Practical no: 01

Aim: using R execute the basic commands , array, list and frames and vectors.

R Command Prompt:

This will launch R interpreter and you will get a prompt > where you can start typing your program as follows −

```
> myString <- "Hello, World!"

> print ( myString)

[1] "Hello, World!"
```

Here first statement defines a string variable myString, where we assign a string "Hello, World!" and then next statement print() is being used to print the value stored in variable myString.

R Script File

Usually, you will do your programming by writing your programs in script files and then you execute those scripts at your command prompt with the help of R interpreter called Rscript. So let's start with writing following code in a text file called test.R as under –

```
# My first program in R Programming

myString <- "Hello, World!"


print ( myString)
```

Save the above code in a file test.R and execute it at Linux command prompt as given below. Even if you are using Windows or other system, syntax will remain same.

FACTORS :

Factors are the r-objects which are created using vector. It stores the vector along with the distinct values of the elements in the vector as labels. Labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. They are useful in statistical modeling

Factor created using factor

>Apple_colors<-c('green','green','yellow','red','red','red','green')

#create a factor object

>Factor_apple<-factor(apple_colors)

#Print the factor

>Print(factor_apple)

>Print(nlevels(factor_apple))

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.

#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","2013-09-20","2014-11-15","2014-05-11","2015-03-27")),

+ stringsAsFactors=FALSE)

#Print the data frame

> print(emp.data)

#Get the structure of data frame

The structure of the data frame can be seen by using str() function

> str(emp.data)

#Extract data from data frame

Extract specific column from a data frame using column name

> result<-data.frame(emp.data$emp_name,emp.data$salary)

> print(result)

#Extract first two rows

> result<-emp.data[1:2,]

> print(result)

#Extract 3rd and 5th row with 2nd and 4th column

> result<-emp.data[c(3,5),c(2,4)]

> print(result)

LIST

Create alist containing strings ,numbers,vectors and logical values

> list_data<-list("RED","GREEN",c(21,32,11),TRUE,51.23,119.1)

> print(list_data)

Naming list element

The list elements can be given names and they can be accessed using these names

#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")
#show the list
> print(list_data)
#Access the first elements' of the list
> print(list_data[1])
#Access the list elements using the name of the elements
> print(list_data$A_Matrix)
```

Manipulating list elements

We can add ,delete, and update list elements only at the end of a list. But we can update any element.

```
#Create a list containing a vector a matrix and 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")
#Add elements at the end of the list
> list_data[4]<-"New element"
> print(list_data[4])
#Remove the last element
> list_data[4]<-NULL
```

Update the 3rd element

```
> list_data[3]<-"Updated element"
> print(list_data[3])
```

Merging lists

You can merge the lists one list by placing all the lists inside one list() function

#Create two lists

> list1<-list(1,2,3)

> list2<-list("SUN","MON","TUE")

#Merge the two lists

> merged.list<-c(list1,list2)

#Print the merged list

> print(merged.list)

Converting list to vector

A list can be converted to a vector so that the elements of the vector can be used for further manipulation. To do this conversion we use the unlist() function. It takes the list as input and produces vector

#Create lists

> list1<-list(1:5)

> print(list1)

> list2<-list(10:14)

> print(list2)

#Convert the list to vector

> v1<-unlist(list1)

> v2<-unlist(list2)

> print(v1)

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.

```
# 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
```

```
, , 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
```

<u>Basic DataTypes:</u>

<u>Numeric</u>

Decimal values are called numerics in R. It is the default computational data type. If we assign a decimal value to a variable x as follows, x will be of numeric type.

```
> x = 10.5       # assign a decimal value

> x              # print the value of x

[1] 10.5

> class(x)       # print the class name of x

[1] "numeric"
```

Furthermore, even if we assign an integer to a variable k, it is still being saved as a numeric value.

```
> k = 1

> k              # print the value of k

[1] 1
```

```
> class(k)       # print the class name of k
[1] "numeric"
```

The fact that k is not an integer can be confirmed with the is.integer function. We will discuss how to create an integer in our next tutorial on the integer type.

```
> is.integer(k)  # is k an integer?
[1] FALSE
```

Integer

In order to create an integer variable in R, we invoke the integer function. We can be assured that y is indeed an integer by applying the is.integer function.

```
> y = as.integer(3)
> y              # print the value of y
[1] 3
> class(y)       # print the class name of y
[1] "integer"
> is.integer(y)  # is y an integer?
[1] TRUE
```

Complex

A complex value in R is defined via the pure imaginary value i.

```
> z = 1 + 2i     # create a complex number
> z              # print the value of z
[1] 1+2i
> class(z)       # print the class name of z
[1] "complex"
```

The following gives an error as −1 is not a complex value.

```
> sqrt(−1)       # square root of −1
[1] NaN
```

Instead, we have to use the complex value −1 + 0i.

```
> sqrt(−1+0i)    # square root of −1+0i
[1] 0+1i
```

Logical

A logical value is often created via comparison between variables.

```
> x = 1; y = 2   # sample values
> z = x > y      # is x larger than y?
> z              # print the logical value
[1] FALSE
> class(z)       # print the class name of z
[1] "logical"
```

Standard logical operations are "&" (and), "|" (or), and "!" (negation).

```
> u = TRUE; v = FALSE
> u & v          # u AND v
[1] FALSE
> u | v          # u OR v
[1] TRUE
> !u             # negation of u
[1] FALSE
```

Character

A character object is used to represent string values in R. We convert objects into character values with the as.character() function:

```
> x = as.character(3.14)
> x              # print the character string
[1] "3.14"
> class(x)       # print the class name of x
```

```
[1] "character"
```

Two character values can be concatenated with the paste function.

```
> fname = "Joe"; lname ="Smith"
> paste(fname, lname)
[1] "Joe Smith"
```

However, it is often more convenient to create a readable string with the sprintf function, which has a C language syntax.

```
> sprintf("%s has %d dollars", "Sam", 100)
[1] "Sam has 100 dollars"
```

To extract a substring, we apply the substr function. Here is an example showing how to extract the substring between the third and twelfth positions in a string.

```
> substr("Mary has a little lamb.", start=3, stop=12)
[1] "ry has a l"
```

And to replace the first occurrence of the word "little" by another word "big" in the string, we apply the sub function.

```
> sub("little", "big", "Mary has a little lamb.")
[1] "Mary has a big lamb."
```

<u>Vector</u>

A vector is a sequence of data elements of the same basic type. Members in a vector are officially called components. Nevertheless, we will just call them members in this site.

Here is a vector containing three numeric values 2, 3 and 5.

```
> c(2, 3, 5)
[1] 2 3 5
```

And here is a vector of logical values.

```
> c(TRUE, FALSE, TRUE, FALSE, FALSE)
[1]  TRUE FALSE  TRUE FALSE FALSE
```

A vector can contain character strings.

```
> c("aa", "bb", "cc", "dd", "ee")
[1] "aa" "bb" "cc" "dd" "ee"
```

Incidentally, the number of members in a vector is given by the length function.

```
> length(c("aa", "bb", "cc", "dd", "ee"))
[1] 5
```

Combining Vectors

Vectors can be combined via the function c. For examples, the following two vectors n and s are combined into a new vector containing elements from both vectors.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc", "dd", "ee")
> c(n, s)
[1] "2"  "3"  "5"  "aa" "bb" "cc" "dd" "ee"
```

Recycling Rule

If two vectors are of unequal length, the shorter one will be recycled in order to match the longer vector. For example, the following vectors u and v have different lengths, and their sum is computed by recycling values of the shorter vector u.

```
> u = c(10, 20, 30)
> v = c(1, 2, 3, 4, 5, 6, 7, 8, 9)
> u + v
[1] 11 22 33 14 25 36 17 28 39
```

Vector Index

We retrieve values in a vector by declaring an index inside a *single square bracket* "[]" operator.
For example, the following shows how to retrieve a vector member. Since the vector index is 1-based, we use the index position 3 for retrieving the third member.

```
> s = c("aa", "bb", "cc", "dd", "ee")
> s[3]
[1] "cc"
```

Unlike other programming languages, the square bracket operator returns more than just individual members. In fact, the result of the square bracket operator is another vector, and s[3] is a vector **slice** containing a single member "cc".

Negative Index

If the index is negative, it would strip the member whose position has the same absolute value as the negative index. For example, the following creates a vector slice with the third member removed.

```
> s[-3]
[1] "aa" "bb" "dd" "ee"
```

Out-of-Range Index
If an index is out-of-range, a missing value will be reported via the symbol NA.

```
> s[10]
[1] NA
```

Numeric Index Vector

A new vector can be sliced from a given vector with a **numeric index vector**, which consists of member positions of the original vector to be retrieved.

Here it shows how to retrieve a vector slice containing the second and third members of a given vector s.

```
> s = c("aa", "bb", "cc", "dd", "ee")
> s[c(2, 3)]
[1] "bb" "cc"
```

Duplicate Indexes

The index vector allows duplicate values. Hence the following retrieves a member twice in one operation.

```
> s[c(2, 3, 3)]
[1] "bb" "cc" "cc"
```

Out-of-Order Indexes

The index vector can even be out-of-order. Here is a vector slice with the order of first and second members reversed.

```
> s[c(2, 1, 3)]
[1] "bb" "aa" "cc"
```

Range Index

To produce a vector slice between two indexes, we can use the colon operator ":". This can be convenient for situations involving large vectors.

```
> s[2:4]
[1] "bb" "cc" "dd"
```

Named Vector Members

We can assign names to vector members.

For example, the following variable v is a character string vector with two members.

```
> v = c("Mary", "Sue")
> v
[1] "Mary" "Sue"
```

We now name the first member as First, and the second as Last.

```
> names(v) = c("First", "Last")
> v
```

```
 First   Last
"Mary"  "Sue"
```

Then we can retrieve the first member by its name.

```
> v["First"]
[1] "Mary"
```

Furthermore, we can reverse the order with a character string index vector.

```
> v[c("Last", "First")]
  Last  First
 "Sue" "Mary"
```

Vector Manipulation

Vector arithmetic

Two vectors of same length can be added, subtracted, multiplied or divided giving the result as a vector output.

```
> v<-c(2,5.5,6)

> t<-c(8,3,4)

> print(v+t)

[1] 10.0  8.5 10.0


> print(v-t)

[1] -6.0  2.5  2.0

> print(v*t)

[1] 16.0 16.5 24.0

> print(v/t)

[1] 0.250000 1.833333 1.500000

> print(v%%t)

[1] 2.0 2.5 2.0
```

Practical:2

Aim: Matrix operation

```
> M<-matrix(c(3:14),nrow=4,byrow=TRUE)
> print(M)
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    6    7    8
[3,]    9   10   11
[4,]   12   13   14
> N<-matrix(c(3:14),nrow=4,byrow=FALSE)
> print(N)
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
[4,]    6   10   14
> rownames=c("row1","row2","row3","row4")
> colnames=c("col1","col2","col3")
> p<-
matrix(c(3:14),nrow=4,byrow=TRUE,dimnames=list(rownames,colnames))
> print(p)
     col1 col2 col3
row1    3    4    5
row2    6    7    8
row3    9   10   11
row4   12   13   14
> print(p[1,3])
[1] 5
```

```
> print(p[4,2])
[1] 13
> print(p[2,])
col1 col2 col3
   6    7    8
> print(p[,3])
row1 row2 row3 row4
   5    8   11   14
> matrix1<-matrix(c(3,9,-1,4,2,6),nrow=2)
> print(matrix1)
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
> matrix2<-matrix(c(5,2,0,9,3,4),nrow=2)
> print(matrix2)
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4
>
> result<-matrix1+matrix2
> cat("Result of Addition ","\n")
Result of Addition
> print(result)
     [,1] [,2] [,3]
[1,]    8   -1    5
[2,]   11   13   10
>
```

```
> result<-matrix1-matrix2
> cat("Result of Sub ","\n")
Result of Sub
> print(result)
     [,1] [,2] [,3]
[1,]   -2   -1   -1
[2,]    7   -5    2

> result<-matrix1*matrix2
> cat("Result of Multiplication ","\n")
Result of Multiplication
> print(result)
     [,1] [,2] [,3]
[1,]   15    0    6
[2,]   18   36   24

> result<-matrix1/matrix2
> cat("Result of Division ","\n")
Result of Division
> print(result)
     [,1]      [,2]      [,3]
[1,]  0.6      -Inf 0.6666667
[2,]  4.5 0.4444444 1.5000000
```

Note:

%% operator= is used to find remainder between Vectors of two set.

M%*%t(M) operator= is used to calculate multiplication between matrix M amd its transpose t(M) by %*%

Practical no: 3

Aim:

Using R execute the statistical functions: mean, median, mode, quartiles, range, inter quartile range histogram.

Mean:

```
# 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)
```

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

```
[1] 8.22
```

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.

```
# 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)
```

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

```
[1] 5.55
```

Median:

```
# Create the vector.
```

```
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)

# Find the median.
median.result <- median(x)
print(median.result)
```

Mode

The mode is the value that has highest number of occurrences in a set of data. Unike mean and median, mode can have both numeric and character data.

R does not have a standard in-built function to calculate mode. So we create a user function to calculate mode of a data set in R. This function takes the vector as input and gives the mode value as output.

Example

```
# Create the function.
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

# Create the vector with numbers.
v <- c(2,1,2,3,1,2,3,4,1,5,5,3,2,3)

# Calculate the mode using the user function.
result <- getmode(v)
print(result)

# Create the vector with characters.
charv <- c("o","it","the","it","it")

# Calculate the mode using the user function.
result <- getmode(charv)
print(result)
```

Range:

```
x=c(45,43,46,48,51,46,50,47,46,45)

> max(x)-min(x)

[1] 8
```

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.

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

Syntax

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.

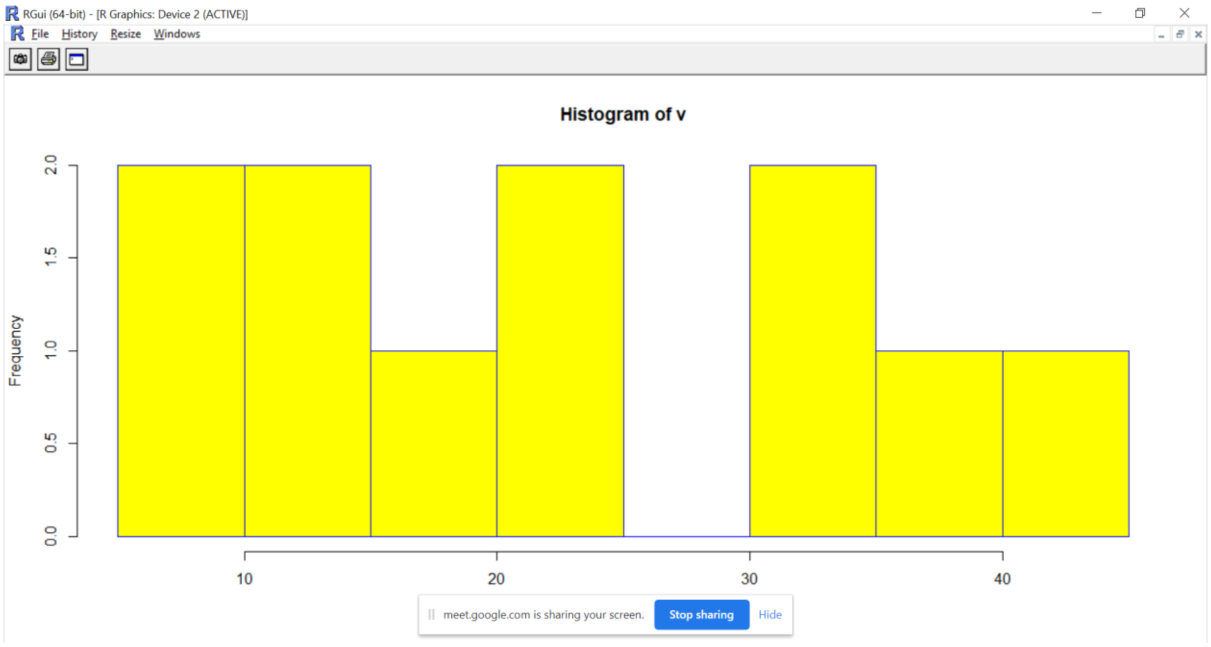
breaks is used to mention the width of each bar.

Example

A simple histogram is created using input vector, label, col and border parameters.

The script given below will create and save the histogram in the current R working directory.

```
# Create data for the graph.
v <-  c(9,13,21,8,36,22,12,41,31,33,19)

# Give the chart file a name.
png(file = "histogram.png")

# Create the histogram.
hist(v,xlab = "Weight",col = "yellow",border = "blue")

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

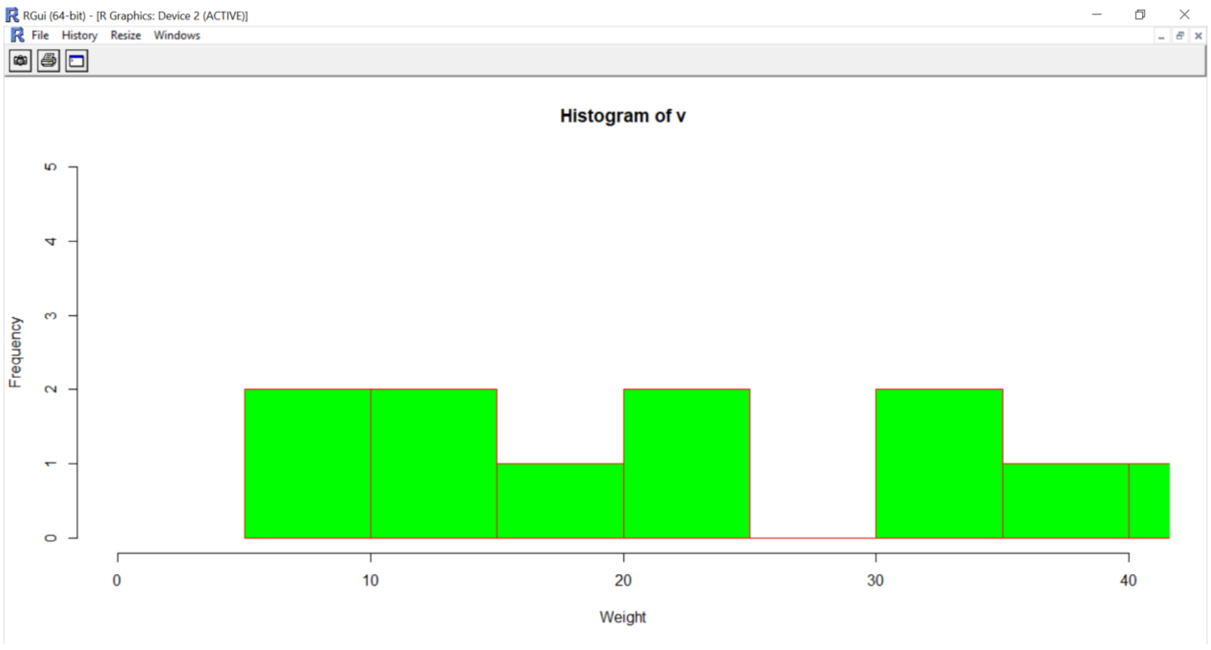

RGui (64-bit) - [R Graphics: Device 2 (ACTIVE)]
File History Resize Windows
Histogram of v
Frequency
0.0
0.5
1.0
1.5
2.0
10
20
30
40
meet.google.com is sharing your screen.
Stop sharing
Hide

Range of X and Y values

To specify the range of values allowed in X axis and Y axis, we can use the xlim and ylim parameters.

The width of each of the bar can be decided by using breaks.

```
# Create data for the graph.
v <- c(9,13,21,8,36,22,12,41,31,33,19)

# Give the chart file a name.
png(file = "histogram_lim_breaks.png")

# Create the histogram.
hist(v,xlab = "Weight",col = "green",border = "red", xlim = c(0,40), ylim =
c(0,5),
  breaks = 5)

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


RGui (64-bit) - [R Graphics: Device 2 (ACTIVE)]
File History Resize Windows
Histogram of v
Frequency
0
1
2
3
4
5
0
10
20
30
40
Weight

Practical 4:

Aim:

Using R import the data from Excel/. Csv file and perform the above function.

Reading a CSV File

Following is a simple example of **read.csv()** function to read a CSV file available in your current working directory −

```
data <- read.csv("input.csv")
print(data)
```

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

```
id,    name,     salary,   start_date,     dept
1        1     Rick      623.30    2012-01-01      IT
2        2     Dan       515.20    2013-09-23      Operations
3        3     Michelle  611.00    2014-11-15      IT
4        4     Ryan      729.00    2014-05-11      HR
5       NA     Gary      843.25    2015-03-27      Finance
6        6     Nina      578.00    2013-05-21      IT
7        7     Simon     632.80    2013-07-30      Operations
8        8     Guru      722.50    2014-06-17      Finance
```

1.Mean

The mean of an observation variable is a numerical measure of the central location of the data values. It is the sum of its data values divided by data count.

Hence, for a data sample of size *n,* its **sample mean** is defined as follows:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Similarly, for a data population of size *N*, the **population mean** is:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Problem

Find the mean eruption duration in the data set faithful.

Solution

We apply the mean function to compute the mean value of eruptions.

```
> duration = faithful$eruptions     # the eruption durations
> mean(duration)                    # apply the mean function
[1] 3.4878
```

Answer

The mean eruption duration is 3.4878 minutes.

2.Median

The median of an observation variable is the value at the middle when the data is sorted in ascending order. It is an ordinal measure of the central location of the data values.

Problem

Find the median of the eruption duration in the data set faithful.

Solution

We apply the median function to compute the median value of eruptions.

```
> duration = faithful$eruptions     # the eruption durations
> median(duration)                  # apply the median function
[1] 4
```

Answer

The median of the eruption duration is 4 minutes.

3.Quartile

There are several quartiles of an observation variable. The first quartile, or lower quartile, is the value that cuts off the first 25% of the data when it is sorted in ascending order. The second quartile, or median, is the value that cuts off the first 50%. The third quartile, or upper quartile, is the value that cuts off the first 75%.

Problem

Find the quartiles of the eruption durations in the data set faithful.

Solution

We apply the quantile function to compute the quartiles of eruptions.

```
> duration = faithful$eruptions     # the eruption durations
> quantile(duration)               # apply the quantile function
   0%    25%    50%    75%   100%
1.6000 2.1627 4.0000 4.4543 5.1000
```

Answer

The first, second and third quartiles of the eruption duration are 2.1627, 4.0000 and 4.4543 minutes respectively.

4.Range

The range of an observation variable is the difference of its largest and smallest data values. It is a measure of how far apart the entire data spreads in value.

Range = Largest Value− Smallest Value

Problem

Find the range of the eruption duration in the data set faithful.

Solution

We apply the max and min function to compute the largest and smallest values of eruptions, then take the difference.

```
> duration = faithful$eruptions     # the eruption durations
> max(duration) − min(duration)     # apply the max and min functions
[1] 3.5
```

Answer

The range of the eruption duration is 3.5 minutes.

5.Interquartile Range

The interquartile range of an observation variable is the difference of its upper and lower quartiles. It is a measure of how far apart the middle portion of data spreads in value.

Interquartile Range = Upper Quartile − Lower Quartile

Problem

Find the interquartile range of eruption duration in the data set faithful.

Solution

We apply the IQR function to compute the interquartile range of eruptions.

```
> duration = faithful$eruptions     # the eruption durations
> IQR(duration)                     # apply the IQR function
[1] 2.2915
```

Answer

The interquartile range of eruption duration is 2.2915 minutes.

Practical no 5

Standard Deviation

The **standard deviation** of an observation variable is the square root of its variance.

Problem

Find the standard deviation of the eruption duration in the data set faithful.

Solution

We apply the sd function to compute the standard deviation of eruptions.

```
> duration = faithful$eruptions    # the eruption durations
> sd(duration)                     # apply the sd function
[1] 1.1414
```

Answer

The standard deviation of the eruption duration is 1.1414.

Covariance

The covariance of two variables x and y in a data set measures how the two are linearly related. A positive covariance would indicate a positive linear relationship between the variables, and a negative covariance would indicate the opposite.

he **sample covariance** is defined in terms of the sample means as:

$$s_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

Similarly, the **population covariance** is defined in terms of the population mean μ_x, μ_y as:

$$\sigma_{xy} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)$$

Problem

Find the covariance of eruption duration and waiting time in the data set faithful. Observe if there is any linear relationship between the two variables.

Solution

We apply the **cov** function to compute the covariance of eruptions and waiting.

```
> duration = faithful$eruptions   # eruption durations
> waiting = faithful$waiting      # the waiting period
> cov(duration, waiting)          # apply the cov function
[1] 13.978
```

Answer

The covariance of eruption duration and waiting time is about 14. It indicates a positive linear relationship between the two variables.

Variance

The **variance** is a numerical measure of how the data values is dispersed around the mean. In particular, the **sample variance** is defined as:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Similarly, the **population variance** is defined in terms of the population mean *μ* and population size *N*:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

Problem

Find the variance of the eruption duration in the data set faithful.

Solution

We apply the **var** function to compute the variance of eruptions.

```
> duration = faithful$eruptions    # the eruption durations
> var(duration)                    # apply the var function
[1] 1.3027
```

Answer

The variance of the eruption duration is 1.3027.

Practical no 6

Skewness:

The skewness of a data population is defined by the following formula, where μ2 and μ3 are the second and third central moments.

γ1 = μ3⁄μ3⁄22

Intuitively, the skewness is a measure of symmetry. As a rule, negative skewness indicates that the mean of the data values is less than the median, and the data distribution is left-skewed. Positive skewness would indicate that the mean of the data values is larger than the median, and the data distribution is right-skewed.

Problem

Find the skewness of eruption duration in the data set faithful.

Solution

We apply the function skewness from the e1071 package to compute the skewness coefficient of eruptions. As the package is not in the core R library, it has to be installed and loaded into the R workspace.

```
> library(e1071)                    # load e1071
> duration = faithful$eruptions     # eruption durations
> skewness(duration)                # apply the skewness function
[1] -0.41355
```

Answer

The skewness of eruption duration is -0.41355. It indicates that the eruption duration distribution is skewed towards the left.

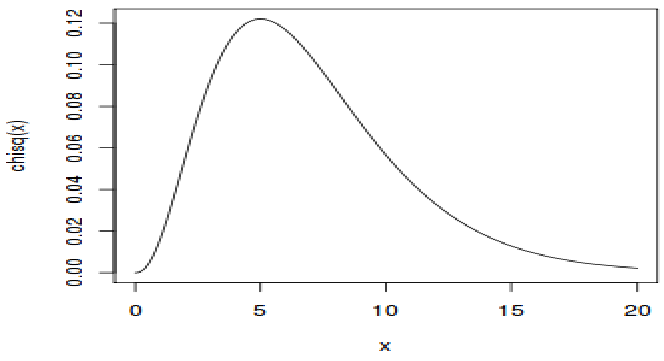
Practical 7

Chi-squared Distribution

If $X_1,X_2,\dots,X_m$ are m independent random variables having the standard normal distribution, then the following quantity follows a **Chi-Squared distribution** with *m degrees of freedom*. Its mean is *m*, and its variance is 2*m*.

$$V = X_1^2 + X_2^2 + \cdots + X_m^2 \sim \chi^2_{(m)}$$

Here is a graph of the Chi-Squared distribution 7 degrees of freedom.



Problem
Find the 95th percentile of the Chi-Squared distribution with 7 degrees of freedom.

Solution
We apply the quantile function qchisq of the Chi-Squared distribution against the decimal values 0.95.

```
> qchisq(.95, df=7)        # 7 degrees of freedom
[1] 14.067
```

#Load the library

Library("MASS")

#Create a data frame from the main data set.

car.data<-data.frame(Car93$AirBags, Car93$Type)

#Create a table with the needed variables.

car.data=table(Car93$AirBags, Car93$Type)

print(car.data)

#Perform the Chi-Square test.

print(chisq.test(car.data))

Practical no 8

Binomial Distribution

The binomial distribution is a discrete probability distribution. It describes the outcome of n independent trials in an experiment. Each trial is assumed to have only two outcomes, either success or failure. If the probability of a successful trial is p, then the probability of having x successful outcomes in an experiment of n independent trials is as follows.

$$f(x) = \binom{n}{x} p^x (1-p)^{(n-x)} \quad where\ x = 0, 1, 2, ..., n$$

Problem

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.

Solution

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

```
> dbinom(4, size=12, prob=0.2)

[1] 0.1329
```

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

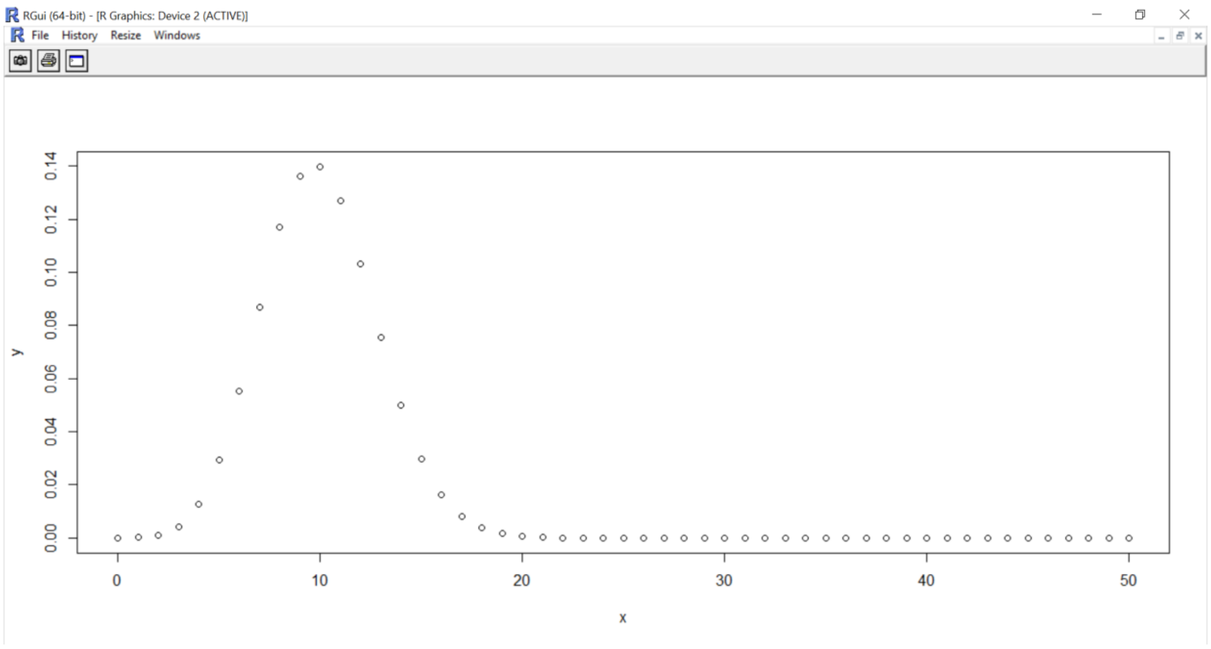
```
> dbinom(0, size=12, prob=0.2) +

+ dbinom(1, size=12, prob=0.2) +

+ dbinom(2, size=12, prob=0.2) +

+ dbinom(3, size=12, prob=0.2) +

+ dbinom(4, size=12, prob=0.2)

[1] 0.9274
```

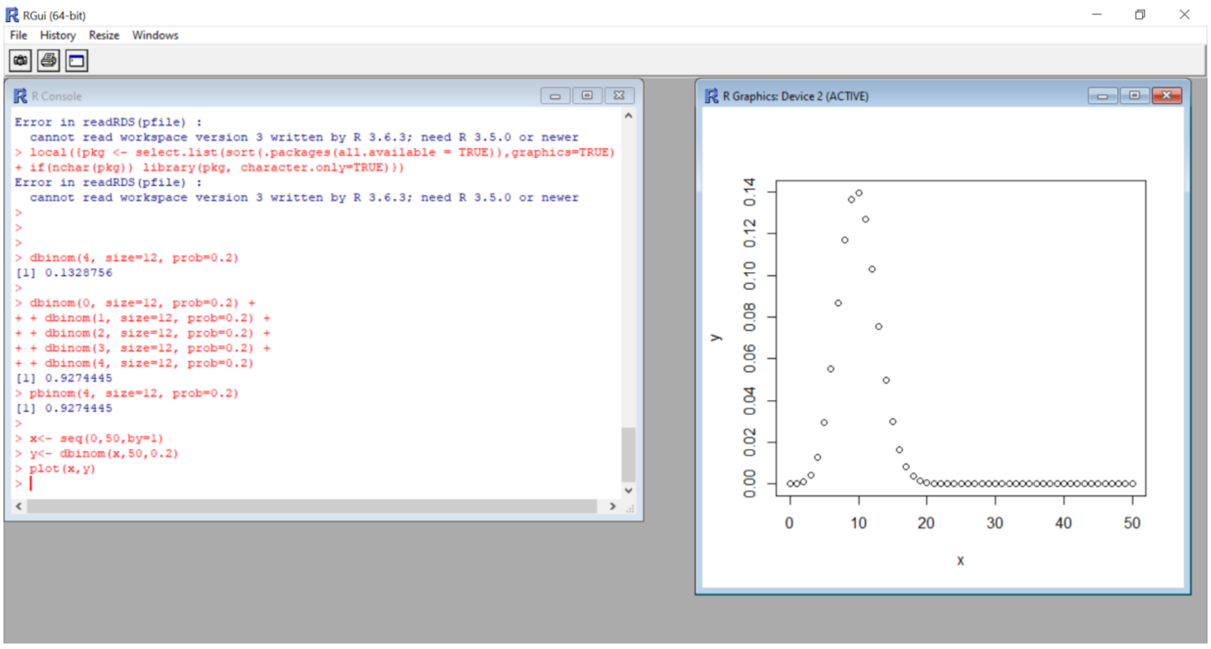
Alternatively, we can use the cumulative probability function for binomial distribution pbinom.

```
> pbinom(4, size=12, prob=0.2)
[1] 0.92744
```

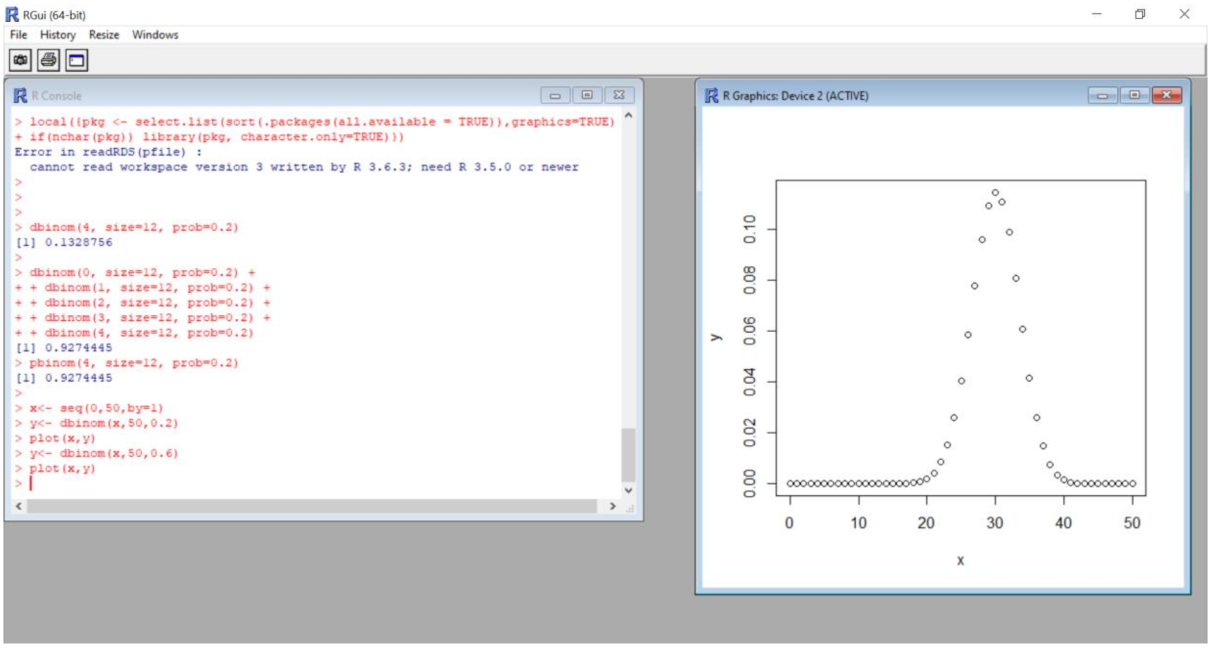
Answer

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

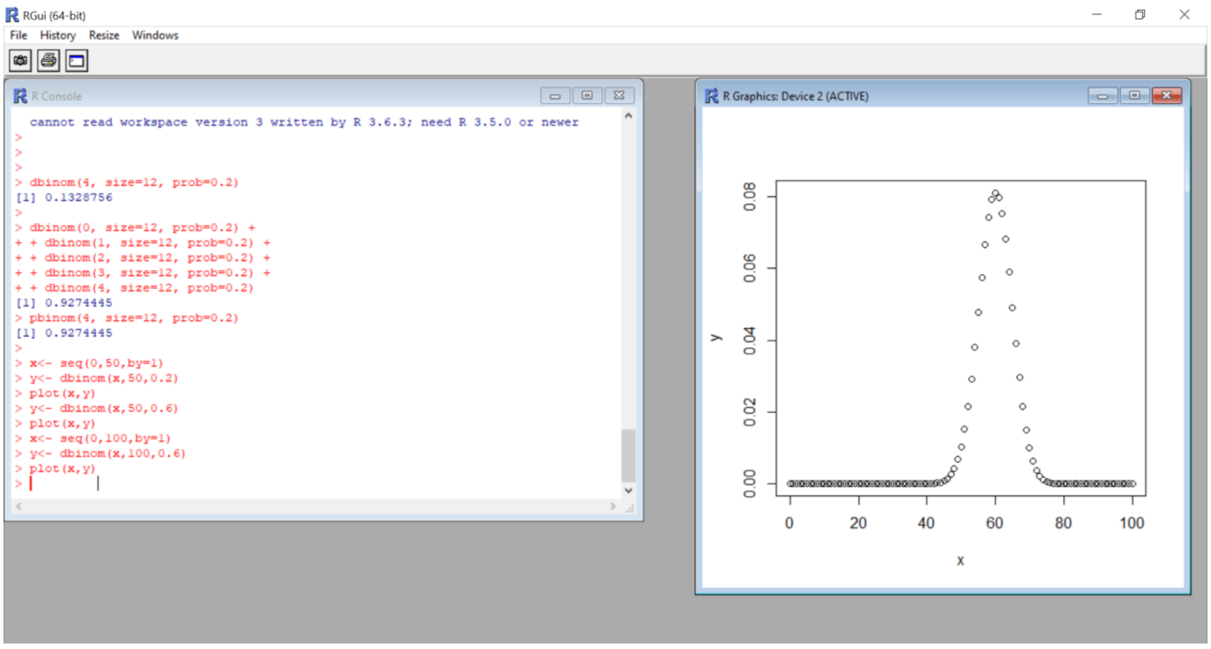




```
Error in readRDS(pfile) :
  cannot read workspace version 3 written by R 3.6.3; need R 3.5.0 or newer
> local({pkg <- select.list(sort(.packages(all.available = TRUE)),graphics=TRUE)
+ if(nchar(pkg)) library(pkg, character.only=TRUE)})
Error in readRDS(pfile) :
  cannot read workspace version 3 written by R 3.6.3; need R 3.5.0 or newer
>
>
>
> dbinom(4, size=12, prob=0.2)
[1] 0.1328756
>
> dbinom(0, size=12, prob=0.2) +
+ + dbinom(1, size=12, prob=0.2) +
+ + dbinom(2, size=12, prob=0.2) +
+ + dbinom(3, size=12, prob=0.2) +
+ + dbinom(4, size=12, prob=0.2)
[1] 0.9274445
> pbinom(4, size=12, prob=0.2)
[1] 0.9274445
>
> x<- seq(0,50,by=1)
> y<- dbinom(x,50,0.2)
> plot(x,y)
>
```



```
> local({pkg <- select.list(sort(.packages(all.available = TRUE)),graphics=TRUE)
+ if(nchar(pkg)) library(pkg, character.only=TRUE)})
Error in readRDS(pfile) :
  cannot read workspace version 3 written by R 3.6.3; need R 3.5.0 or newer
>
>
>
> dbinom(4, size=12, prob=0.2)
[1] 0.1328756
>
> dbinom(0, size=12, prob=0.2) +
+ + dbinom(1, size=12, prob=0.2) +
+ + dbinom(2, size=12, prob=0.2) +
+ + dbinom(3, size=12, prob=0.2) +
+ + dbinom(4, size=12, prob=0.2)
[1] 0.9274445
> pbinom(4, size=12, prob=0.2)
[1] 0.9274445
>
> x<- seq(0,50,by=1)
> y<- dbinom(x,50,0.2)
> plot(x,y)
> y<- dbinom(x,50,0.6)
> plot(x,y)
>
```



```
  cannot read workspace version 3 written by R 3.6.3; need R 3.5.0 or newer
>
>
>
> dbinom(4, size=12, prob=0.2)
[1] 0.1328756
>
> dbinom(0, size=12, prob=0.2) +
+ + dbinom(1, size=12, prob=0.2) +
+ + dbinom(2, size=12, prob=0.2) +
+ + dbinom(3, size=12, prob=0.2) +
+ + dbinom(4, size=12, prob=0.2)
[1] 0.9274445
> pbinom(4, size=12, prob=0.2)
[1] 0.9274445
>
> x<- seq(0,50,by=1)
> y<- dbinom(x,50,0.2)
> plot(x,y)
> y<- dbinom(x,50,0.6)
> plot(x,y)
> x<- seq(0,100,by=1)
> y<- dbinom(x,100,0.6)
> plot(x,y)
>
```

Normal Distribution

The normal distribution is defined by the following probability density function, where μ is the population mean and σ2 is the variance.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

If a random variable *X* follows the normal distribution, then we write:

$$X \sim N(\mu, \sigma^2)$$

In particular, the normal distribution with μ = 0 and σ = 1 is called the standard normal distribution, and is denoted as N(0,1). It can be graphed as follows.

PIC

The normal distribution is important because of the Central Limit Theorem, which states that the population of all possible samples of size n from a population with mean μ and variance σ2 approaches a normal distribution with mean μ and σ2⁄n when n approaches infinity.

Problem

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?

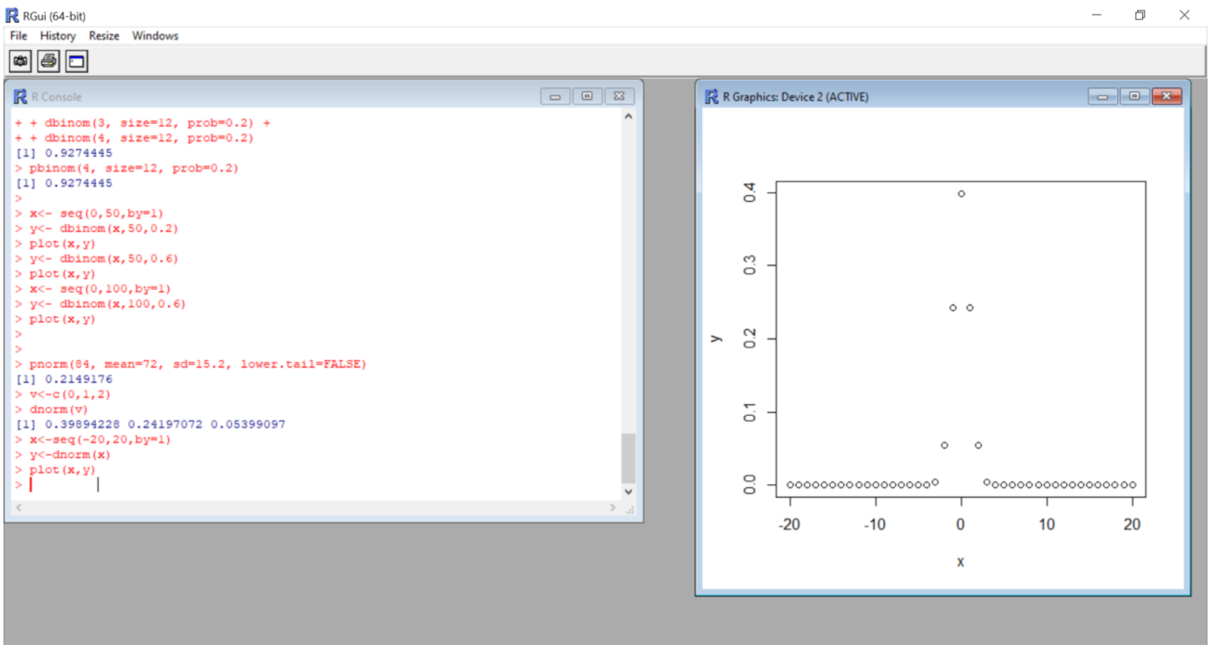
Solution

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.

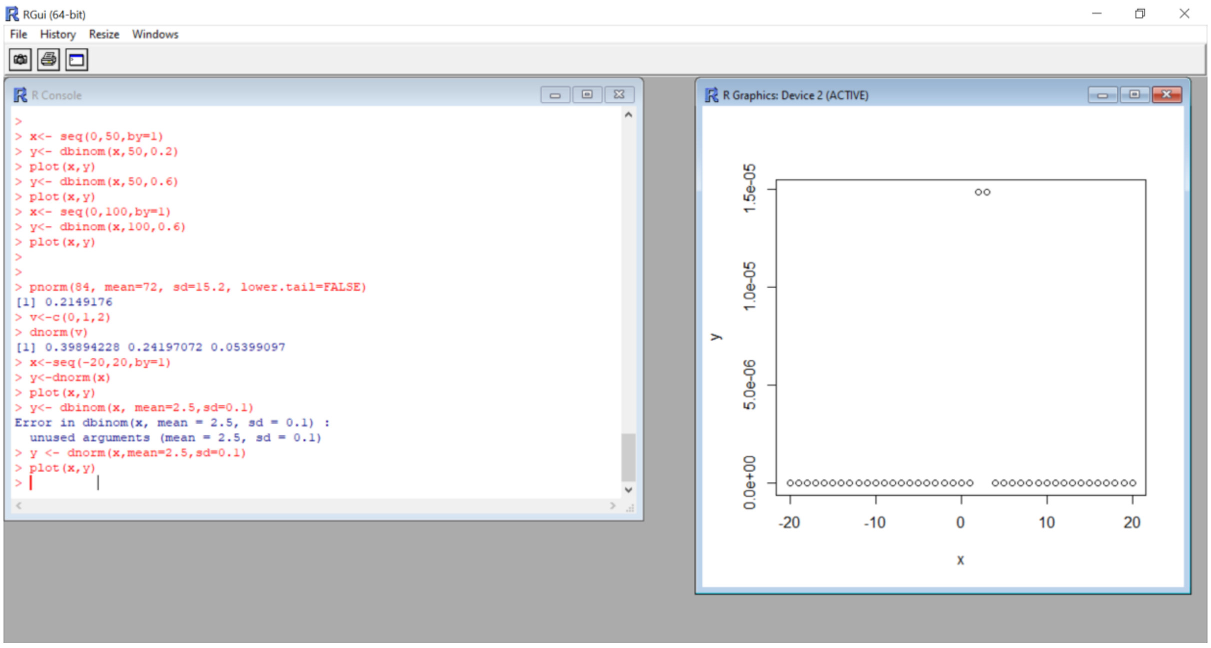
```
> pnorm(84, mean=72, sd=15.2, lower.tail=FALSE)

[1] 0.21492
```

Answer

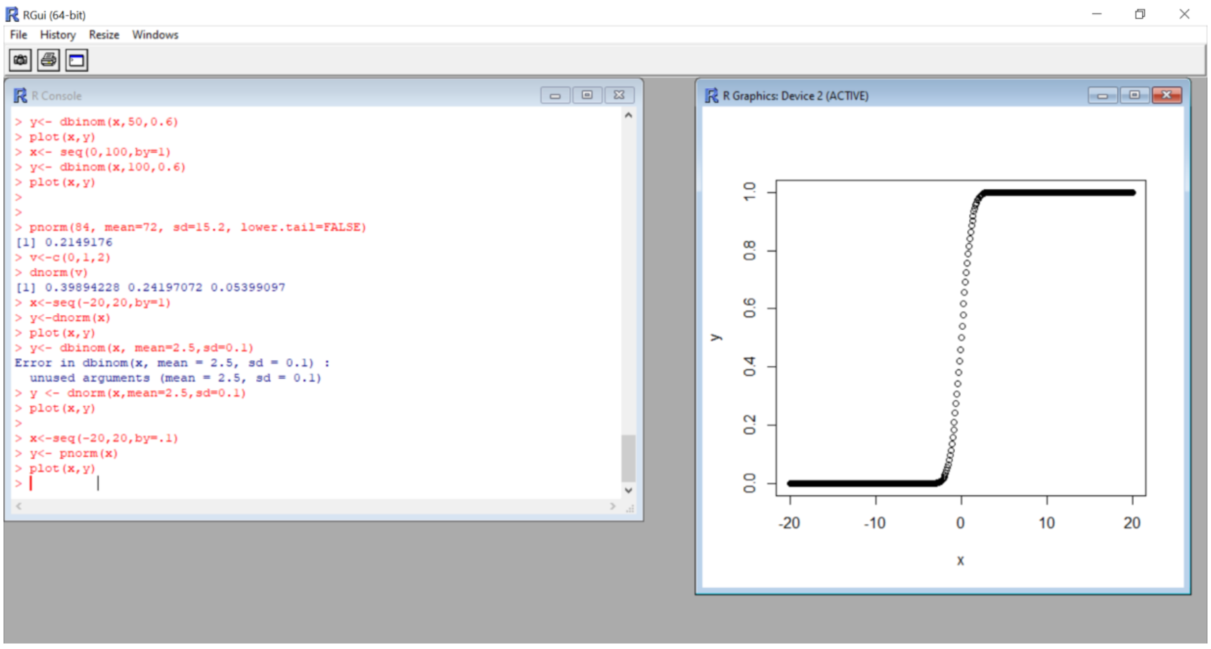
The percentage of students scoring 84 or more in the college entrance exam is 21.5%.



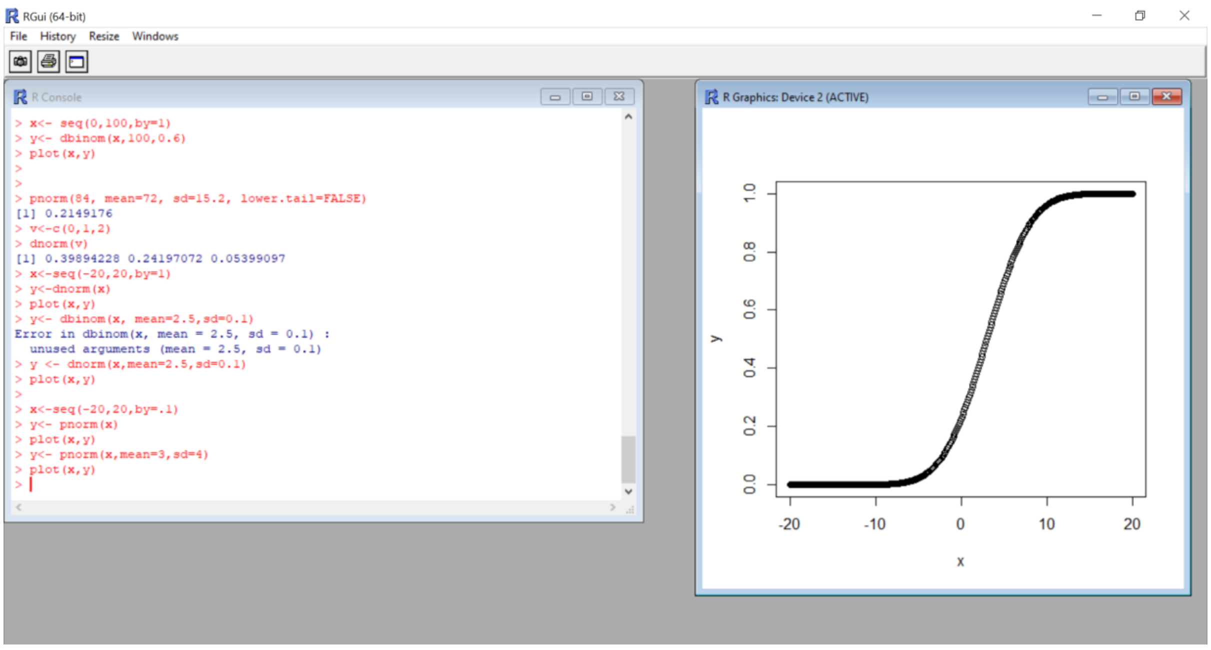
```
+ + dbinom(3, size=12, prob=0.2) +
+ + dbinom(4, size=12, prob=0.2)
[1] 0.9274445
> pbinom(4, size=12, prob=0.2)
[1] 0.9274445
>
> x<- seq(0,50,by=1)
> y<- dbinom(x,50,0.2)
> plot(x,y)
> y<- dbinom(x,50,0.6)
> plot(x,y)
> x<- seq(0,100,by=1)
> y<- dbinom(x,100,0.6)
> plot(x,y)
>
>
> pnorm(84, mean=72, sd=15.2, lower.tail=FALSE)
[1] 0.2149176
> v<-c(0,1,2)
> dnorm(v)
[1] 0.39894228 0.24197072 0.05399097
> x<-seq(-20,20,by=1)
> y<-dnorm(x)
> plot(x,y)
>
```



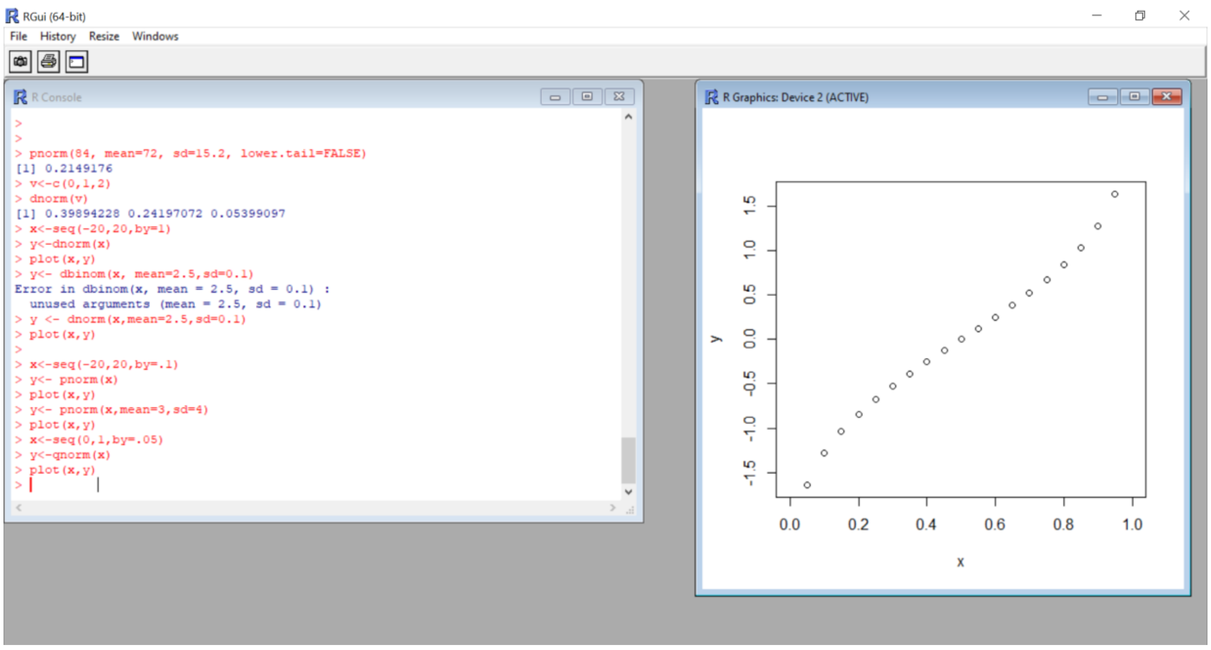
```
>
> x<- seq(0,50,by=1)
> y<- dbinom(x,50,0.2)
> plot(x,y)
> y<- dbinom(x,50,0.6)
> plot(x,y)
> x<- seq(0,100,by=1)
> y<- dbinom(x,100,0.6)
> plot(x,y)
>
>
> pnorm(84, mean=72, sd=15.2, lower.tail=FALSE)
[1] 0.2149176
> v<-c(0,1,2)
> dnorm(v)
[1] 0.39894228 0.24197072 0.05399097
> x<-seq(-20,20,by=1)
> y<-dnorm(x)
> plot(x,y)
> y<- dbinom(x, mean=2.5,sd=0.1)
Error in dbinom(x, mean = 2.5, sd = 0.1) :
  unused arguments (mean = 2.5, sd = 0.1)
> y <- dnorm(x,mean=2.5,sd=0.1)
> plot(x,y)
>
```



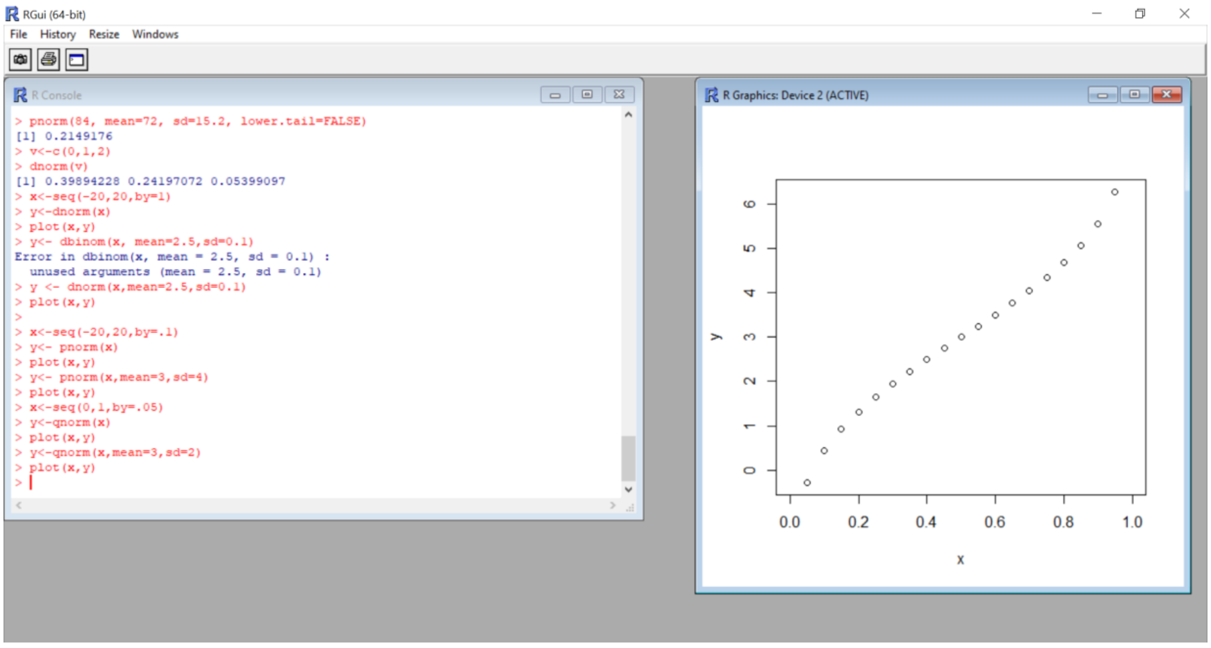
```
> y<- dbinom(x,50,0.6)
> plot(x,y)
> x<- seq(0,100,by=1)
> y<- dbinom(x,100,0.6)
> plot(x,y)
>
>
> pnorm(84, mean=72, sd=15.2, lower.tail=FALSE)
[1] 0.2149176
> v<-c(0,1,2)
> dnorm(v)
[1] 0.39894228 0.24197072 0.05399097
> x<-seq(-20,20,by=1)
> y<-dnorm(x)
> plot(x,y)
> y<- dbinom(x, mean=2.5,sd=0.1)
Error in dbinom(x, mean = 2.5, sd = 0.1) :
  unused arguments (mean = 2.5, sd = 0.1)
> y <- dnorm(x,mean=2.5,sd=0.1)
> plot(x,y)
>
> x<-seq(-20,20,by=.1)
> y<- pnorm(x)
> plot(x,y)
>
```



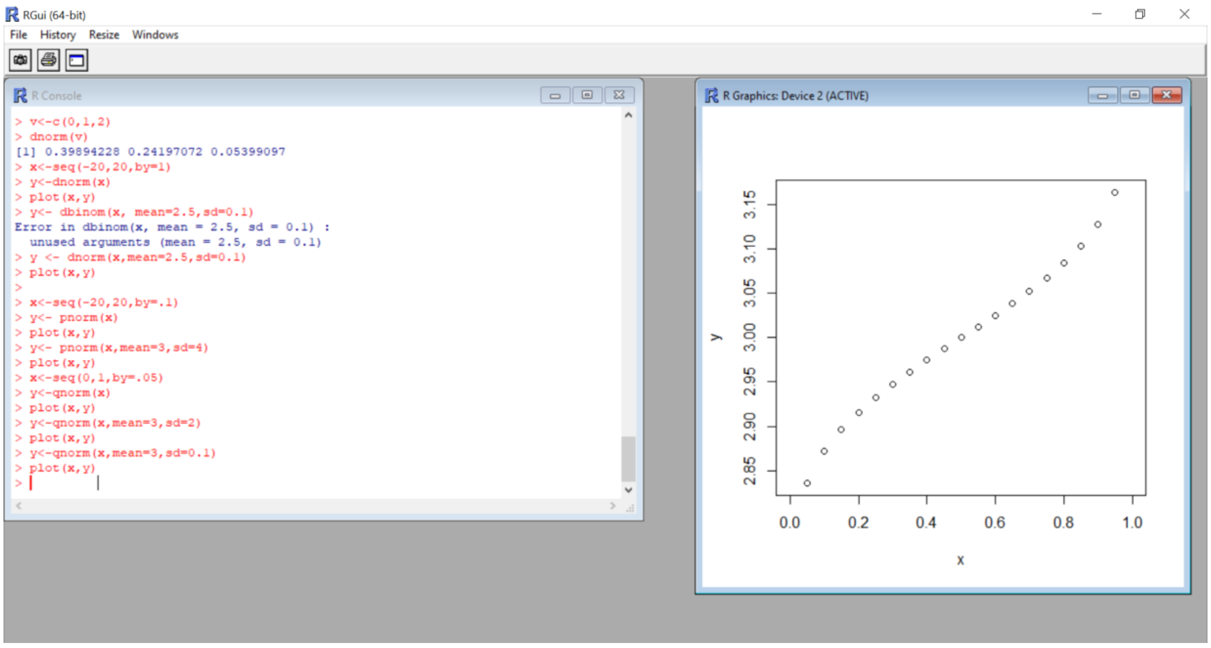
```
> x<- seq(0,100,by=1)
> y<- dbinom(x,100,0.6)
> plot(x,y)
>
>
> pnorm(84, mean=72, sd=15.2, lower.tail=FALSE)
[1] 0.2149176
> v<-c(0,1,2)
> dnorm(v)
[1] 0.39894228 0.24197072 0.05399097
> x<-seq(-20,20,by=1)
> y<-dnorm(x)
> plot(x,y)
> y<- dbinom(x, mean=2.5,sd=0.1)
Error in dbinom(x, mean = 2.5, sd = 0.1) :
  unused arguments (mean = 2.5, sd = 0.1)
> y <- dnorm(x,mean=2.5,sd=0.1)
> plot(x,y)
>
> x<-seq(-20,20,by=.1)
> y<- pnorm(x)
> plot(x,y)
> y<- pnorm(x,mean=3,sd=4)
> plot(x,y)
>
```



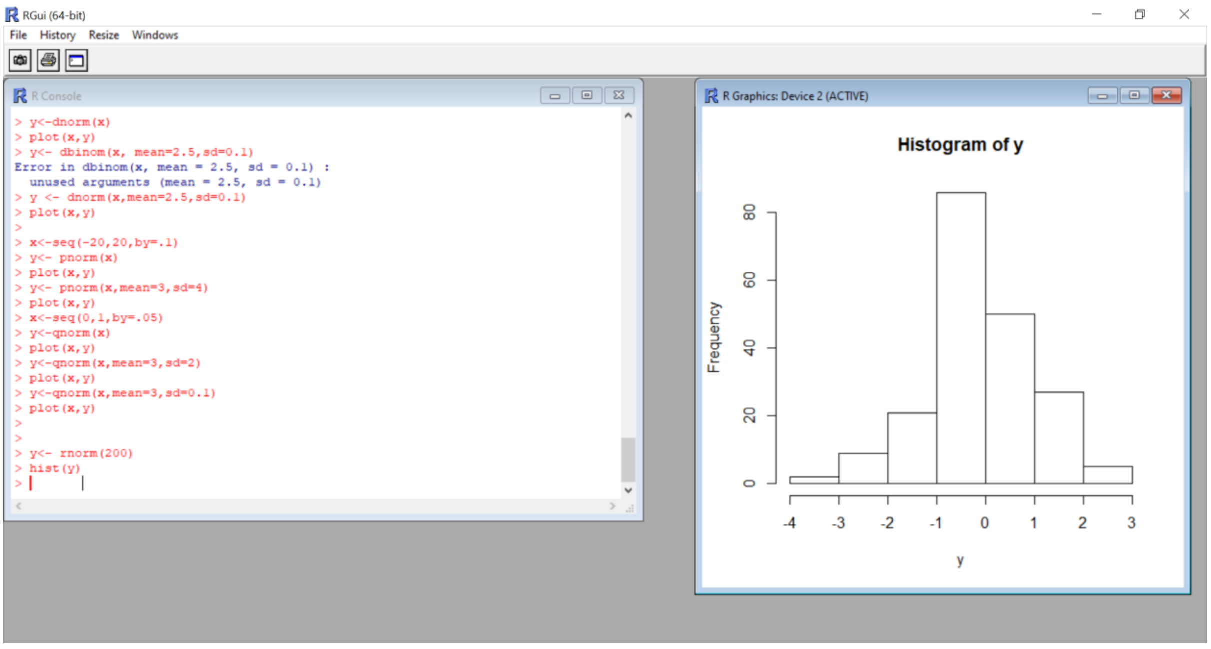
```
>
>
> pnorm(84, mean=72, sd=15.2, lower.tail=FALSE)
[1] 0.2149176
> v<-c(0,1,2)
> dnorm(v)
[1] 0.39894228 0.24197072 0.05399097
> x<-seq(-20,20,by=1)
> y<-dnorm(x)
> plot(x,y)
> y<- dbinom(x, mean=2.5,sd=0.1)
Error in dbinom(x, mean = 2.5, sd = 0.1) :
  unused arguments (mean = 2.5, sd = 0.1)
> y <- dnorm(x,mean=2.5,sd=0.1)
> plot(x,y)
>
> x<-seq(-20,20,by=.1)
> y<- pnorm(x)
> plot(x,y)
> y<- pnorm(x,mean=3,sd=4)
> plot(x,y)
> x<-seq(0,1,by=.05)
> y<-qnorm(x)
> plot(x,y)
>
```



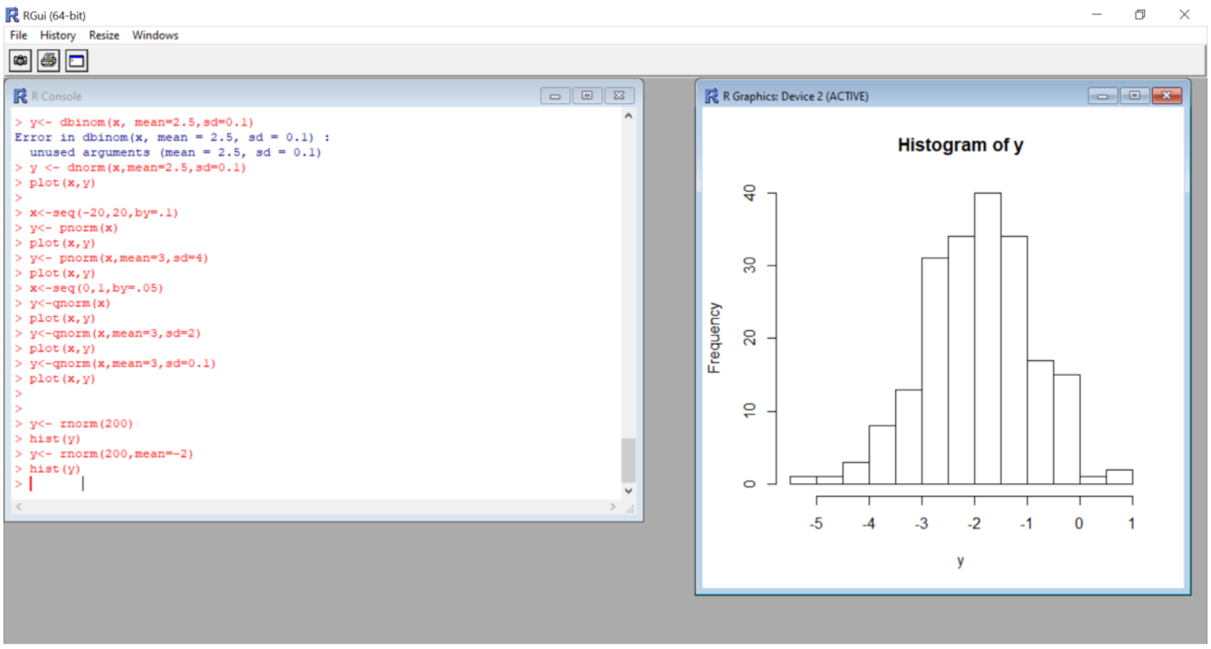
```
> pnorm(84, mean=72, sd=15.2, lower.tail=FALSE)
[1] 0.2149176
> v<-c(0,1,2)
> dnorm(v)
[1] 0.39894228 0.24197072 0.05399097
> x<-seq(-20,20,by=1)
> y<-dnorm(x)
> plot(x,y)
> y<- dbinom(x, mean=2.5,sd=0.1)
Error in dbinom(x, mean = 2.5, sd = 0.1) :
  unused arguments (mean = 2.5, sd = 0.1)
> y <- dnorm(x,mean=2.5,sd=0.1)
> plot(x,y)
>
> x<-seq(-20,20,by=.1)
> y<- pnorm(x)
> plot(x,y)
> y<- pnorm(x,mean=3,sd=4)
> plot(x,y)
> x<-seq(0,1,by=.05)
> y<-qnorm(x)
> plot(x,y)
> y<-qnorm(x,mean=3,sd=2)
> plot(x,y)
>
```



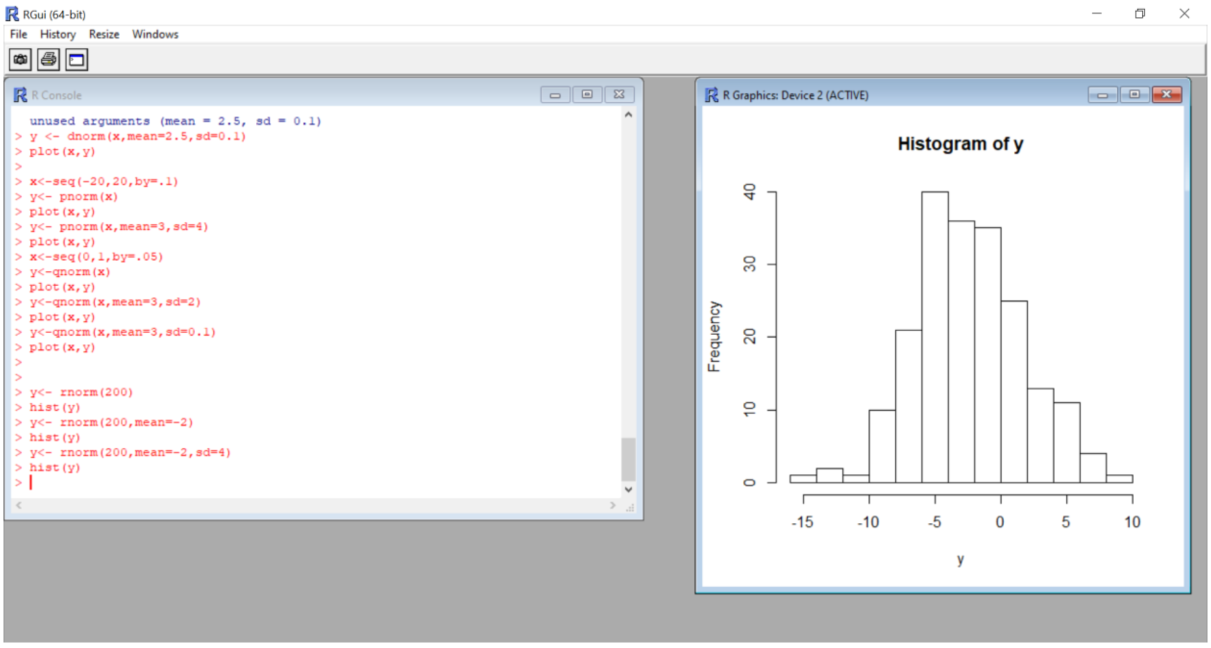
```
> v<-c(0,1,2)
> dnorm(v)
[1] 0.39894228 0.24197072 0.05399097
> x<-seq(-20,20,by=1)
> y<-dnorm(x)
> plot(x,y)
> y<- dbinom(x, mean=2.5,sd=0.1)
Error in dbinom(x, mean = 2.5, sd = 0.1) :
  unused arguments (mean = 2.5, sd = 0.1)
> y <- dnorm(x,mean=2.5,sd=0.1)
> plot(x,y)
>
> x<-seq(-20,20,by=.1)
> y<- pnorm(x)
> plot(x,y)
> y<- pnorm(x,mean=3,sd=4)
> plot(x,y)
> x<-seq(0,1,by=.05)
> y<-qnorm(x)
> plot(x,y)
> y<-qnorm(x,mean=3,sd=2)
> plot(x,y)
> y<-qnorm(x,mean=3,sd=0.1)
> plot(x,y)
>
```



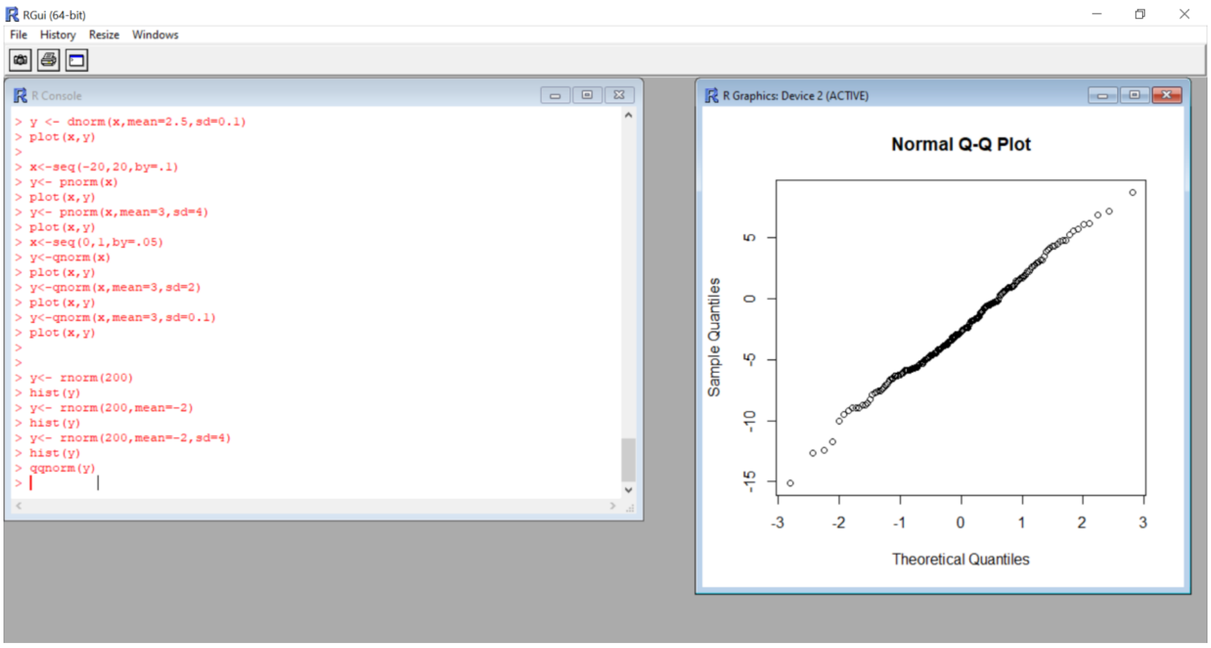
```
> y<-dnorm(x)
> plot(x,y)
> y<- dbinom(x, mean=2.5,sd=0.1)
Error in dbinom(x, mean = 2.5, sd = 0.1) :
  unused arguments (mean = 2.5, sd = 0.1)
> y <- dnorm(x,mean=2.5,sd=0.1)
> plot(x,y)
>
> x<-seq(-20,20,by=.1)
> y<- pnorm(x)
> plot(x,y)
> y<- pnorm(x,mean=3,sd=4)
> plot(x,y)
> x<-seq(0,1,by=.05)
> y<-qnorm(x)
> plot(x,y)
> y<-qnorm(x,mean=3,sd=2)
> plot(x,y)
> y<-qnorm(x,mean=3,sd=0.1)
> plot(x,y)
>
>
> y<- rnorm(200)
> hist(y)
>
```



```
> y<- dbinom(x, mean=2.5,sd=0.1)
Error in dbinom(x, mean = 2.5, sd = 0.1) :
  unused arguments (mean = 2.5, sd = 0.1)
> y <- dnorm(x,mean=2.5,sd=0.1)
> plot(x,y)
>
> x<-seq(-20,20,by=.1)
> y<- pnorm(x)
> plot(x,y)
> y<- pnorm(x,mean=3,sd=4)
> plot(x,y)
> x<-seq(0,1,by=.05)
> y<-qnorm(x)
> plot(x,y)
> y<-qnorm(x,mean=3,sd=2)
> plot(x,y)
> y<-qnorm(x,mean=3,sd=0.1)
> plot(x,y)
>
>
> y<- rnorm(200)
> hist(y)
> y<- rnorm(200,mean=-2)
> hist(y)
>
```



```
  unused arguments (mean = 2.5, sd = 0.1)
> y <- dnorm(x,mean=2.5,sd=0.1)
> plot(x,y)
>
> x<-seq(-20,20,by=.1)
> y<- pnorm(x)
> plot(x,y)
> y<- pnorm(x,mean=3,sd=4)
> plot(x,y)
> x<-seq(0,1,by=.05)
> y<-qnorm(x)
> plot(x,y)
> y<-qnorm(x,mean=3,sd=2)
> plot(x,y)
> y<-qnorm(x,mean=3,sd=0.1)
> plot(x,y)
>
>
> y<- rnorm(200)
> hist(y)
> y<- rnorm(200,mean=-2)
> hist(y)
> y<- rnorm(200,mean=-2,sd=4)
> hist(y)
>
```



```
> y <- dnorm(x,mean=2.5,sd=0.1)
> plot(x,y)
>
> x<-seq(-20,20,by=.1)
> y<- pnorm(x)
> plot(x,y)
> y<- pnorm(x,mean=3,sd=4)
> plot(x,y)
> x<-seq(0,1,by=.05)
> y<-qnorm(x)
> plot(x,y)
> y<-qnorm(x,mean=3,sd=2)
> plot(x,y)
> y<-qnorm(x,mean=3,sd=0.1)
> plot(x,y)
>
>
> y<- rnorm(200)
> hist(y)
> y<- rnorm(200,mean=-2)
> hist(y)
> y<- rnorm(200,mean=-2,sd=4)
> hist(y)
> qqnorm(y)
>
```

```
x<- seq(0,50,by=1)

> y<- dbinom(x,50,0.2)

> plot(x,y)

> y<- dbinom(x,50,0.6)

> plot(x,y)

> x<- seq(0,100,by=1)
> y<- dbinom(x,100,0.6)
> plot(x,y)
```

Plot of Normal distribution:

d: returns height of pdf

```
> v<-c(0,1,2)
> dnorm(v)
[1] 0.39894228 0.24197072 0.05399097
> x<-seq(-20,20,by=1)
> y<-dnorm(x)
> plot(x,y)
> y <- dnorm(x,mean=2.5,sd=0.1)
> plot(x,y)
```

p:returns cdf

```
> x<-seq(-20,20,by=.1)
> y<- pnorm(x)
> plot(x,y)
> y<- pnorm(x,mean=3,sd=4)
> plot(x,y)
```

q: returns inverse of cdf

```
> x<-seq(0,1,by=.05)
> y<-qnorm(x)
> plot(x,y)
> y<-qnorm(x,mean=3,sd=2)
> plot(x,y)
> y<-qnorm(x,mean=3,sd=0.1)
> plot(x,y)
```

r: returns random generated numbers.

```
> y<- rnorm(200)
> hist(y)
> y<- rnorm(200,mean=-2)
> hist(y)
> y<- rnorm(200,mean=-2,sd=4)
> hist(y)
> qqnorm(y)
```