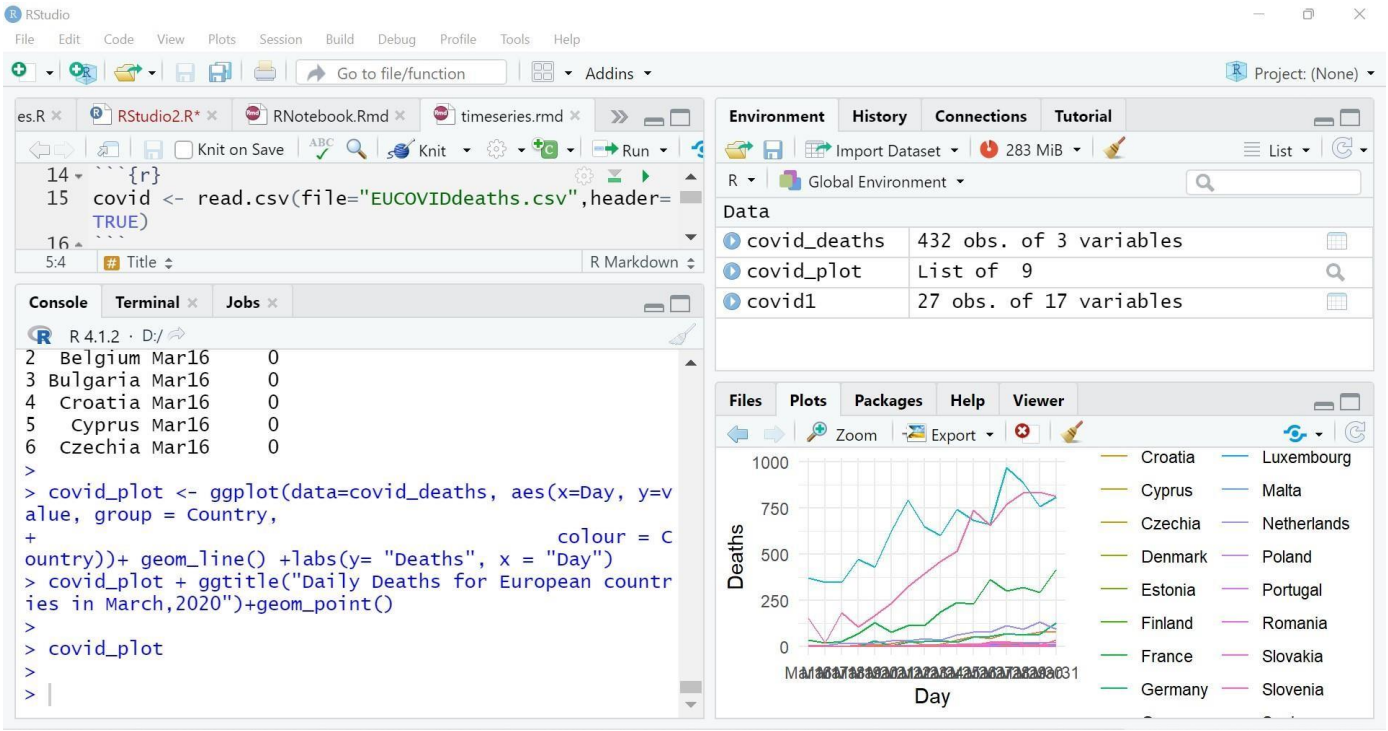
covid_deaths <- melt(covid1,id.vars=c("Country"),value.name="value",
variable.name="Day")

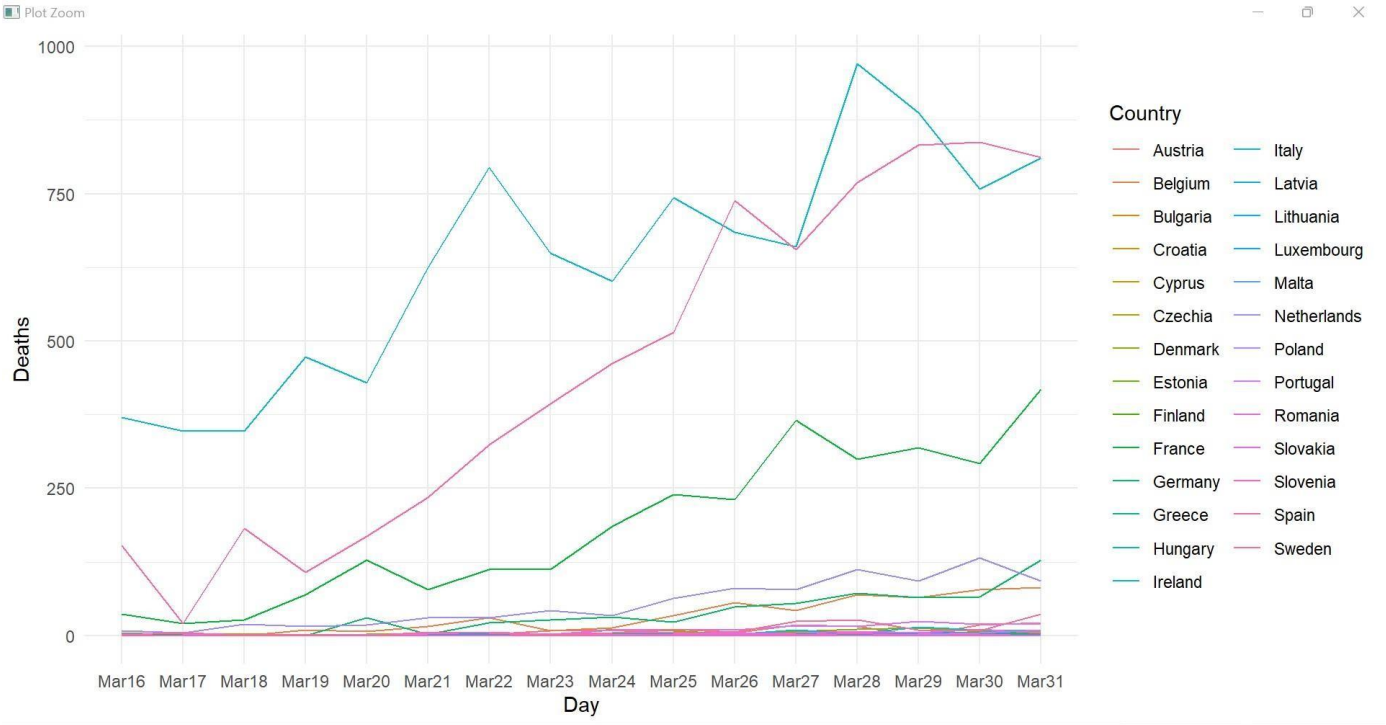
head(covid_deaths)

covid_plot <- ggplot(data=covid_deaths, aes(x=Day, y=value, group = Country, colour =

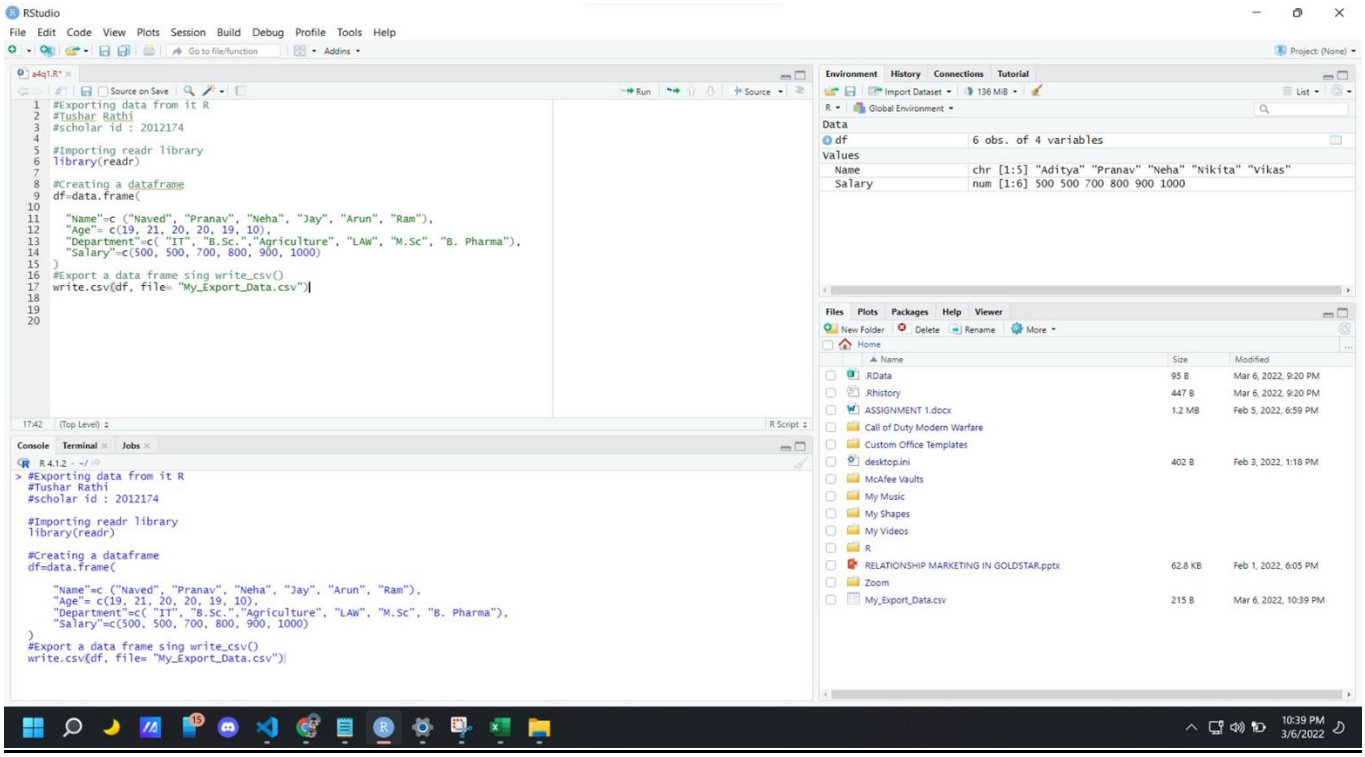
Country)) + geom_line() +labs(y= "Deaths", x = "Day") covid_plot +
ggtitle("Daily Deaths for European countries in
March,2020")+geom_point()

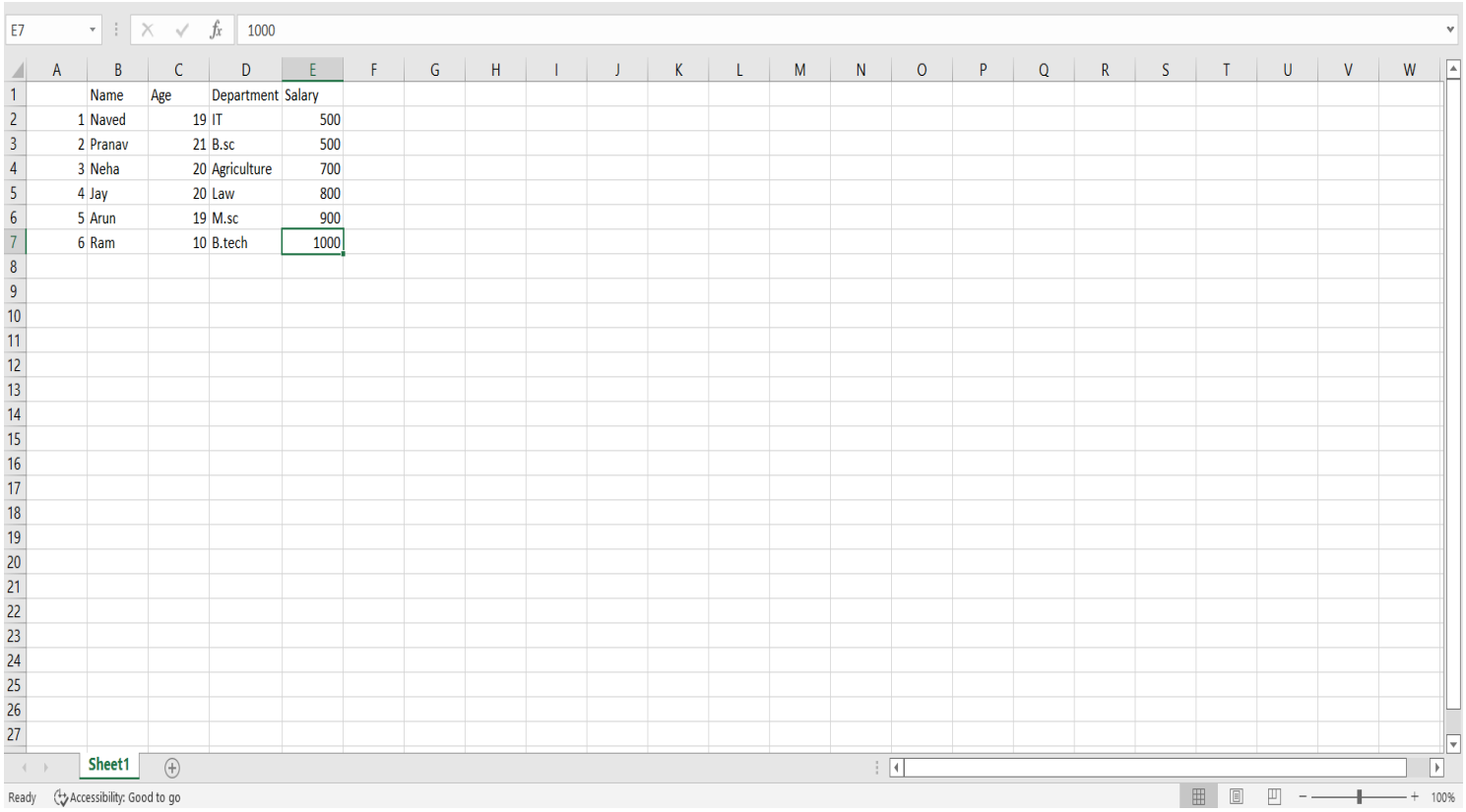
covid_plot

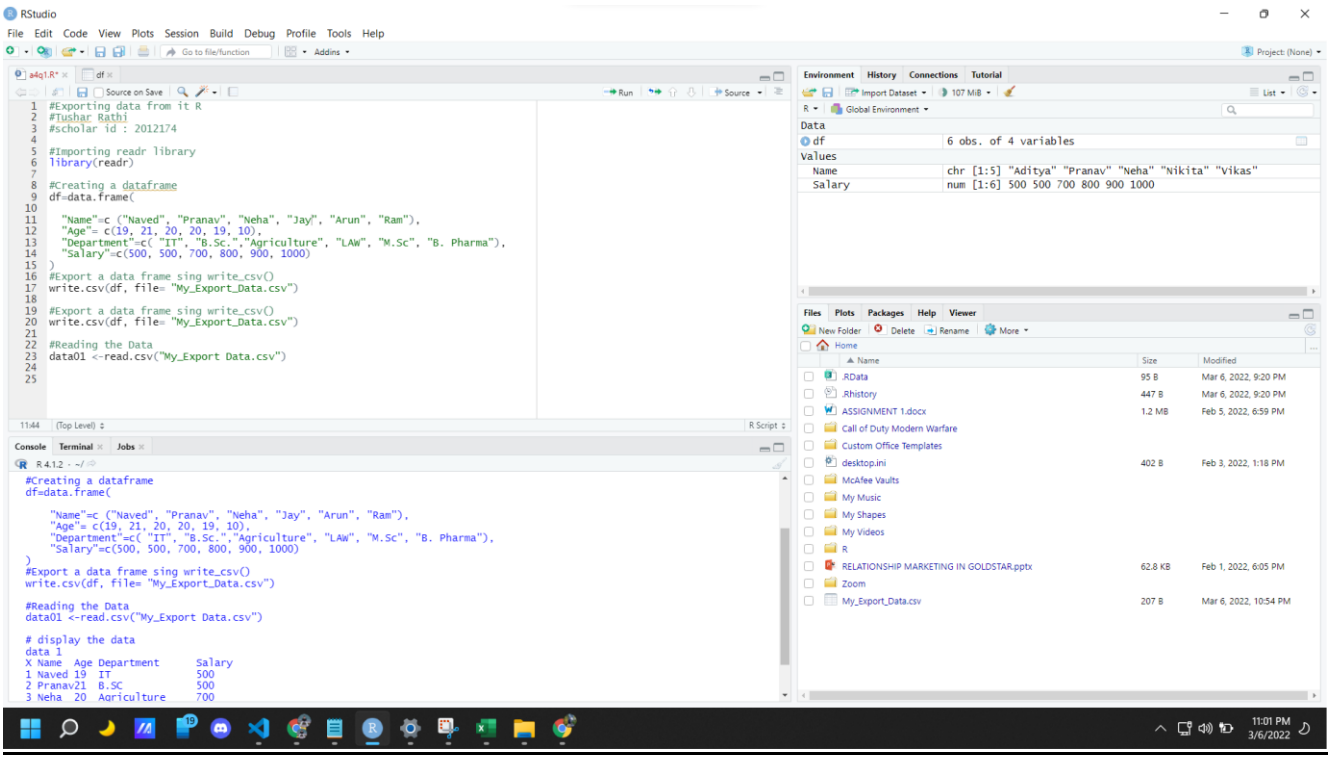

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
es.R
RStudio2.R*
RNotebook.Rmd
timeseries.rmd
Knit on Save
Knit
Run
```{r}
covid <- read.csv(file="EUCOVIDdeaths.csv",header=
TRUE)
```
5:4
Title
R Markdown
Console
Terminal
Jobs
R 4.1.2 · D:/
2 Belgium Mar16 0
3 Bulgaria Mar16 0
4 Croatia Mar16 0
5 Cyprus Mar16 0
6 Czechia Mar16 0
> covid_plot <- ggplot(data=covid_deaths, aes(x=Day, y=value, group = Country,
+ colour = Country))+ geom_line() +labs(y= "Deaths", x = "Day")
> covid_plot + ggtitle("Daily Deaths for European countries in March,2020")+geom_point()
> covid_plot
Environment
History
Connections
Tutorial
Import Dataset
283 MiB
List
R
Global Environment
Data
covid_deaths 432 obs. of 3 variables
covid_plot List of 9
covid1 27 obs. of 17 variables
Files
Plots
Packages
Help
Viewer
Zoom
Export
Deaths
Day
Croatia
Cyprus
Czechia
Denmark
Estonia
Finland
France
Germany
Luxembourg
Malta
Netherlands
Poland
Portugal
Romania
Slovakia
Slovenia


Plot Zoom
Deaths
Day
Country
Austria
Belgium
Bulgaria
Croatia
Cyprus
Czechia
Denmark
Estonia
Finland
France
Germany
Greece
Hungary
Ireland
Italy
Latvia
Lithuania
Luxembourg
Malta
Netherlands
Poland
Portugal
Romania
Slovakia
Slovenia
Spain
Sweden

LAB ASSIGNMENT A4:







Q2. A) .gz files:

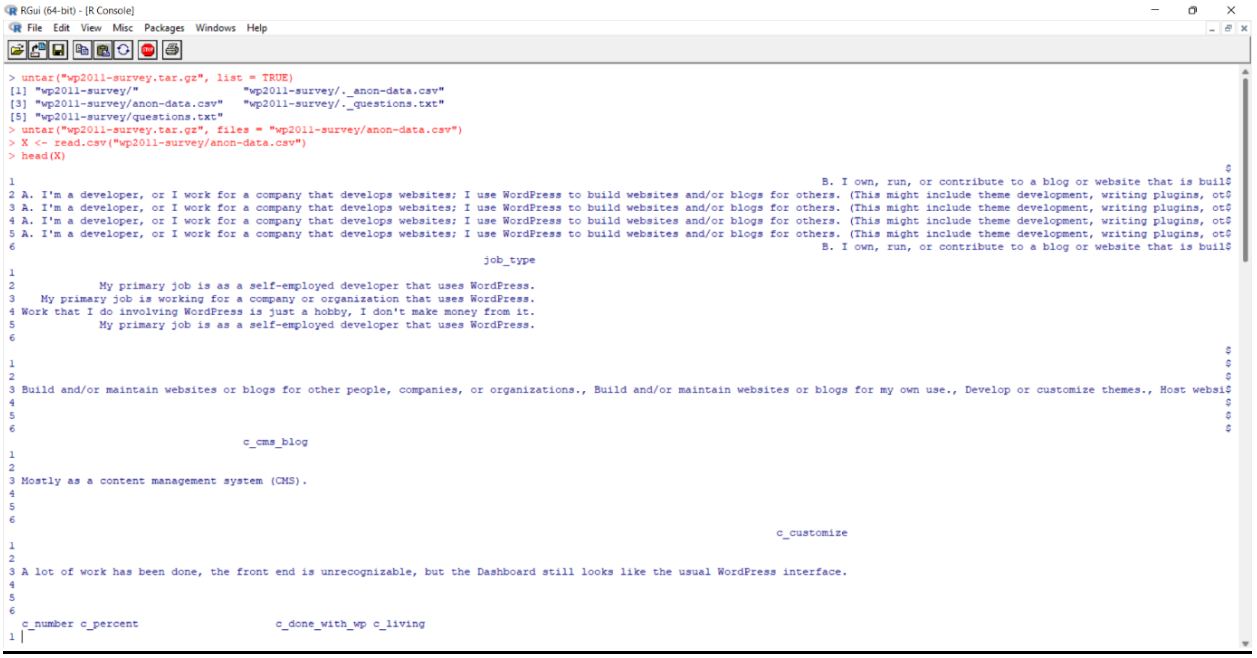
gzfile(description, open = "", encoding = getOption("encoding"), compression = 6)

Internally gzfile() (see connections) is used to read (write) chunks to (from) the gzip file. If the process is interrupted before completed, the partially written output file is automatically removed.

For gzfile the description is the path to a file compressed by gzip: it can also open for reading uncompressed files and those compressed by bzip2, xz or lzma.

R Code to read .gz files:

```
>untar("wp2011-survey.tar.gz", list = TRUE)
>untar("wp2011-survey.tar.gz", files = "wp2011-survey/anon-data.csv")
>data <- read.csv("wp2011-survey/anon-data.csv")
>head(data)
```

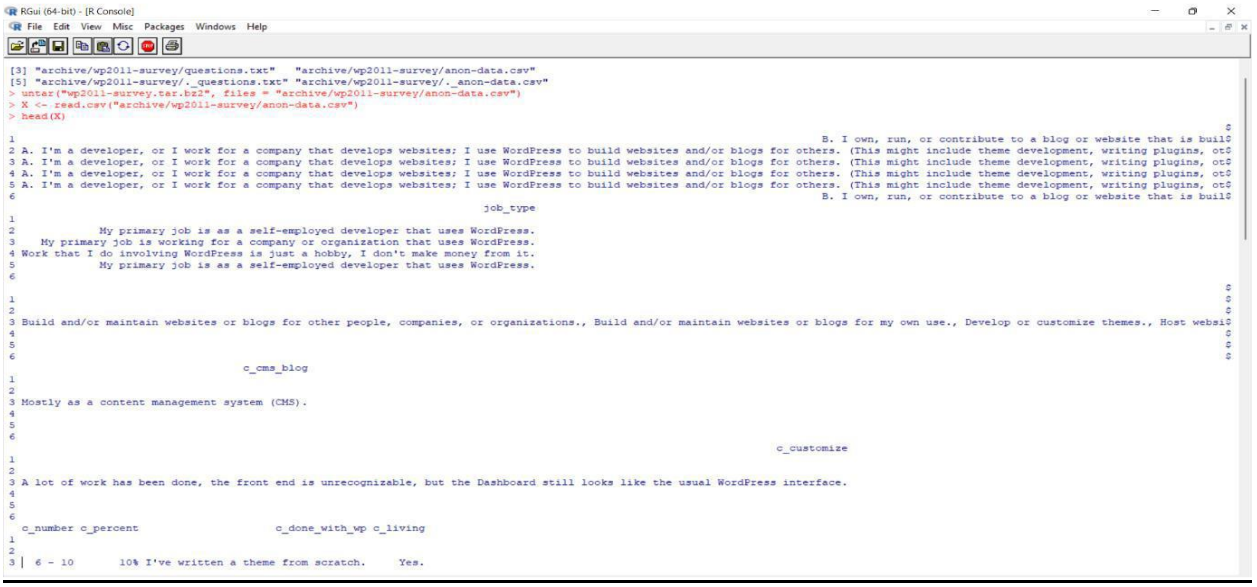


B) .bz2 files:

Similarly to read .bz2 files **untar()** function is also used. The choice of compression whether ".gz" or ".bz2" is passed as a parameter in both the functions. In **untar()** the parameter is called compressed and is deprecated in favour of auto detection of the compression technique.

R Code to read .bz2 files:

```
>untar("wp2011-survey.tar.bz2", list = TRUE)
>untar("wp2011-survey.tar.bz2", files = "archive/wp2011-survey/anon-data.csv")
>X <- read.csv("archive/wp2011-survey/anon-data.csv")
>head(X)
```

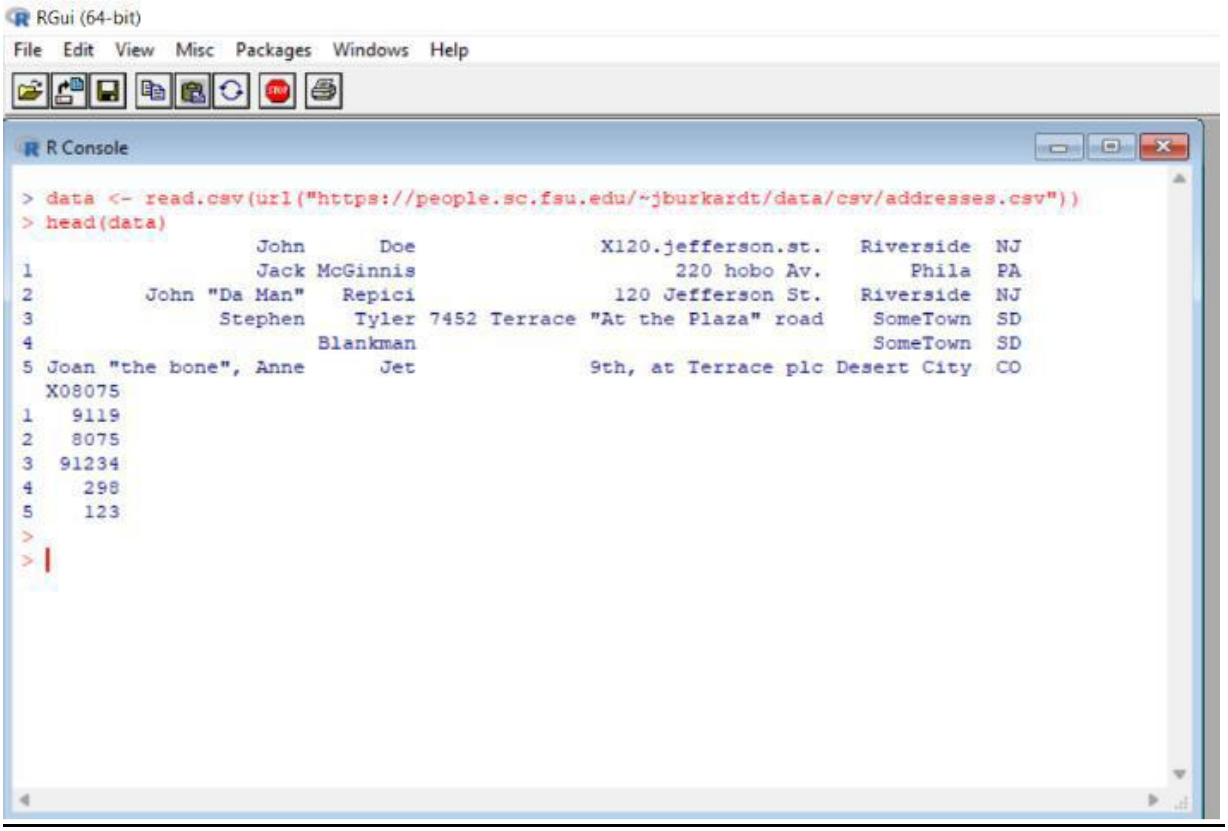


C) URLs:

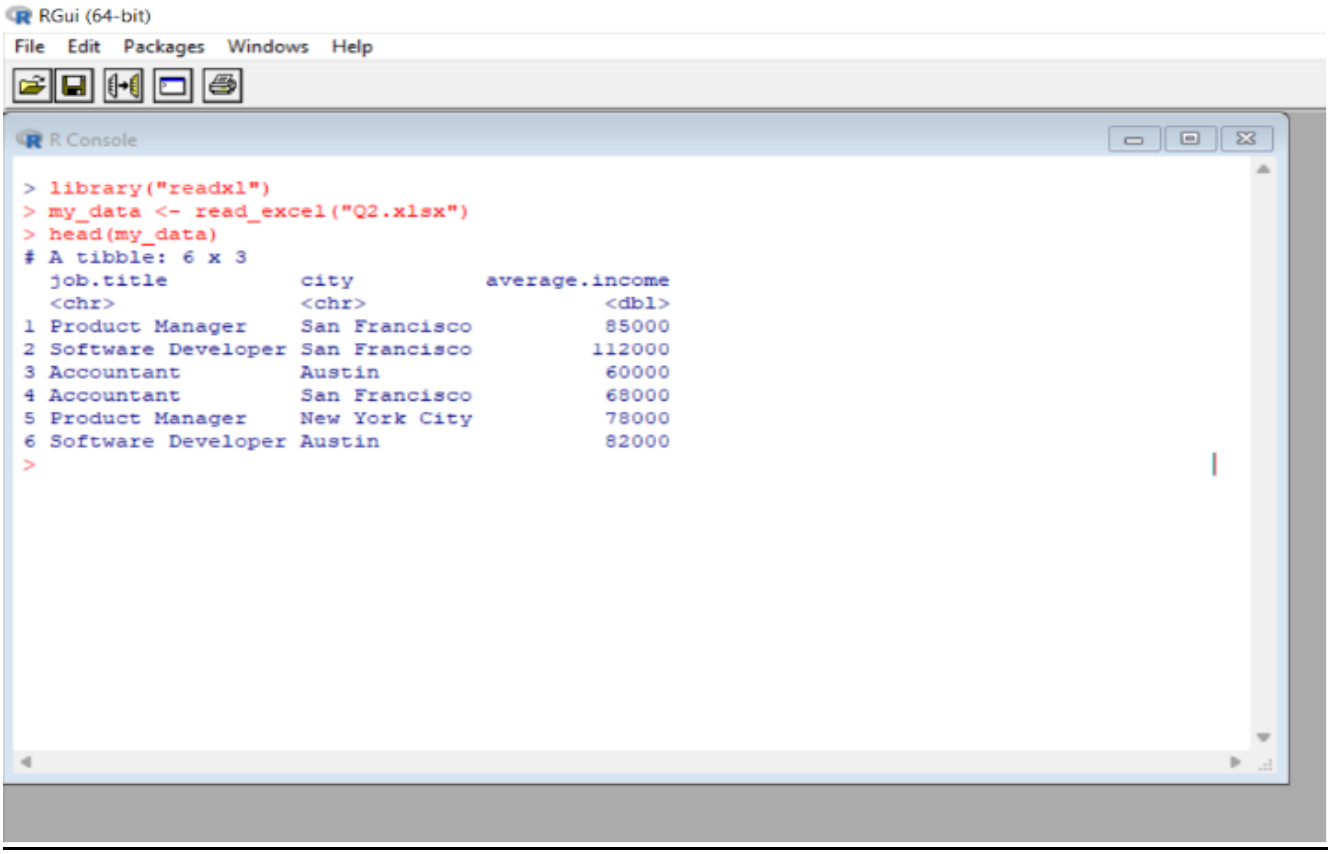
For URL the description is a complete URL including scheme (such as 'http://', 'https://', 'ftp://' or 'file:///'). Method "internal" is that available since connections were introduced but now mainly defunct.

Function used:

url(description, open = "", blocking = TRUE, encoding = getOption("encoding"), method = getOption("url.method", "default"), headers = NULL)
R Code to read URLs:
>data <-
read.csv(url("https://people.sc.fsu.edu/~jburkardt/data/csv/addresses.scv)")
>head(data)



D) MS Excel files:
To import MS Excel files one can use:
>my_data <- read.table(file = "clipboard",
+sep = "\t", header = TRUE)
OR
>library("readxl")
># For xls files
>my_data <- read_excel("my_file.xls")
># For xlsx files
>my_data <- read_excel("my_file.xlsx")
For the 2nd procedure we need to install the readxl package using install.packages("readxl").
R Code for reading MS Excel files:
library("readxl")
my_data <- read_excel("Q2.xlsx")



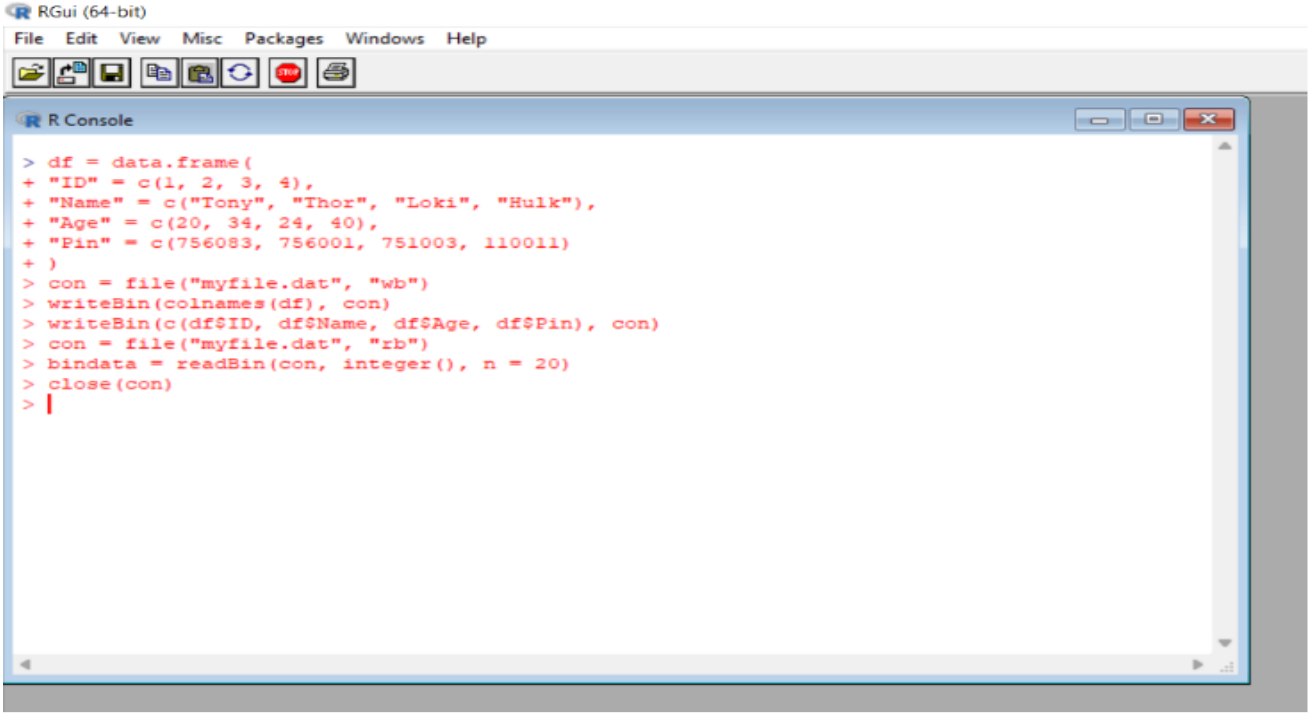
E) Binary files:

The both creating and writing to a binary file can be performed by a single function **writeBin()** by opening the file in "**wb**" mode where w indicates write and b indicates binary mode. The Syntax is writeBin(object, con). Con is a connection objext or a character string naming a file or a raw vector etc.

R Code for Writing Binary files:

```
>df = data.frame(
+"ID" = c(1, 2, 3, 4),
+"Name" = c("Tony", "Thor", "Loki", "Hulk"),
+"Age" = c(20, 34, 24, 40),
+"Pin" = c(756083, 756001, 751003, 110011)
>)
>
># Creating a connection object to write the binary file using mode "wb"
>con = file("myfile.dat", "wb")
>
># Write the column names of the data frame to the connection object
>writeBin(colnames(df), con)
>
># Write the records in each of the columns to the file
>writeBin(c(df$ID, df$Name, df$Age, df$Pin), con)
```

>
># Close the connection object
>close(con)

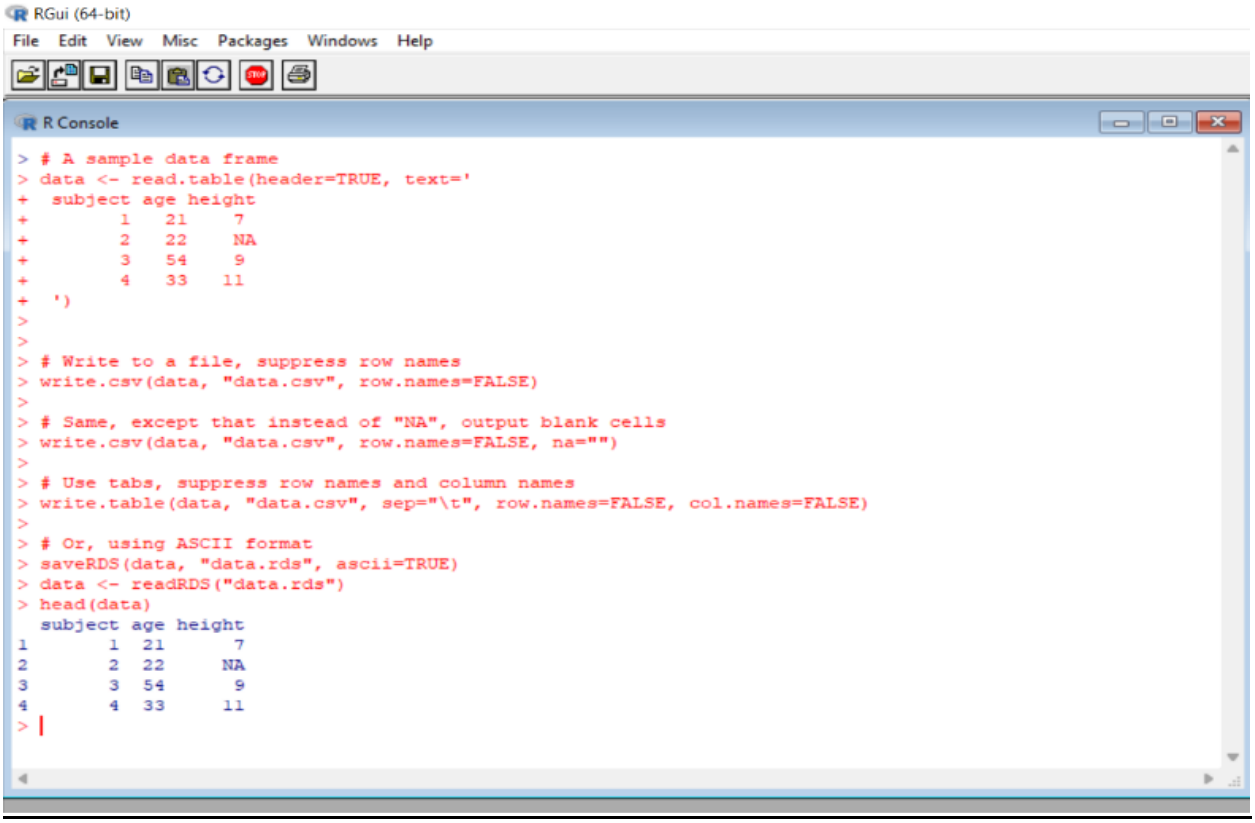


F) ASCII files:

The function used here to write an ACII file is write.cvs() and write.table(). To save the object in ASCII format we use saveRDS(data, “data.rds”, ascii = TRUE).

R Code to write ASCII files:

```
>data <- read.table(header=TRUE, text='
+ subject sex size
+ 1 M 7
+ 2 F NA
+ 3 F 9
+ 4 M 11
+')
># Write to a file, suppress row names
>write.csv(data, "data.csv", row.names=FALSE)
># Same, except that instead of "NA", output blank cells
>write.csv(data, "data.csv", row.names=FALSE, na="")
># Use tabs, suppress row names and column names
>write.table(data, "data.csv", sep="\t", row.names=FALSE, col.names=FALSE)
>saveRDS(data,”data.rds”, ascii = TRUE)
># To load the data again:
>data <- readRDS(“data.rds”)
>head(data)
```



LAB ASSIGNMENT A5:

1. What is KS Test?

The Kolmogorov-Smirnov Test is a type of nonparametric test of the equality of discontinuous and continuous of a 1D probability distribution that is used to compare the sample with the reference probability test (known as one-sample K-S Test) or among two samples (known as two-sample K-S test).

It quantifies a distance between the cumulative distribution function of the given reference distribution and the empirical distributions of given two samples, or between the empirical distribution of given two samples.

In a one-sample K-S test, the distribution that is considered under a null hypothesis can be purely discrete or continuous or mixed. In the two-sample K-S test, the distribution considered under the null hypothesis is generally continuous distribution but it is unrestricted otherwise.
Kolmogorov-Smirnov Test Formula
The formula for the Kolmogorov-Smirnov test can be given as:

Dn = supx |Fn(x) - F(x)| where, supx : the supremum of the set of distances
Fn(x) : the empirical distribution function for n id observations Xi .

The empirical distribution function is a distribution function that is associated with the empirical measures of the chosen sample. Being a step function, this cumulative distribution jumps up by a 1/n step at each and every n data points.

2. What is the Necessity to perform KS Test?

The Kolmogorov–Smirnov test is a nonparametric goodness-of-fit test and is used to determine whether two distributions differ, or whether an underlying probability distribution differes from a hypothesized distribution. It is used when we have two samples coming from two populations that can be different.

3. Steps to perform KS Test in R

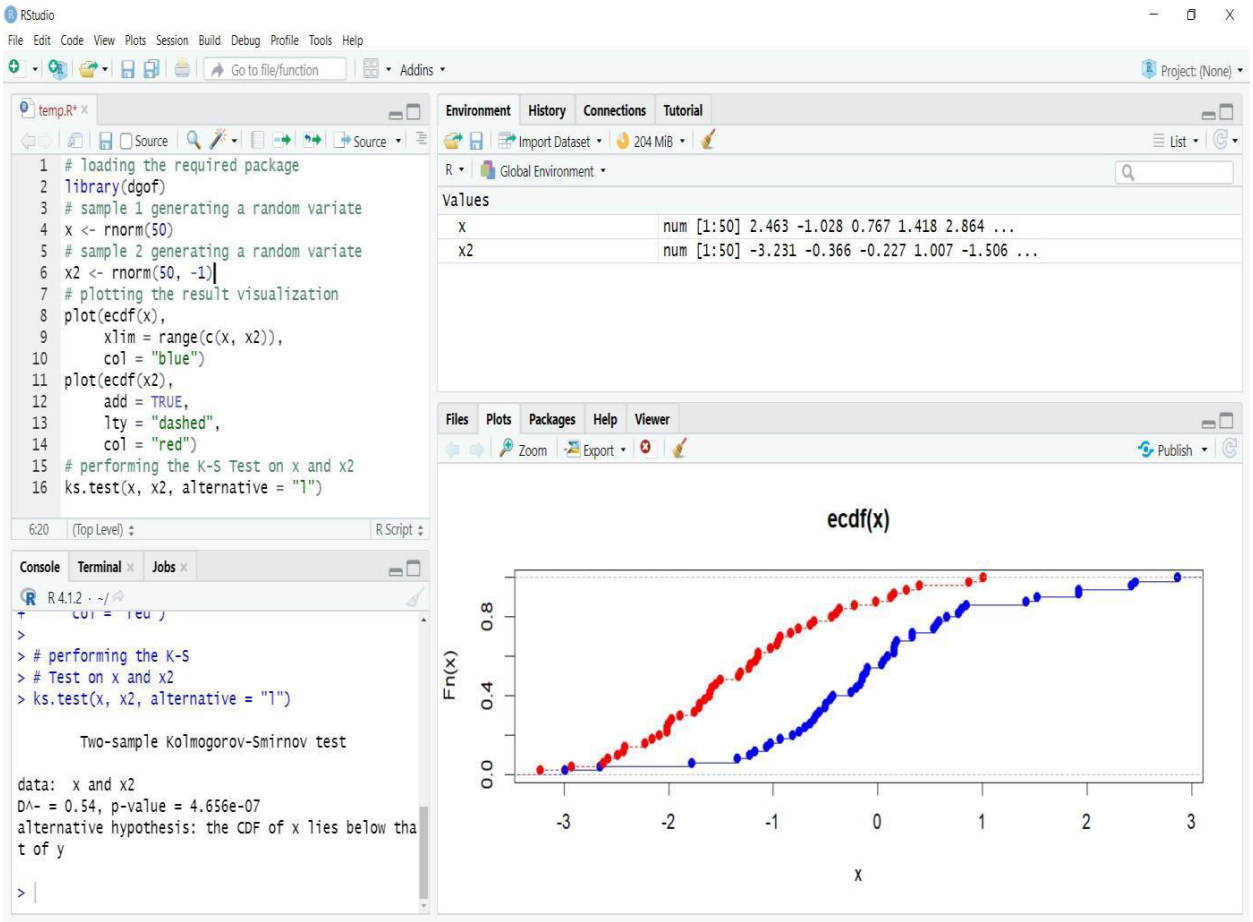
The K-S test can be performed using the ks.test()
function in R.
Syntax: ks.text(x, y, …, alternative = c(“two.sided”, “less”, “greater”), exact= NULL, tol= 1e-8, simulate.p.value = FALSE, B=2000)
Parameters: x: numeric vector of data values y: numeric vector of data values or a character string which is used to name a

cumulative distribution function. ….: the parameters which are defined by the y value alternative: used to indicate the alternate hypothesis.
exact: usually NULL or it indicates a logic that an exact pvalue should be computed.

tol: an upper bound used for rounding off errors in the data values.
simulate.p.value: a logic that checks whether to use Monte Carlo method to compute the p-value.
B: an integer value that indicates the number of replicates to be created while using the Monte Carlo method. Step 1: At first install the required packages. For performing the K-S test we need to install the “dgof” package using the install.packages() function from the R console.
install.packages("dgof")
Step 2: After a successful installation of the package, load the required package in our R Script. for that purpose, use the library() function as follows:

loading the required package library("dgof")
Step 3: Use the rnorm() function and the runif() function to generate to samples say x and y. The rnorm() function is used to generate random variates while the runif() function is used to generate random deviates.
loading the required package library(dgof)
generating random variate sample 1 x
<- rnorm(50)
generating random deviates sample 2 y <- runif(30)
Step 4: Now perform the K-S test on these two samples. For that purpose, use the ks.test() of the dgof package. # loading the required package
library(dgof)

generating random variate # sample 1 x
<- rnorm(50)
generating random deviates # sample 2 y
<- runif(30)
performing the K-S Test # Do x and y come from # the same distribution? ks.test(x, y)
4. Results with snapshots of experiments with Graph for Q1.

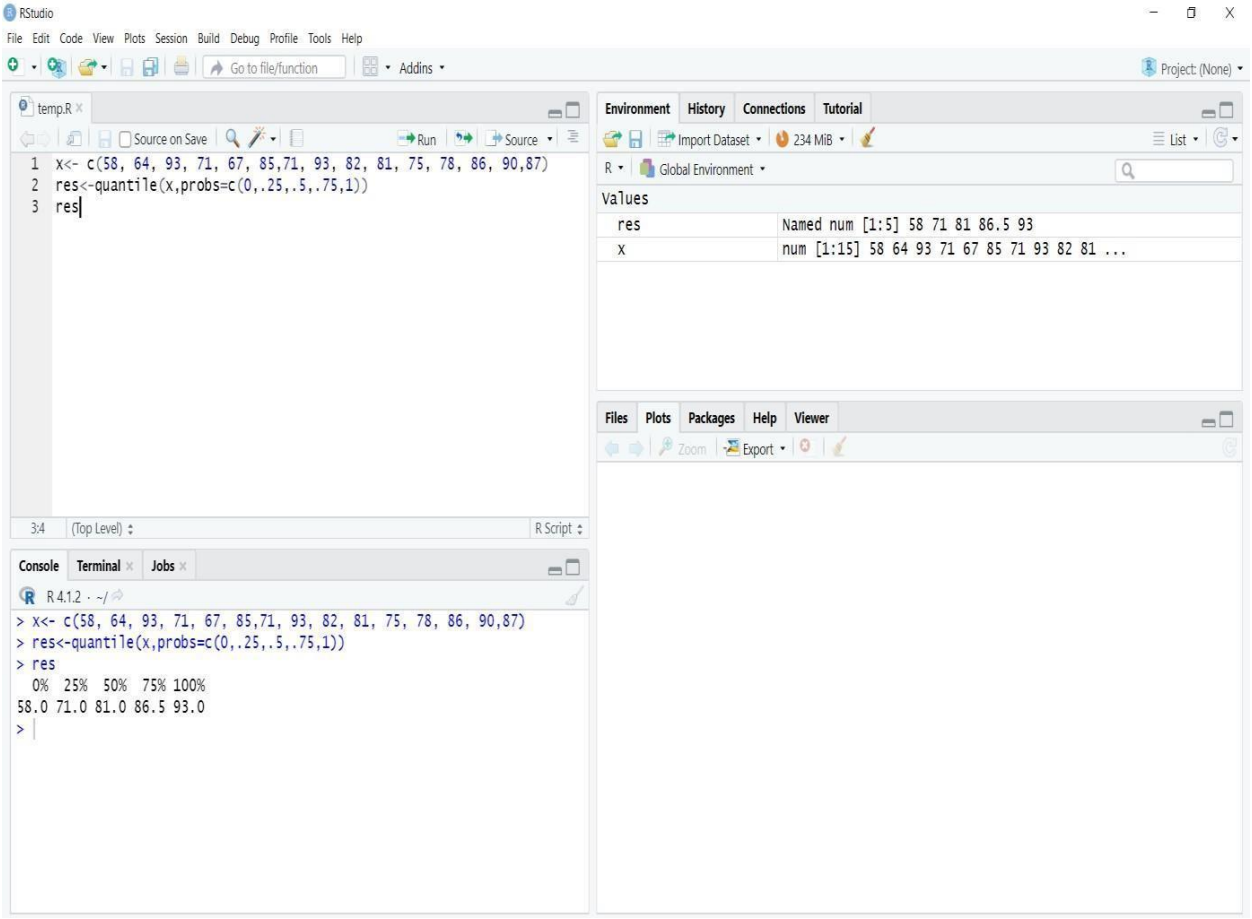


a) The Stat students of a reputed college have the following test scores: 58, 64, 93, 71, 67, 85,71, 93, 82, 81, 75, 78, 86, 90, and 87.

Determine and interpret the Quartiles of these scores.

Answer:

1. To determine Quartiles of given scores use the quantile ( ) in R console.
2. For interpretation after calculating Quartiles based on results we analyze our results.
3. For Eg. If Q= 0% and corresponding value is 60 means no scores fall below the score 60. If Q=50% means, from the given scores half of them are located above and half will located below the score 50. Likewise you have to interpret the results.



Ananlysis:

Therefore Q1=0% implies that, none of the scores fall below or equal to 58.0.

Q2=25%, implies that one-fourth of the data are below or equal to 71.0

Q3=50% is the median, and thus half of the scores are below or equal to 81.0, while the other half, are above 81.0

Q4=75%, implies that three-fourth of the data are below or equal to 86.5, while the remaining one-fourth are above 86.5.

And the minimum and maximum values are 58.0 and 93.0, respectively.

b) The surveyed weights (in kilogram) of the students in Stat dept. were the following: 50, 65,72, 62, 77, 84, 57, 74, 66, 68, 75, 58, 52, 69, and 87.

Compute and interpret the Deciles of these weights.

Answer:

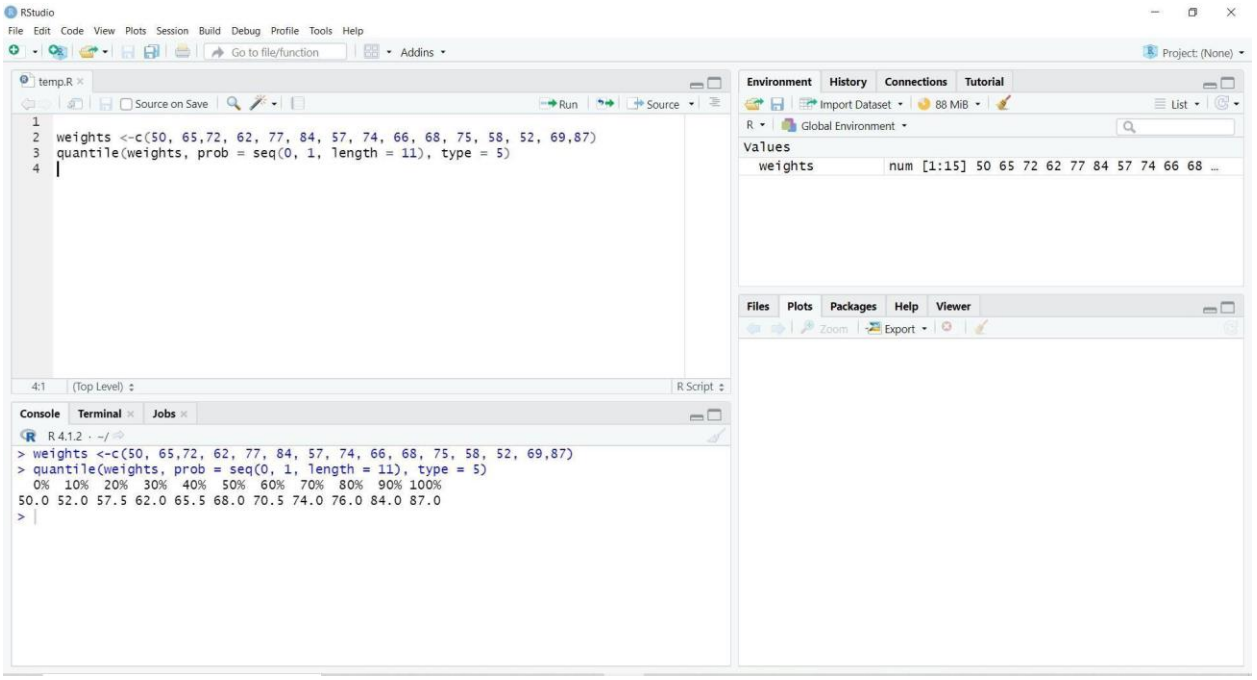
1. Load the weights of the students in with a variable in R.
2. Calculate deciles with quantile(weights, prob = seq(0 , 1 , length =), type =)

From the above function

Weights - weights of the students.

Prob - position of the data point.

Seq - used for prob's value that is from 0 to 1 of length length - number of weights in addition with 0, because the value 0 is the minimum of the data points.

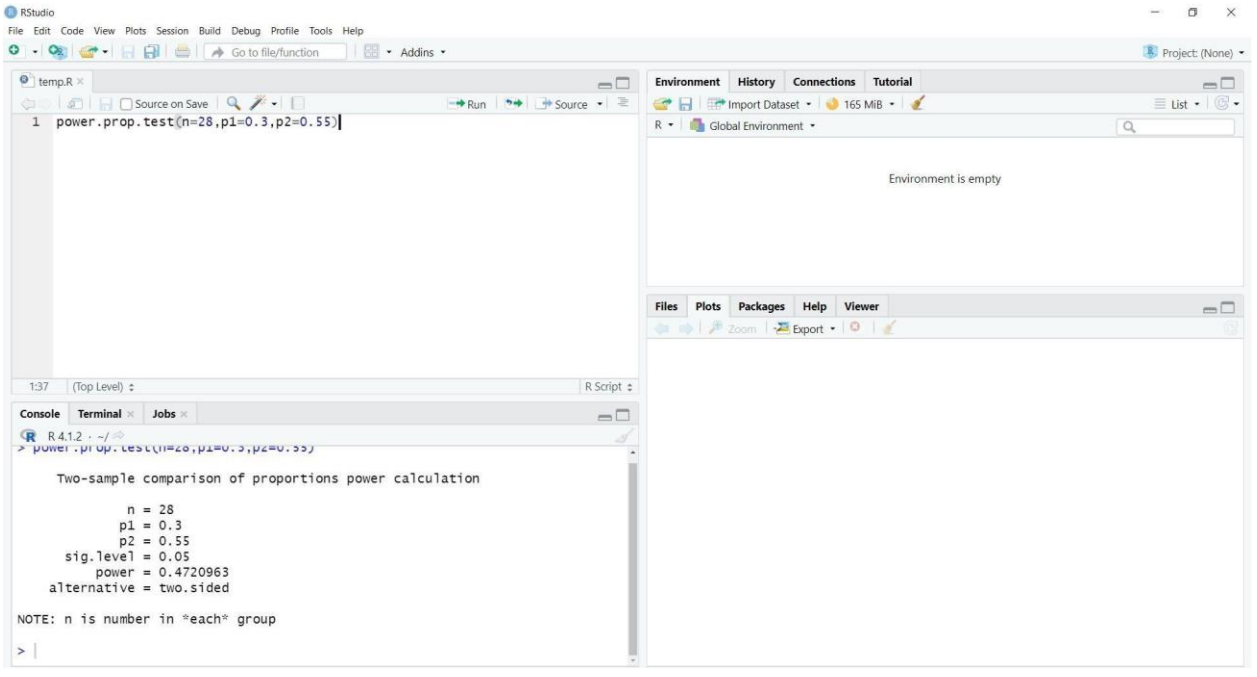


Analysis:

The first decile is D1=10%, implies that one-tenth of the weights fall below or equal to 52.0, and the remaining ninetenth fall above 52.0. The D5=50% is the median, thus half of the students' weights weigh below or equal to 68.0, while the other half fall above it. And so on.

c) Compute the power of a study to show a difference between group 1 (n=28) in which the event probability is 30% and group 2 (n=28) in which the event probability is 55%.

Answer:



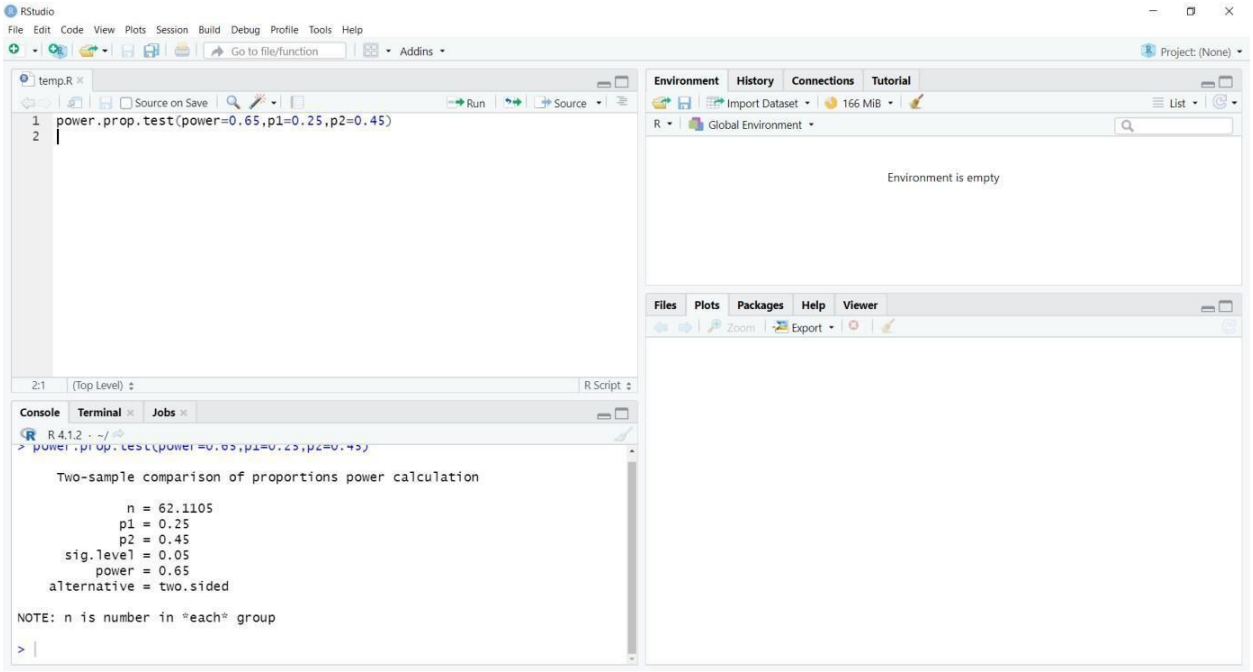
Conclusion:

The power of a study which includes 28 subjects in each of two experimental groups to see a difference between event probabilities is 48% under the assumption that the event probabilities are 30% in group 1 and 55% in group

2.

d) Compute the sample size of a study to show a difference between group 1 in which the event probability is 25% and group 2 in which the event probability is 45% with a power of 65%.

Answer:



Conclusion: In order to achieve a power of 65% under the assumed event probabilities the study should include at least 63 subjects in each of the experimental groups.

LAB ASSIGNMENT-A6

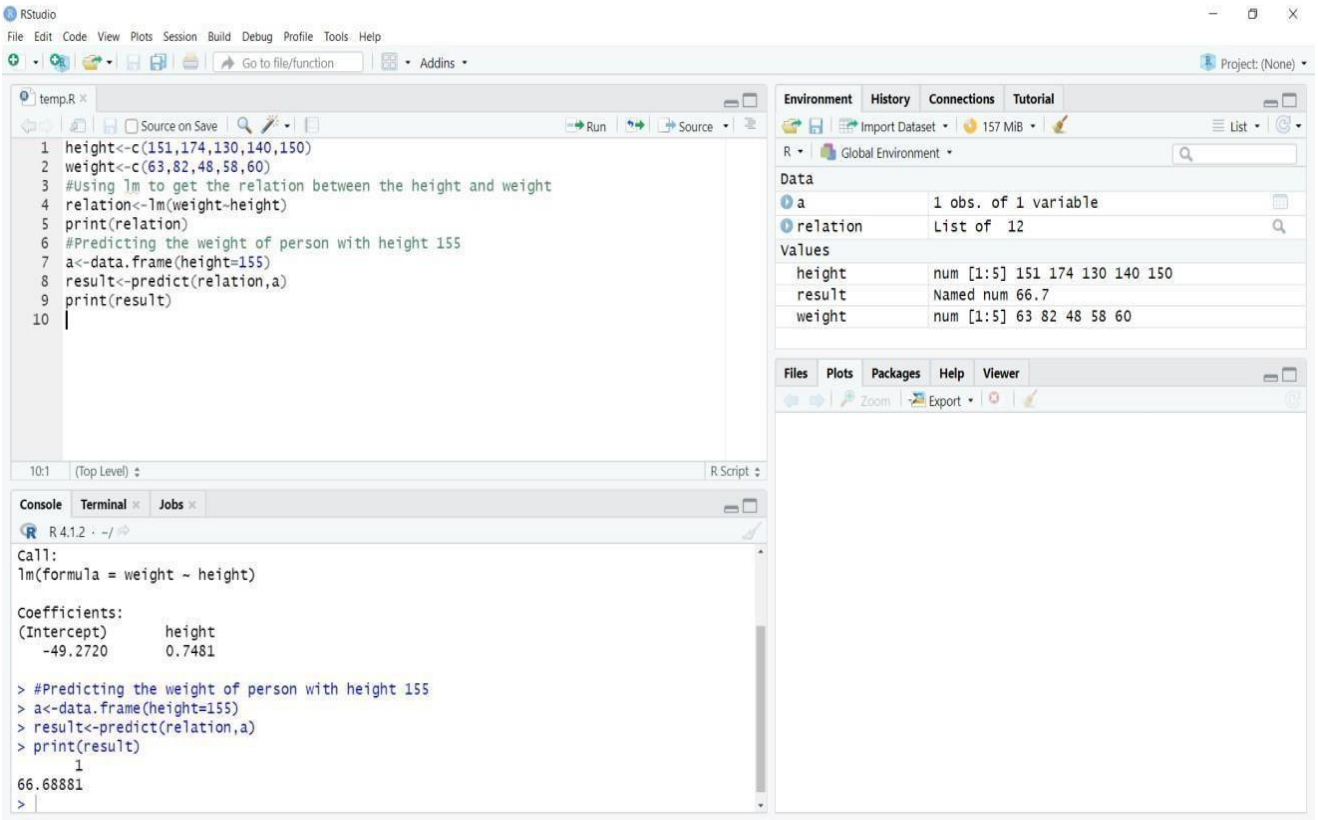
1. Write a R Program based on linear regression model on given data set X=height and Y=weight, where predicted weight of a person having height is 155.

   Dataset: -

Height	151	174	130	140	150
Weight	63	82	48	58	60

   Answer: R-Code snippet:

```
height<-
c(151,174,130,140,150)
weight<-c(63,82,48,58,60)
#Using lm to get the relation between the height and weight
relation<-lm(weight~height)
print(relation)
#Predicting the weight of person with height
155 a<-data.frame(height=155) result<-
predict(relation,a) print(result)
```



Hence weight of the person with height 155is 66.68881.

2. Answer all by examples and with R language specified functions/library if required.
   a. Dependent and independent variable:
      - ❾The “dependent variable” represents the output or effect, or is tested to see if it is the effect. A dependent variable is the variable that changes as a result of the independent variable manipulation. It’s the outcome you’re interested in measuring, and it “depends” on your independent variable.
      - ❾The “independent variables” represent the inputs or causes, or are tested to see if they are the cause.An independent variable is the variable you manipulate or vary in an experimental study to explore its effects. It’s called “independent” because it’s not influenced by any other variables in the study.

      Example:
      Let us consider Y=3X+5, here the value of X is independent of any other variable, but the value of Y is dependent on the value of X , therefore X is independent variable and Y is dependent variable.

b. Dummy variables:

❾Dummy variable in R programming is a type of variable that represents a characteristic of an experiment. A dummy variable is either 1 or 0 and 1 can be represented as either True or False and 0 can be represented as False or True depending upon the user. This variable is used to categorize the characteristic of an observation.

❾We can create dummy variables in R using 2 methods, ifelse( ) function or by using dummy_cols( ) function.

❾Using ifelse() function: ifelse() function performs a test and based on the result of the test return true value or false value as provided in the parameters of the function. Using this function, dummy variable can be created accordingly.

Syntax:

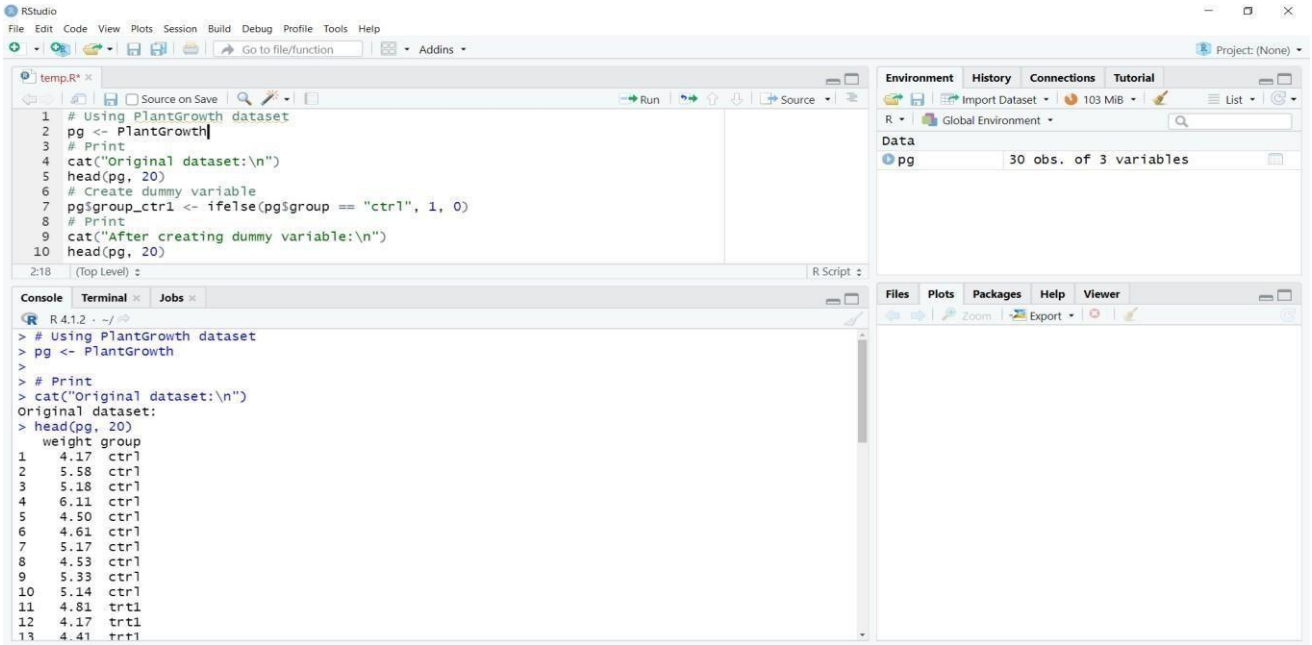
ifelse(test, yes, no)

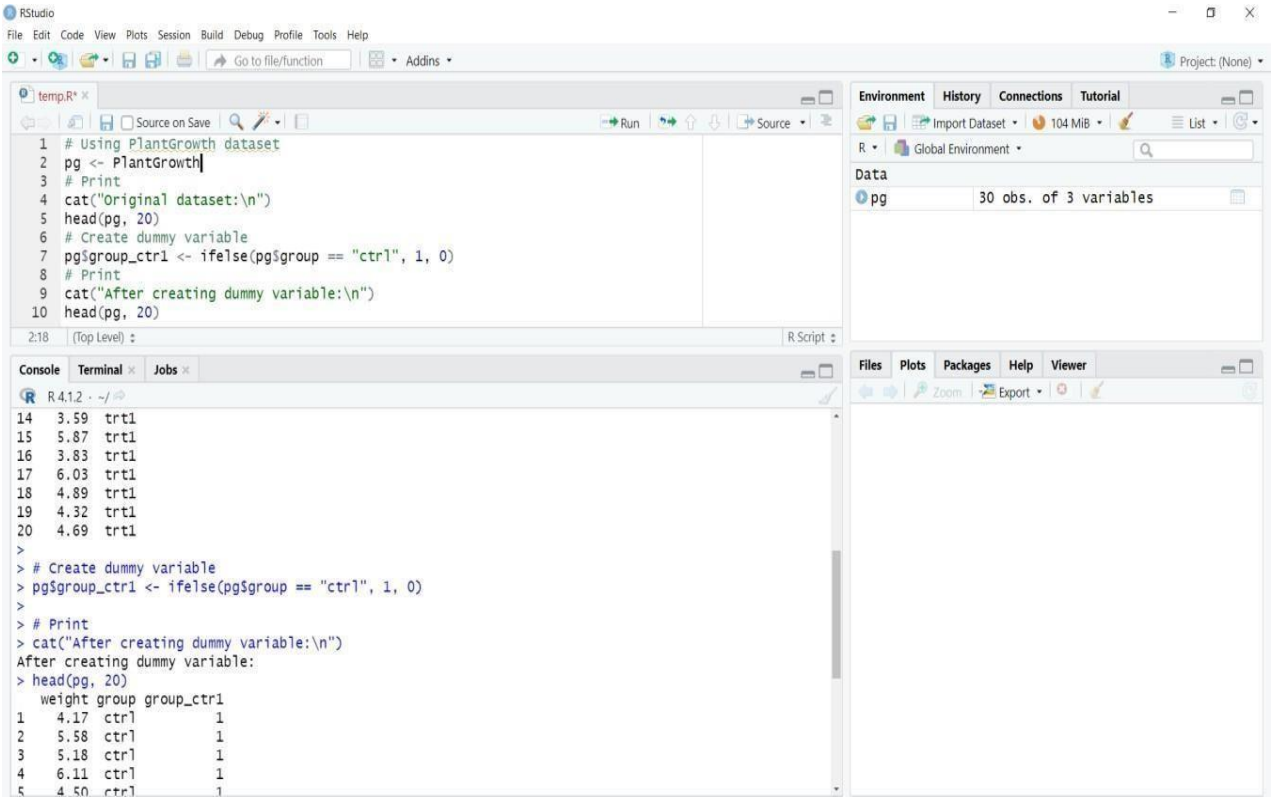
Parameters:

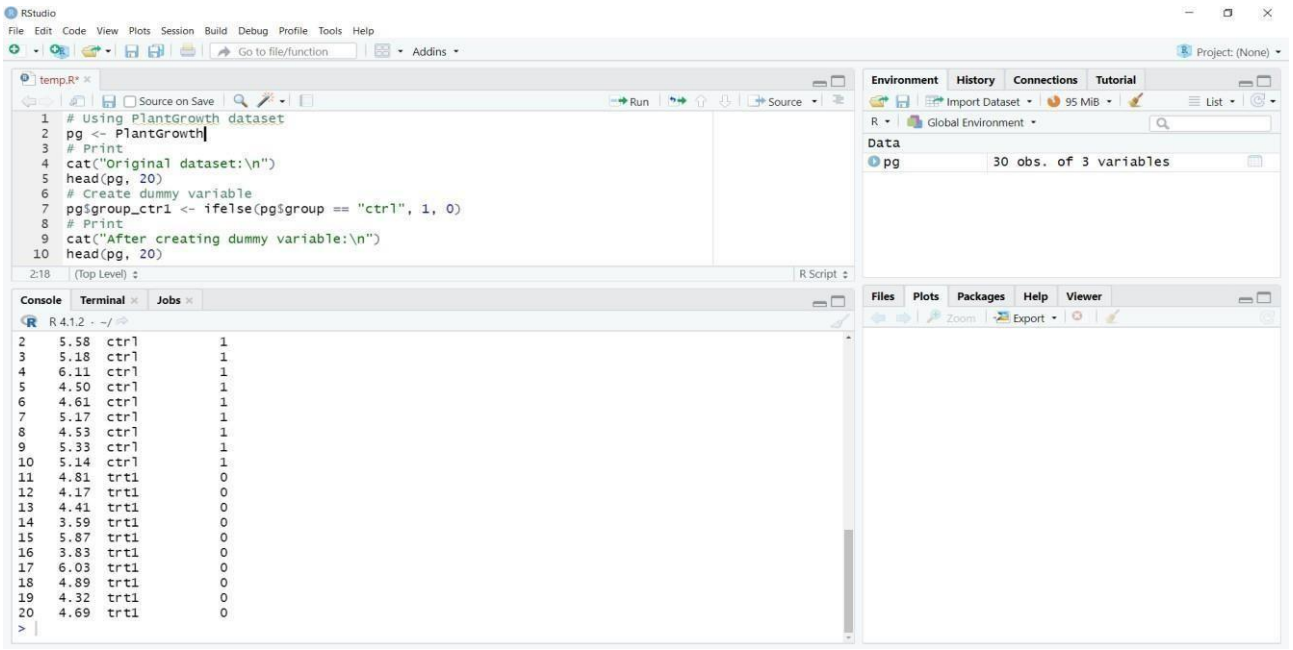
test: represents test condition. yes: represents the value which will be executed if test condition satisfies. no: represents the value which will be executed if test condition does not satisfies.

❾R-Code Snippet:

```
# Using PlantGrowth dataset
pg <- PlantGrowth
# Print
cat("Original dataset:\n")
head(pg, 20)
# Create dummy variable
pg$group_ctr1 <- ifelse(pg$group == "ctrl", 1, 0)
# Print cat("After creating dummy variable:\n")
head(pg,20)
```


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
temp.R*
Source on Save
Run
Source
1 # Using PlantGrowth dataset
2 pg <- PlantGrowth
3 # Print
4 cat("Original dataset:\n")
5 head(pg, 20)
6 # Create dummy variable
7 pg$group_ctrl <- ifelse(pg$group == "ctrl", 1, 0)
8 # Print
9 cat("After creating dummy variable:\n")
10 head(pg, 20)
2:18 (Top Level)
R Script
Console Terminal Jobs
R 4.1.2 · ~/
> # Using PlantGrowth dataset
> pg <- PlantGrowth
>
> # Print
> cat("Original dataset:\n")
Original dataset:
> head(pg, 20)
weight group
1 4.17 ctrl
2 5.58 ctrl
3 5.18 ctrl
4 6.11 ctrl
5 4.50 ctrl
6 4.61 ctrl
7 5.17 ctrl
8 4.53 ctrl
9 5.33 ctrl
10 5.14 ctrl
11 4.81 trt1
12 4.17 trt1
13 4.41 trt1
Environment History Connections Tutorial
Import Dataset
103 MiB
List
R
Global Environment
Data
pg
30 obs. of 3 variables
Files Plots Packages Help Viewer
Zoom
Export


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
temp.R*
Source on Save
Run
Source
1 # Using PlantGrowth dataset
2 pg <- PlantGrowth
3 # Print
4 cat("Original dataset:\n")
5 head(pg, 20)
6 # Create dummy variable
7 pg$group_ctrl <- ifelse(pg$group == "ctrl", 1, 0)
8 # Print
9 cat("After creating dummy variable:\n")
10 head(pg, 20)
2:18 (Top Level)
R Script
Console Terminal Jobs
R 4.1.2 · ~/
14 3.59 trt1
15 5.87 trt1
16 3.83 trt1
17 6.03 trt1
18 4.89 trt1
19 4.32 trt1
20 4.69 trt1
>
> # Create dummy variable
> pg$group_ctrl <- ifelse(pg$group == "ctrl", 1, 0)
>
> # Print
> cat("After creating dummy variable:\n")
After creating dummy variable:
> head(pg, 20)
weight group group_ctrl
1 4.17 ctrl 1
2 5.58 ctrl 1
3 5.18 ctrl 1
4 6.11 ctrl 1
5 4.50 ctrl 1
Environment History Connections Tutorial
Import Dataset
104 MiB
List
R
Global Environment
Data
pg
30 obs. of 3 variables
Files Plots Packages Help Viewer
Zoom
Export



❾Using dummy_cols() function

dummy_cols() function is present in fast Dummies package. It creates dummy variables on the basis of parameters provided in the function. If columns are not selected in the function call for which dummy variable has to be created, then dummy variables are created for all characters and factors column in the dataframe.

Syntax: dummy_cols(.data, select_columns

= NULL) Parameters:

.data: represents object for which dummy columns has to be created.

select_columns: represents columns for which dummy variables has to be created.

Example :

Create a dataframe df <- data.frame(gender = c("m", "f",

"m"), age = c(19, 20, 20),

city = c("Delhi", "Mumbai", "Delhi"))

Create dummy variables

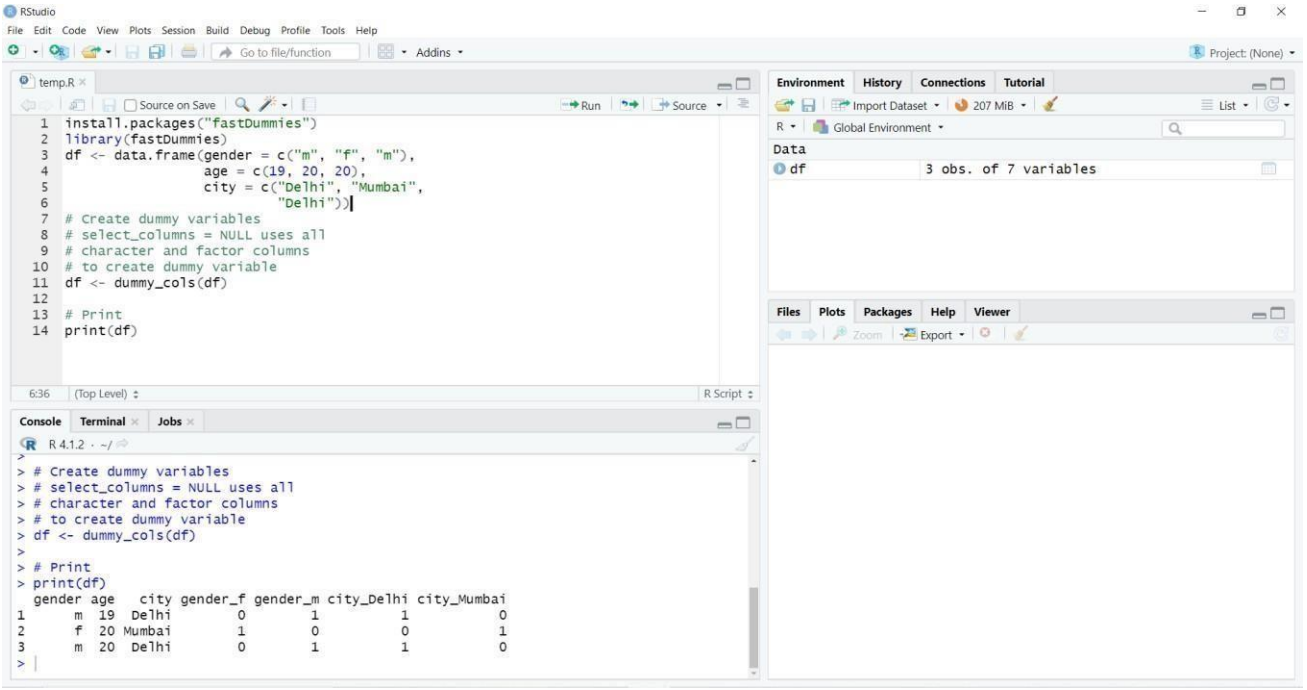
select_columns = NULL uses all

character and factor columns

to create dummy variable df

<- dummy_cols(df)

Print print(df)



c. Mean value with an example

❾It is calculated by taking the sum of the values and dividing with the number of values in a data series. ❾The function mean() is used to calculate this in R.

Syntax:

The basic syntax for calculating mean in R is −

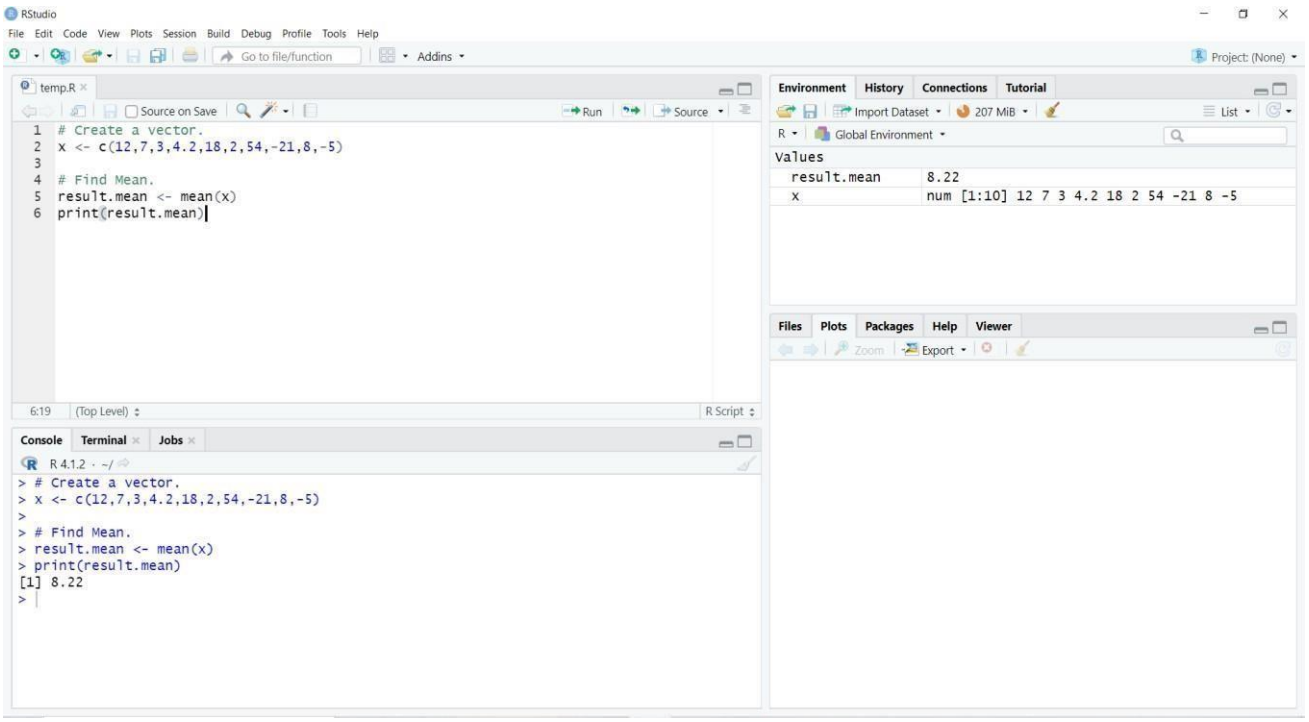
mean(x, trim = 0, na.rm = FALSE, ...) Following is the description of the parameters used :

❾x is the input vector.

- ❾trim is used to drop some observations from both end of the sorted vector.
- ❾na.rm is used to remove the missing values from the input vector.

Example:

```
# Create a vector. x <-
c(12,7,3,4.2,18,2,54,-21,8,-5)      #
Find Mean.
  result.mean <- mean(x)
  print(result.mean)
```



Applying Trim Option:

When trim parameter is supplied, the values in the vector get sorted and then the required numbers of observations are dropped from calculating the mean.

When trim = 0.3, 3 values from each end will be dropped from the calculations to find mean.

In this case the sorted vector is (−21, −5, 2, 3, 4.2, 7, 8, 12, 18, 54) and the values removed from the vector for calculating mean are (−21,−5,2) from left and (12,18,54) from right.

Example:

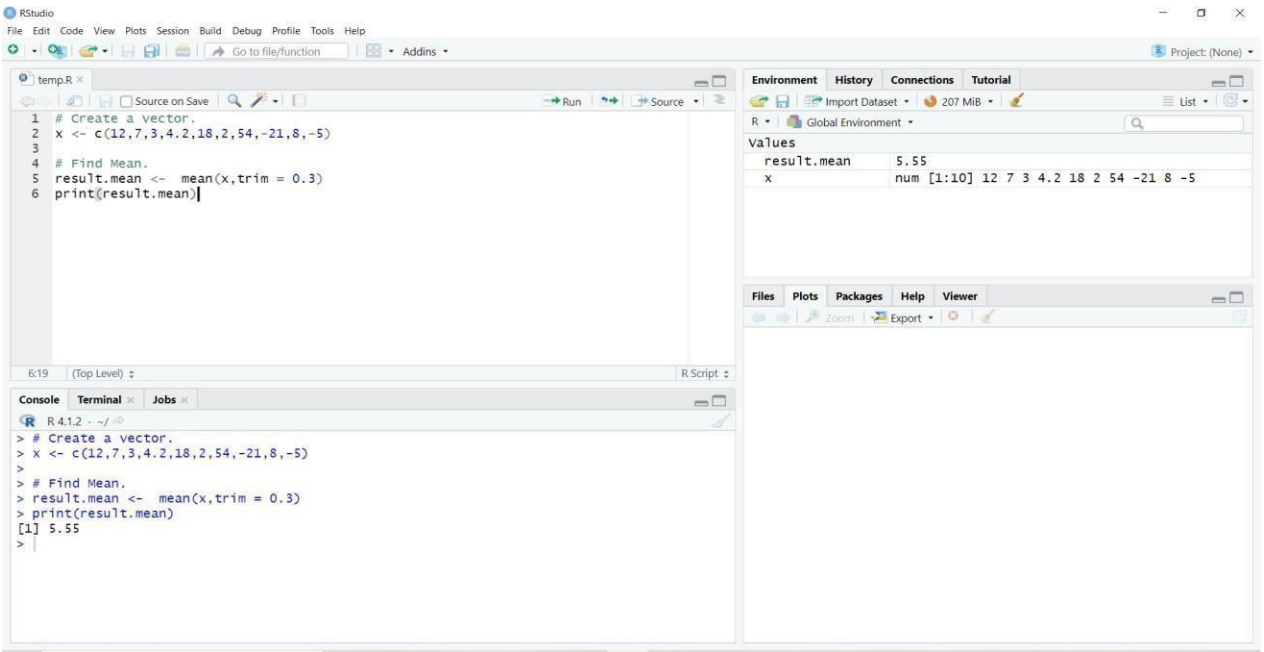
Create a vector. x <-

c(12,7,3,4.2,18,2,54,-21,8,-5) #

Find Mean. result.mean <-

mean(x,trim = 0.3)

print(result.mean)

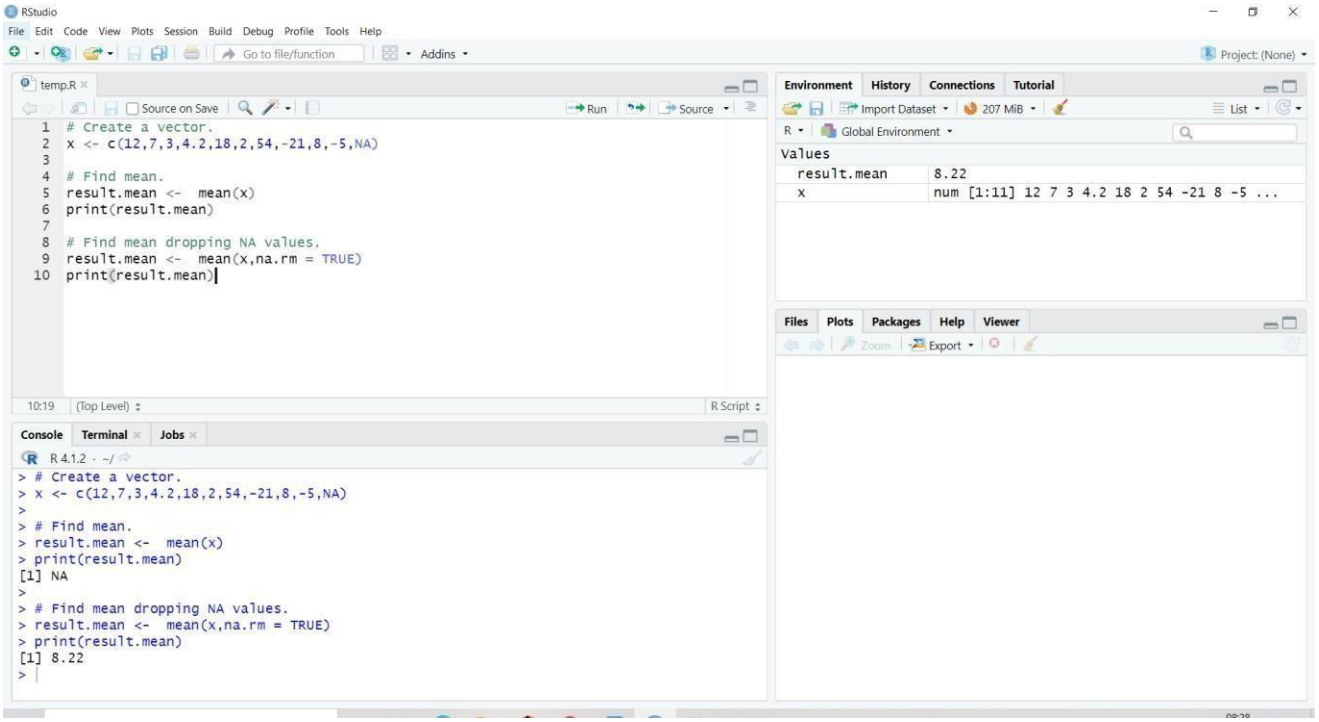


Applying NA Option:

If there are missing values, then the mean function returns NA.

To drop the missing values from the calculation use na.rm = TRUE. which means remove the NA values.

Example:



d. Least Square Variables:

- ❾Least square regression is a technique that helps us draw a line of best fit depending on the data points.The line is called the least square regression line., which perfectly depicts the changes in y(dependent variables) and their corresponding x(independent variables). This line is used to predict the value of y for particular value of x.Having both the dependent variables and independent variables is the first requirement of any regression technique.
- ❾The least-squares method is a crucial statistical method that is practised to find a regression line or a best-fit line for the given pattern. This method is described by an equation with specific parameters. The method of least squares is generously used in evaluation and regression. In regression analysis, this method is said to be a standard approach for the approximation of sets of

equations having more equations than the number of unknowns.

The method of least squares actually defines the solution for the minimization of the sum of squares of deviations or the errors in the result of each equation.

Let us assume that the given points of data are (x_1, y_1), (x_2, y_2), …, (x_n, y_n) in which all x's are independent variables, while all y's are dependent ones. Also, suppose that f(x) be the fitting curve and d represents error or deviation from each given point.

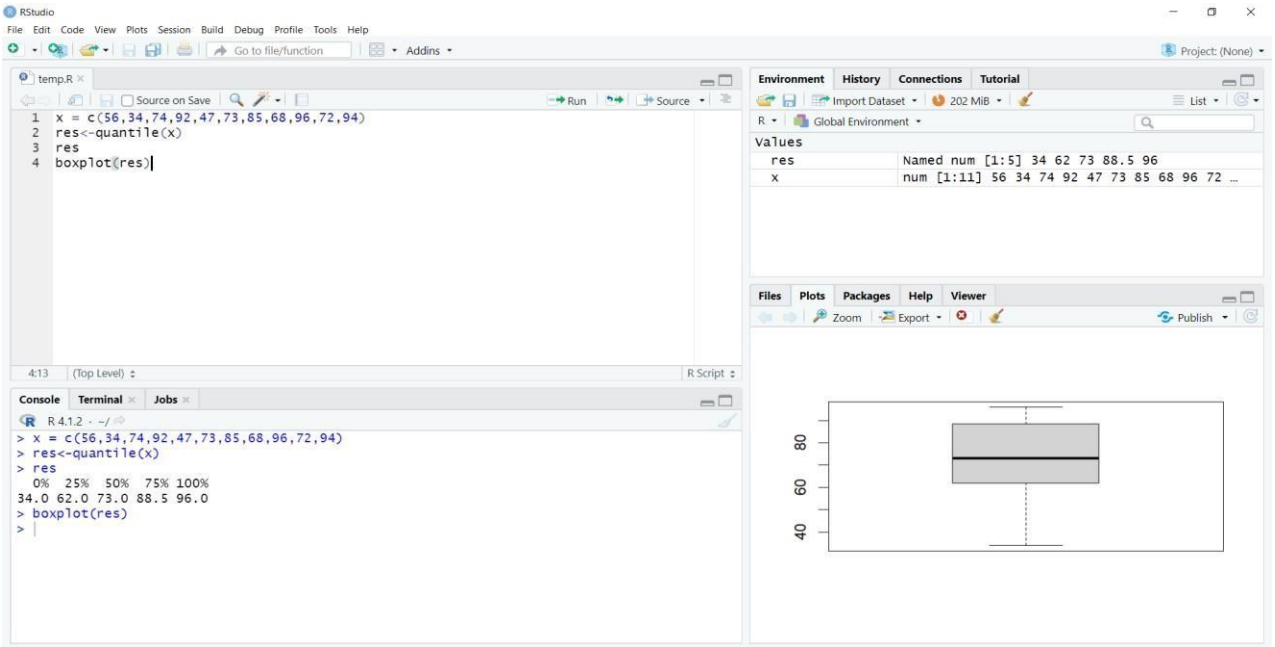
The least-squares explain that the curve that best fits is represented by the property that the sum of squares of all the deviations from given values must be minimum.

❾ The premise of a regression model is to examine the impact of one or more independent variables (in this case time spent writing an essay) on a dependent variable of interest (in this case essay grades). Linear regression analyses such as these are based on a simple equation: $Y = a + bX$

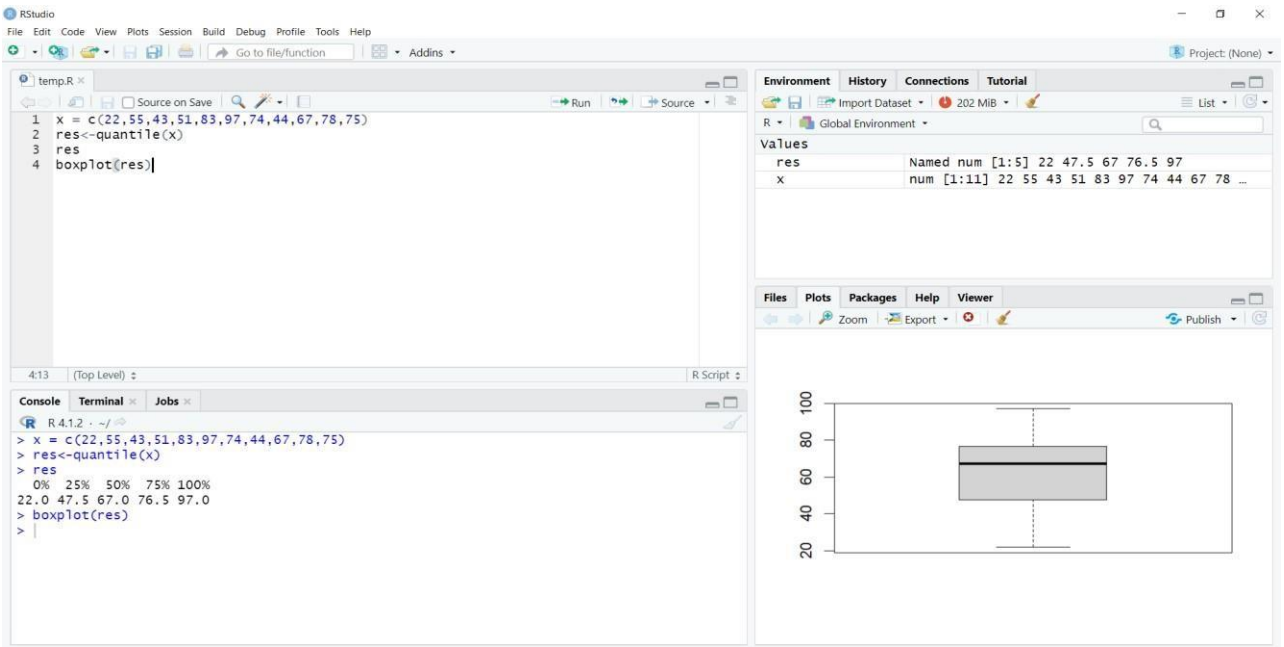
Here Y is dependent variable and X is independent variable.

LAB ASSIGNMENT-A7

1. Calculate and visualize the Quartiles of the following set of data using R programming.
   a. 56,34,74,92,47,73,85,68,96,72,94



   b. 22,55,43,51,83,97,74,44,67,78,75



2. How to calculate the percentiles of set of student marks at a given probabilities and visualize them through bar-chart using R programming.

First, we need to store the marks of the students in a vector to combine the list of marks of the students. To calculate the percentiles of set of student marks at a given probabilities , we use quantile function i.e.,

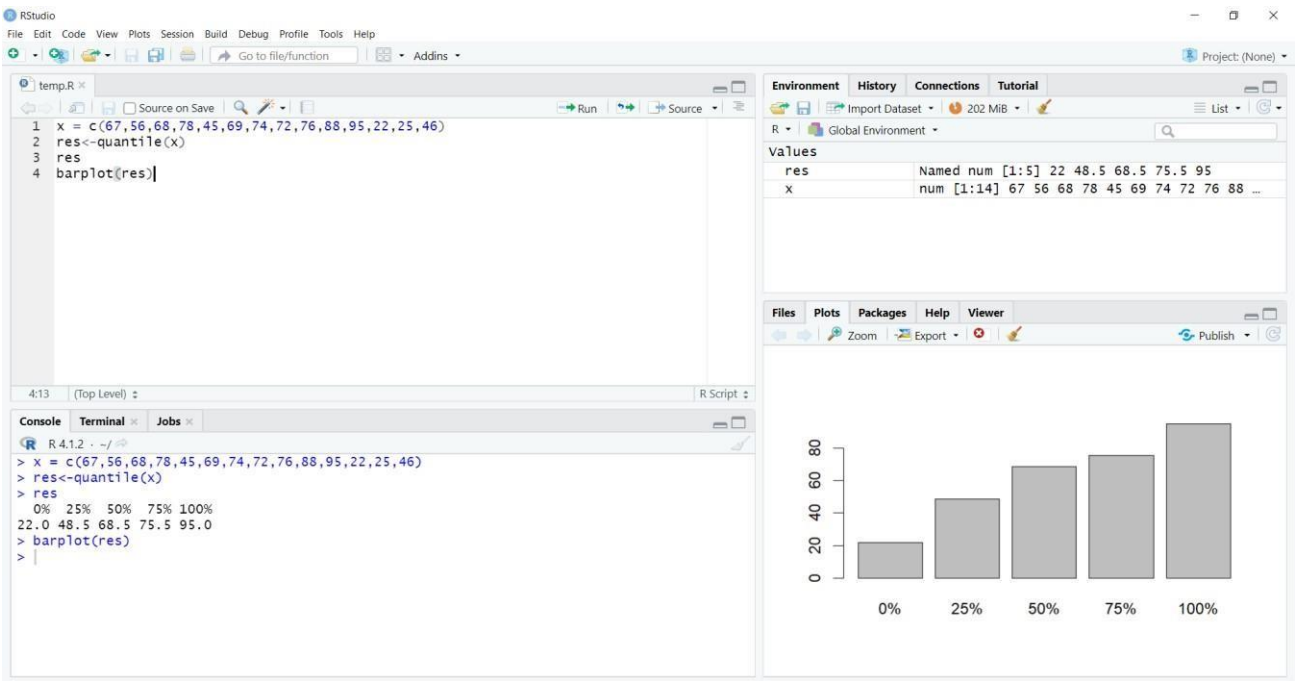
quantile( ) -the quantile function, associated with a probability distribution of a random variable, specifies the value of the random variable such that the probability of the variable being less than or equal to that value equals the given probability.

Now, we need to store the quantile so calculated in a varaible.

To visualize the percentiles of the set of student marks so obtained using the quantile function, we should use the barplot() function.

R uses the function barplot() to create bar charts

a. 67,56,68,78,45,69,74,72,76,88,95,22,25,46.



b. 34,78,88,89,56,67,65,54,64,73,84,86,82,61