

Basic R Programming Part-2

1. Loops

It allows you to execute a statement or group of statements or block of code multiple times. In general, statements are executed sequentially. It provide various control structures that allow for more complicated execution paths.

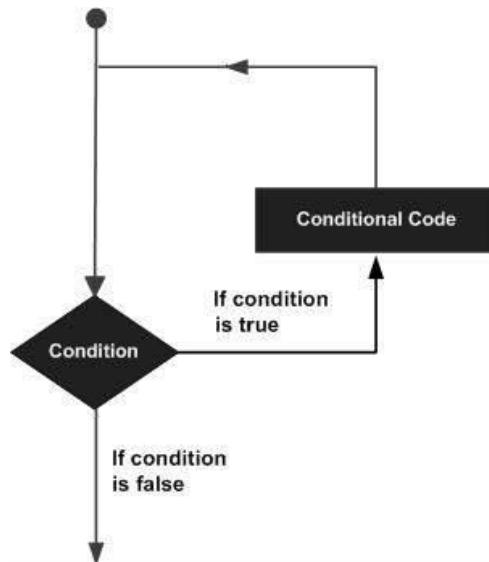


Figure 1: Loop Structure Flow Chart

- a. **Repeat:** Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

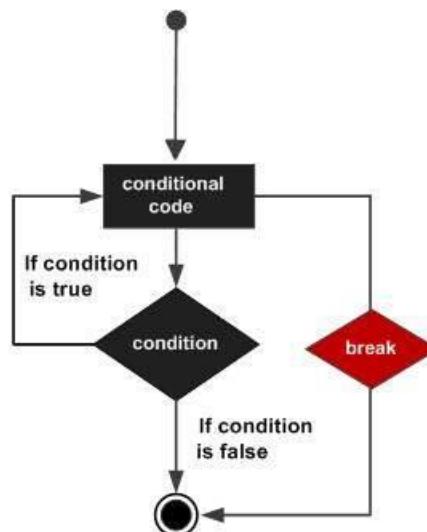


Figure 2: Repeat Loop Flow Chart

Use:

```

repeat {
  commands
  if(condition) {
    break
  }
}
  
```

```

    }
}

hadoop-spark - VMware Workstation 12 Player (Non-commercial use only)
File Edit Code View Plots Session Build Debug Profile Tools Help
Project: (None) Enviro
File R Da
File in table form and create a data frame from it, with case corre to lines and varia to fields in the file.
R Script 28-07-2017
8:10 (Top Level) ~/
Console > source('~/myR/Programs/practice_repeat.R')
0 1 2 3 4 5 6 7 8 9 10
>

```

Figure 3: Repeat Loop Program

- b. **While:** Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

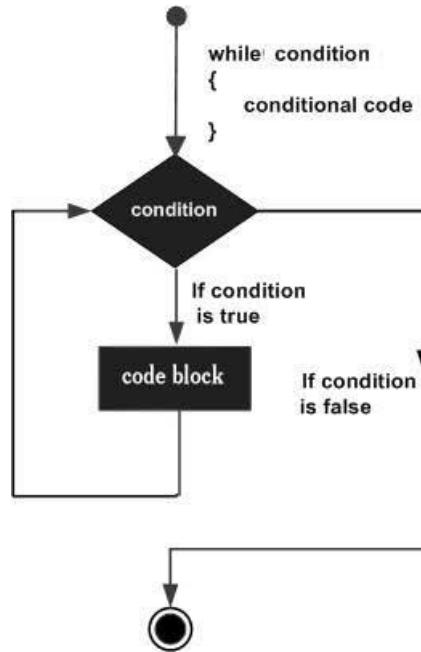


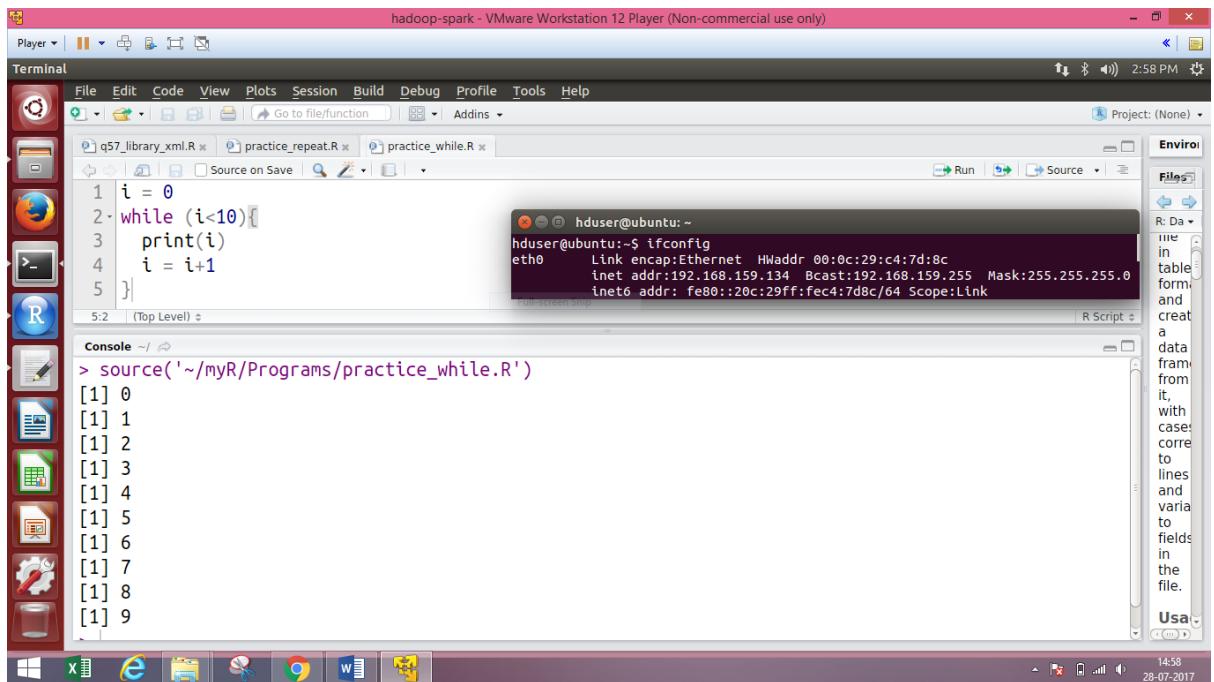
Figure 4: While Loop Flow Chart

Use:

```

while (test_expression) {
  statement
}

```



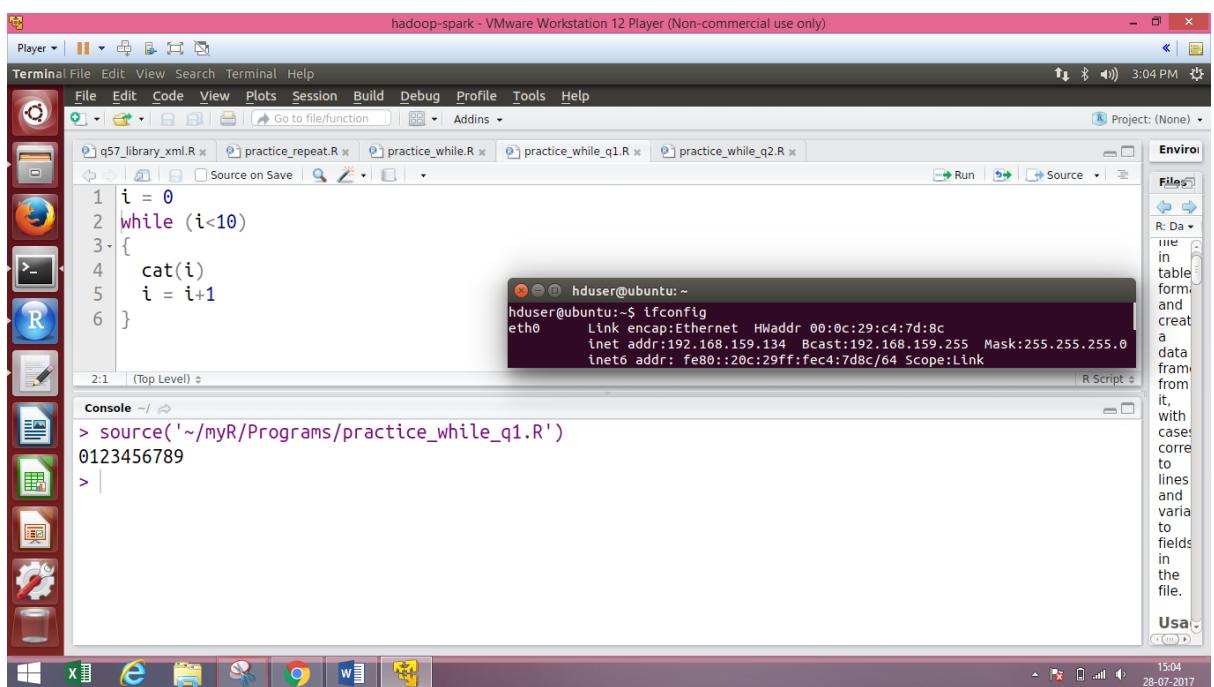
A screenshot of the RStudio interface running on a Windows host (VMware Workstation 12 Player). The main window shows an R script named 'practice_while.R' with the following code:

```
i = 0
while (i<10){
  print(i)
  i = i+1
}
```

The console output shows the numbers 0 through 9 printed sequentially. In the background, a terminal window displays the system's network configuration:

```
hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link
```

Figure 5: While Loop Program



A screenshot of the RStudio interface running on a Windows host (VMware Workstation 12 Player). The main window shows an R script named 'practice_while_q1.R' with the following code:

```
i = 0
while (i<10)
{
  cat(i)
  i = i+1
}
```

The console output shows the numbers 0 through 9 printed sequentially. In the background, a terminal window displays the system's network configuration:

```
hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link
```

Figure 6: q1 - While Loop Cat Program

The screenshot shows an RStudio interface running on a Windows host (VMware Workstation). The code editor displays the following R script:

```

1 i = 0
2 while (i<10)
3 {
4   cat(i, " ")
5   i = i+1
6 }

```

The console window shows the execution of the script and its output:

```

> source('~/myR/Programs/practice_while_q1.R')
0123456789
> source('~/myR/Programs/practice_while_q2.R')
0 1 2 3 4 5 6 7 8 9
>

```

A terminal window in the background shows the command `ifconfig` being run on an Ubuntu host, displaying network interface information.

Figure 7: q2 - While Loop Program

- c. **For:** Like while loop, it is managed by control variable except that it tests the condition at the end of the loop body.

Use:

```

for (value in vector) {
  statements
}

```

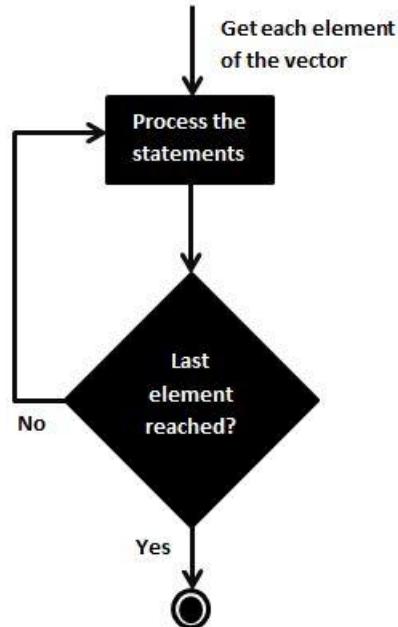


Figure 8: For Loop Flow Chart

```

for(i in c(1:10)){
  if(i%%2==0){
    cat(i, " ")
  }
}
cat("\n")
a = 1:10
for(i in a){
  if(i%%2==0){
    cat(i, " ")
  }
}

```

R Console output:

```

> source('~/myR/Programs/practice_for.R')
2 4 6 8 10

```

Figure 9: For Loop Program

```

cat("for 5 to 7\n")
for(i in 5:7){#1
  cat(i, " ")
}#1 for
cat("\nfor: divisible by 3 inn 1 to 10\n")
for(i in 1:10){#2
  if(i%%3 == 0){#2.1
    cat(i, " ")
  }#2.1 if
}#2 for
cat("\n for loop after break\n")
for(i in 1:10){ #3
  cat(i, " ")
  if(i == 5){ #3.1
    break
  }#3.1 if
} #for|

```

R Console output:

```

> source('~/myR/Programs/for.R')
for 5 to 7
5 6 7
for: divisible by 3 inn 1 to 10
3 6 9
for loop after break
1 2 3 4 5

```

Figure 10: For Loop Program and Break

d. Nested Loops:

Similarly to nested if, loops can be created in nested fashion. It allows bidirectional structure and patterns to be created.

```

i = 1
rows = 3
while(i<=rows)
{
  j = 1

```

```
while(j<=i)
{
  if(j<i)
    cat(i,"*")
  else
    cat(i)
  j = j + 1
}
cat("\n")
i = i + 1
}
if(i%2 != 0)
{
  i = rows - 1
}
while(i >= 1)
{
  j = 1
  while(j <= i)
  {
    if(j<i)
      cat(i,"*")
    else
      cat(i)
    j = j + 1
  }
  cat("\n")
  i = i -1
}
i = 1
rows = 5
repeat
{
  k = i
  j = 1
  if(i>rows)
  {
    break
  }
  repeat
  {
    if(j>i)
    {
      break
    }
    cat(k," ")
    k = k + rows-j;
  }
}
```

The screenshot shows the RStudio interface with the following code in the editor:

```

j = j+1
}
cat("\n")
i = i+1
}

```

The code in the main editor is:

```

41 if(i>rows)
42 {
43   break
44 }
45 repeat
46 {
47   if(j>i)
48   {
49     break
50   }
51   cat(k, " ")
52   k = k + rows-j;
53   j = j+1
54 }
55 cat("\n")
56 i = i+1
57 }

```

The console output shows the generated sequence:

```

1
2 *2
3 *3 *3
4 *4 *4 *4
3 *3 *3
2 *2
1
1
2 6
3 7 10
4 8 11 13
5 9 12 14 15

```

Figure 11: Nested Loop with break

e. Loop Control Statements

- i. **Break:** Terminates the **loop** statement and transfers execution to the statement immediately following the loop. See above example.
- ii. **Next:** The **next** statement simulates the behaviour of R switch.

The screenshot shows the RStudio interface with the following code in the editor:

```

1 x <- 1:5
2
3 for (val in x) {
4   if (val == 3){
5     next
6   }
7   print(val)
8 }

```

The console output shows the printed values:

```

[1] 1
[1] 2
[1] 4
[1] 5
>

```

Figure 12: Next in Loops

2. Functions

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. In R, a function is an object so the R interpreter is able to pass control to the function. The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

a. cat() and print() function:

print() function can print data from any object but it takes only one argument.
 cat() function can take multiple arguments separated by ',' but it cannot print complex object types.

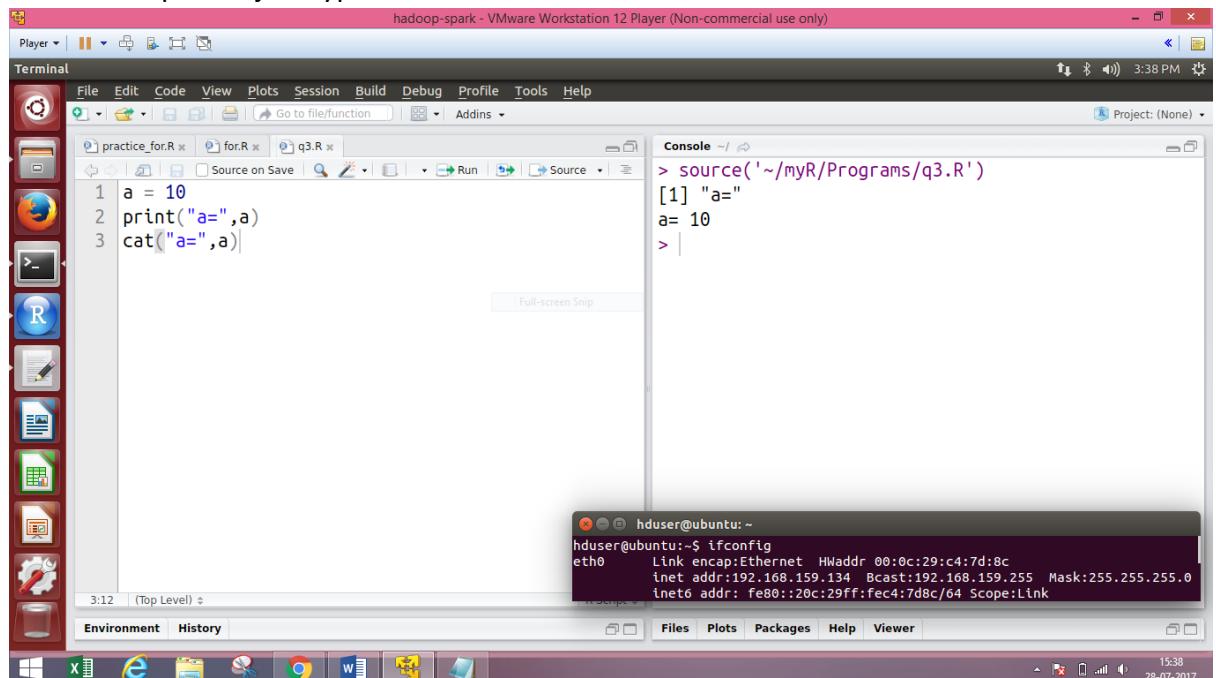


Figure 13: q3 - cat() and print() functions

3. User Defined Functions

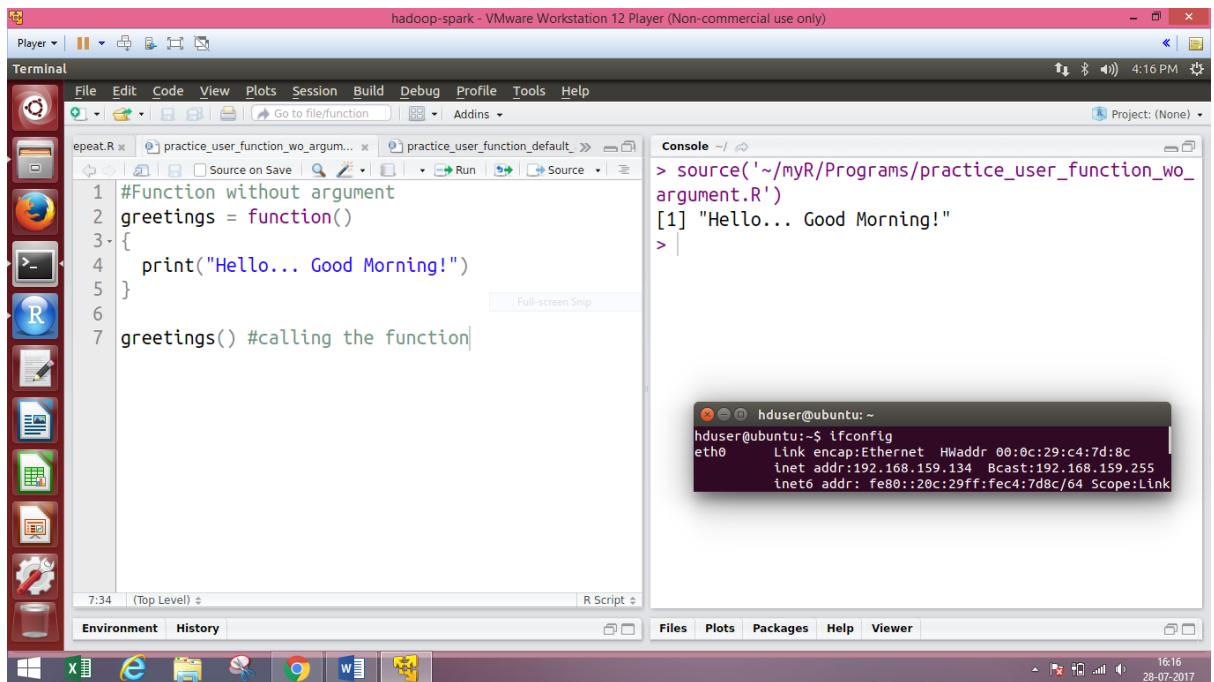
User-defined functions in R, They are specific to what a user wants and once created they can be used like the built-in functions. Below is an example of how a function is created and used.

Use:

```

function_name <- function(arg_1, arg_2, ...) {
  Function body
}
  
```

a. Function without Parameter



The screenshot shows the RStudio interface running on a Windows host. The left pane displays an R script named 'repeat.R' containing the following code:

```

1 #Function without argument
2 greetings = function()
3 {
4   print("Hello... Good Morning!")
5 }
6
7 greetings() #calling the function

```

The right pane shows the R console output:

```

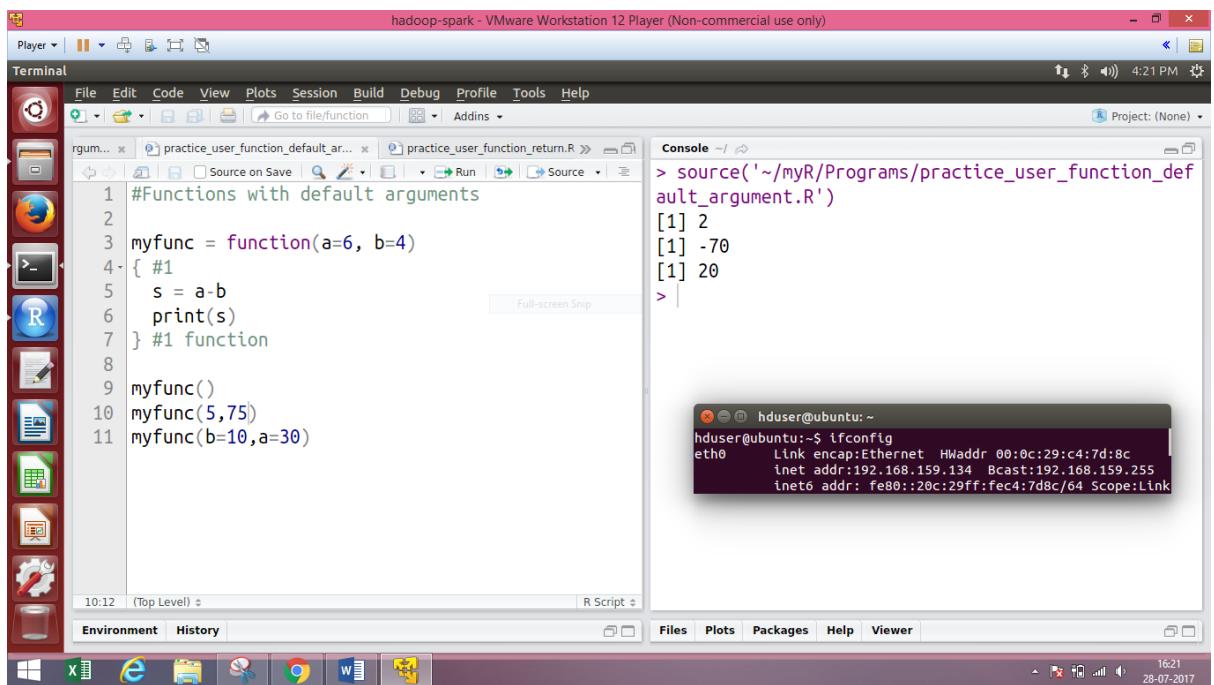
> source('~/myR/Programs/practice_user_function_wo_argument.R')
[1] "Hello... Good Morning!"
>

```

Below the RStudio window, a terminal window is open, showing the command 'ifconfig' and its output for the eth0 interface.

Figure 14: Calling a Function without an Argument

b. Function with Parameter



The screenshot shows the RStudio interface running on a Windows host. The left pane displays an R script named 'rgum.R' containing the following code:

```

1 #Functions with default arguments
2
3 myfunc = function(a=6, b=4)
4 {
5   #1
6   s = a-b
7   print(s)
8 }
9
10 myfunc()
11 myfunc(5,75)
12 myfunc(b=10,a=30)

```

The right pane shows the R console output:

```

> source('~/myR/Programs/practice_user_function_default_argument.R')
[1] 2
[1] -70
[1] 20
>

```

Below the RStudio window, a terminal window is open, showing the command 'ifconfig' and its output for the eth0 interface.

Figure 15: Calling a Function with Argument Values (by position and by name)

i. Fixed Parameters

Call function with defined variable only. Call can be made with default positions or variable name can be assigned.

ii. Variable Parameters Function call

1. '...' variable

The screenshot shows the RStudio interface running on a Windows host (VMware Workstation 12 Player). The left pane displays an R script named 'practice_function_variable_no_arg.R' containing the following code:

```

1 #Functions with default arguments
2
3 myfunc = function(a=6, b=4, ...)
4 { #1
5   s = a+b...
6   print(s)
7 } #1 function
8
9 myfunc()
10 myfunc(5,75)
11 myfunc(5,75,20)
12 myfunc(b=10,a=30)

```

The right pane shows the R console output:

```

> source('~/active-rstudio-document')
[1] 10
[1] 80
[1] 100
[1] 40
>

```

Below the console, a terminal window shows the command 'ifconfig' output:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link

```

Figure 16: variable function parameter

2. Function Overloading

Function overloading is inbuilt available in R programming by taking variable as run time variable. Since its function vary according to type of variable passed as argument. Still Function Overloading can be implements manually by creating functions or using operators. Infix operator can be used for that purpose.

The screenshot shows the RStudio interface running on a Windows host (VMware Workstation 12 Player). The left pane displays an R script named 'practice_func_overloading.R' containing the following code:

```

1 myfunc = function()
2 {
3   print("\nhello")
4 }
5 myfunc = function(a)
6 { #1
7   cat("\nhello ",a)
8 } #1 function
9 myfunc = function(a = "person a", b = "person b")
10 { #1
11   cat("\nhello ",a," and ",b)
12 } #1 function
13
14 myfunc()
15 myfunc("Gazal")
16 myfunc("Gazal","Patel")

```

The right pane shows the R console output:

```

> source('~/myR/Programs/practice_func_overloading.R')
hello person a and b
hello Gazal and b
hello Gazal and b
>

```

Below the console, a terminal window shows the command 'ifconfig' output:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link

```

Figure 17: Function Overloading

c. Return from function

i. return() function

The screenshot shows the RStudio interface. On the left is a tool palette with icons for file operations, browser, terminal, and R. The main area has two tabs: 'practice_user_function_return.R' and 'practice_user_function_return_diff.R'. The code editor contains the following R script:

```

1 mysum = function(a,b)
2 {
3   s = a+b
4   return(s)
5 }
6 print(mysum(10,20))

```

The 'Console' tab shows the output of running the script:

```

> source('~/myR/Programs/practice_user_function_return.R')
[1] 30
>

```

Below the console, a terminal window shows the command 'ifconfig' being run on an Ubuntu system:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fe:c47d8c/64 Scope:Link

```

Figure 18: Function with return() function

ii. Auto return

By default, user defined function returns last assigned and used variable.

The screenshot shows the RStudio interface. The code editor contains the following R script:

```

1 # Functions returning values in a different way
2
3 mysum = function(a,b)
4 {
5   s = a+b #s" will be automatically returned
6 }
7 print(mysum(10,20))
8

```

The 'Console' tab shows the output of running the script:

```

> source('~/myR/Programs/practice_user_function_return_diff.R')
[1] 30
>

```

Below the console, a terminal window shows the command 'ifconfig' being run on an Ubuntu system:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fe:c47d8c/64 Scope:Link

```

Figure 19: Default function return

4. Escape Sequences

Backslash is used to start an escape sequence inside character constants. Escaping a character not in the following table is an error.

Single quotes need to be escaped by backslash in single-quoted strings, and double quotes in double-quoted strings. Single and double quotes delimit character constants. They can be used interchangeably but double quotes are preferred (and character constants are printed

using double quotes), so single quotes are normally only used to delimit character constants containing double quotes.

\n	Newline
\r	carriage return
\t	Tab
\b	Backspace
\a	alert (bell)
\f	form feed
\v	vertical tab
\\\	backslash \
\'	ASCII apostrophe '
\"	ASCII quotation mark "
\`	ASCII grave accent (backtick) `
\nnn	character with given octal code (1, 2 or 3 digits)
\xnn	character with given hex code (1 or 2 hex digits)
\unnnn	Unicode character with given code (1–4 hex digits)
\Unnnnnnnn	Unicode character with given code (1–8 hex digits)

5. Working with Strings

a. `paste()` function

Concatenate the strings. If a value is specified for collapse, the values in the result are then concatenated into a single string, with the elements being separated by the value of collapse. 'sep' argument is the separator. In example it is between the strings a,b for concatenation.

Use:

```
paste(..., sep = " ", collapse = NULL)
paste0(..., collapse = NULL)
```

The screenshot shows the RStudio interface running on a Windows host. The left pane displays an R script named 'q10_q11_string_paste.R' with the following code:

```

9 a = "hello how are you"
10 b = "I am fine"
11
12 q10 = paste(a,b)
13 print(q10)
14
15 q11 = paste(a,b,sep="")
16 print(q11)
17
18 v1 = 1:5
19 v2 = 10:15
20 task = paste(v1,v2,sep=",",collapse = "#")
21 print(task)
22
23 task = paste(a,b,sep=",",collapse = "#")
24 print(task)

```

The right pane shows the R console output:

```

> source('~/myR/Programs/q10_q11_string_paste.R')
[1] "hello how are you I am fine"
[1] "hello how are youI am fine"
[1] "&10#2811#3&12#4&13#5&14#1&15"
[1] "hello how are you&I am fine"
>

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link

```

The bottom status bar indicates the date and time: 28-07-2017 17:12.

Figure 20: q10 and q11 - string paste

b. nchar() function

Find total characters present in string including space. nchar() function takes a character vector as an argument and returns a vector whose elements contain the sizes of the corresponding elements of x. nzchar() function is a fast way to find out if elements of a character vector are non-empty strings.

Use:

```
nchar(x, type = "chars", allowNA = FALSE)
nzchar(x)
```

The screenshot shows the RStudio interface running on a Windows host. The left pane displays an R script named 'q12_nchar.R' with the following code:

```

1 #Find total character in string
2 #nchar() will also count the space
3
4 a = "Hi Hello"
5 print(nchar(a))
6

```

The right pane shows the R console output:

```

> source('~/myR/Programs/q12_nchar.R')
[1] 8
>

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link

```

The bottom status bar indicates the date and time: 28-07-2017 17:13.

Figure 21: q12 - nchar() function

c. toupper() function

Translate characters in character vectors, in upper-case characters. Non-alphabetic characters are left unchanged.

Use:

```
toupper(x)
```

d. tolower() function

Translate characters in character vectors, in lower case of given string. Non-alphabetic characters are left unchanged.

Use:

```
tolower(x)
```

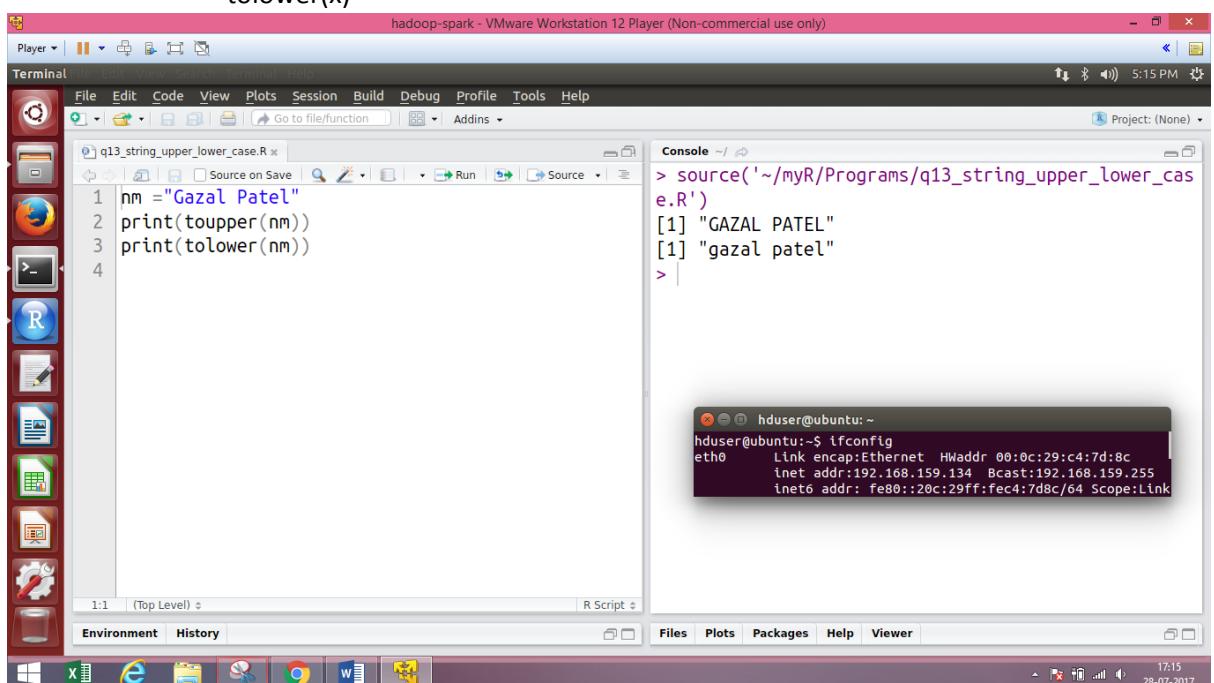


Figure 22: q13 - upper and lower case string

e. sub() and gsub() function

sub and gsub perform replacement of the first and all matches respectively. It matches pattern in x (text) and replaces it with replacement.

Use:

```
sub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
     fixed = FALSE, useBytes = FALSE)
```

```
gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
      fixed = FALSE, useBytes = FALSE)
```

The screenshot shows an RStudio interface running on a Windows host. The left pane displays an R script named 'practice_string_gsub_count.R' with the following code:

```

1 str <- "blabla 23 mai 2000 blabla 18 mai 2004"
2 p <- "0"
3 s2 <- gsub(p,"",str)
4 numof <- nchar(str) - nchar(s2)
5 cat("number of times '",p,"' is repeated in '",s2,"'",sep="")
6
7 str1 <- "How are you"
8 p1 <- " "
9 s3 <- gsub(p1,"",str1)
10 numof1 <- nchar(str1) - nchar(s3)
11 cat("\nnumber of times '",p1,"' is repeated in '",s3,"'",sep="")

```

The right pane shows the R console output:

```

> source('~/myR/Programs/practice_string_gsub_count.R')
number of times ' 0 ' is repeated in ' blabla 23 mai 2000 blabla 18 mai 2004 ' is: 5
number of times ' ' is repeated in ' How are you ' is: 2
>

```

Below the RStudio window, a terminal window is open showing network configuration:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link

```

Figure 23: q14 - sub() and gsub() functions

f. strsplit() function

Split the elements of a character vector 'x' into substrings according to the matches to substring 'split' within them.

Use:

```
strsplit(x, split, fixed = FALSE, perl = FALSE, useBytes = FALSE)
```

g. unlist() function

Given a list structure x, unlist simplifies it to produce a vector which contains all the atomic components which occur in x.

Use:

```
unlist(x, recursive = TRUE, use.names = TRUE)
```

```

1 vect = c("Gazal",5,T)
2 vect2 = c("Kavita",21)
3 ss = strsplit(c(vect,vect2), "$"); print(ss); cat("\n class of ss: ",class(ss))
4 unss = unlist(ss); cat("\n unss: |",unss," \n class of unss: ", class(unss))

```

Console output:

```

[1] "21"

class of ss:  list
unss:  Gazal 5 TRUE Kavita 21
class of unss:  character
> |

```

Figure 24: `strsplit()` and `unlist()` functions

h. `invisible()` function

Return a (temporarily) invisible copy of an object. It cannot be used individually in program. This function can be useful when it is desired to have functions return values which can be assigned, but which do not print when they are not assigned.

This is a primitive function.

Use:

`invisible(x)`

```

3 # the output on the screen.
4 myfunc = function(x)
5 { #1
6   ix = invisible()
7   if(x == 1){ #1.1
8     x = x + 1
9     ix = invisible(x)
10  sum = x + 10
11 } #1.1 if
12 else{ #1.2
13   x = x + 1
14   sum = x - 10
15 }
16 return(ix)
17 } #1.2 else
18 b = myfunc(1); print(b)
19 b = myfunc(10); print(b)

```

Console output:

```

[1] 2
NULL
> |

Details
This function can be useful when it is desired to have functions return values which can be assigned, but which do not print when they are not assigned.

See Also
withVisible, return.function.

```

Figure 25: `invisible()` function

6. Working with Data Frame

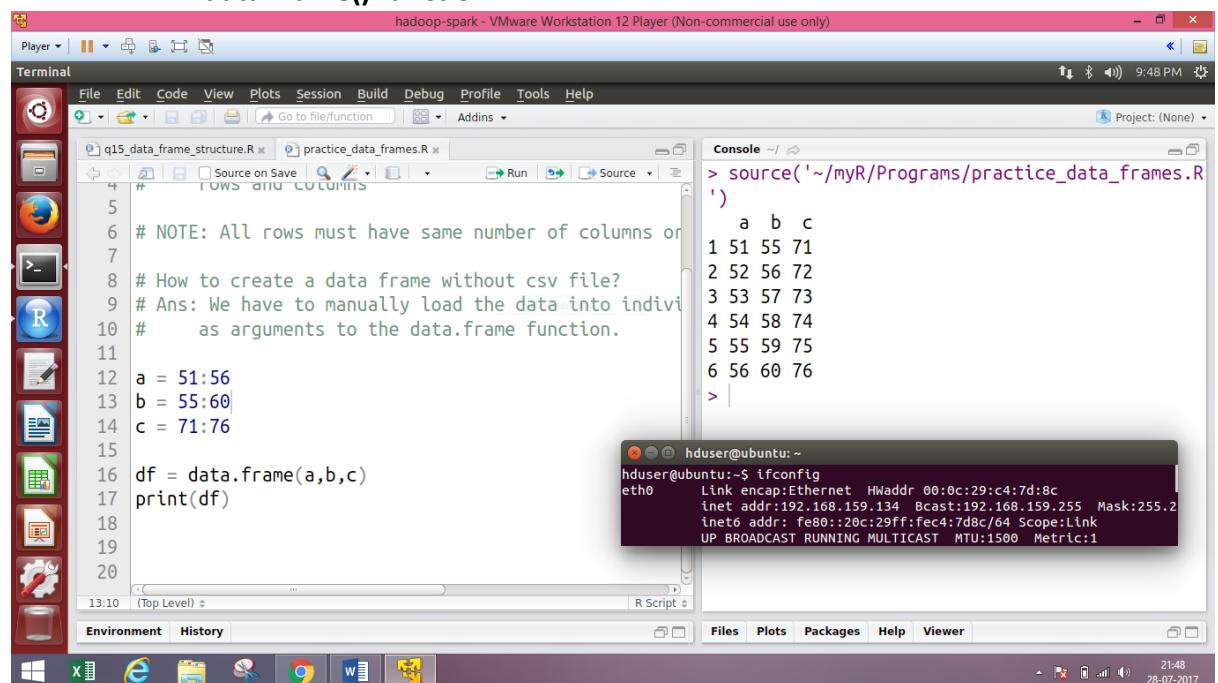
Data frames can be seen as a collection of vector arranged in rows and columns. A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

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.

a. Create Data Frame

i. `data.frame()` function



The screenshot shows the RStudio interface with a terminal window. The terminal window displays R code and its output. The code creates three vectors (a, b, c) and then creates a data frame (df) from them. The output shows the data frame df with columns a, b, and c containing the respective vector values.

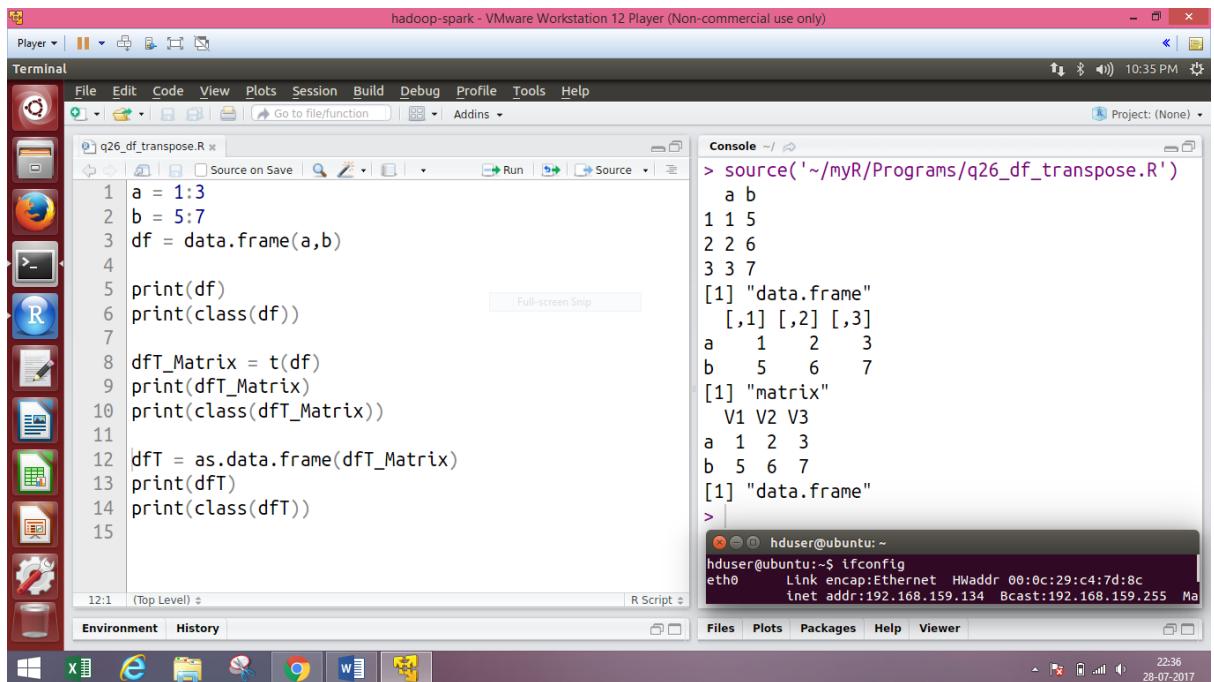
```

> source('~/myR/Programs/practice_data_frames.R')
')
      a   b   c
1 51 55 71
2 52 56 72
3 53 57 73
4 54 58 74
5 55 59 75
6 56 60 76
>
hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

```

Figure 26: `data.frame()` function to create and `print()` function to view Data Frame

ii. Create Data Frame from Matrix



The screenshot shows the RStudio interface. On the left is a tool palette with icons for file operations, browser, terminal, and R. The main area has two panes: a script editor containing R code and a console pane.

```

hadoop-spark - VMware Workstation 12 Player (Non-commercial use only)
File Edit Code View Plots Session Build Debug Profile Tools Help
q26_df_transpose.R x
Source on Save Go to file/function Addins
1 a = 1:3
2 b = 5:7
3 df = data.frame(a,b)
4
5 print(df)
6 print(class(df))
7
8 dfT_Matrix = t(df)
9 print(dfT_Matrix)
10 print(class(dfT_Matrix))
11
12 dft = as.data.frame(dfT_Matrix)
13 print(dft)
14 print(class(dft))
15
12:1 (Top Level) + R Script c
Environment History
Console ~/ ~
> source('~/myR/Programs/q26_df_transpose.R')
a b
1 1 5
2 2 6
3 3 7
[1] "data.frame"
[,1] [,2] [,3]
a 1 2 3
b 5 6 7
[1] "matrix"
V1 V2 V3
a 1 2 3
b 5 6 7
[1] "data.frame"
>
hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          ...

```

The status bar at the bottom indicates the date and time: 28-07-2017 22:36.

Figure 27: Matrix to Data Frame

iii. See [Subset](#)

a. View Data Frame

To view Data Frame `print()` is used. See [Data Frame](#)

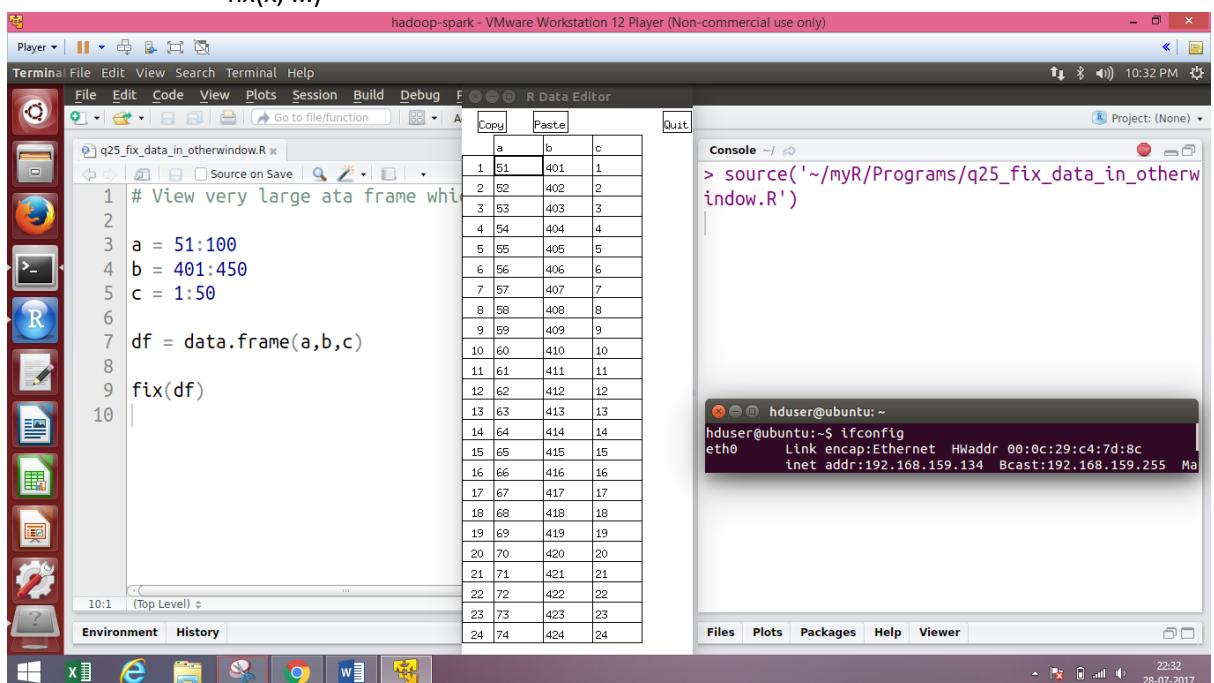
Use:

```
print(x, ...)
```

To view Data Frame in excel like sheet in external window, use `fix()` function.

Use:

```
fix(x, ...)
```



The screenshot shows the RStudio interface. The script editor contains R code to create a large data frame and then view it using the `fix()` function. The data frame is displayed in a grid format.

```

hadoop-spark - VMware Workstation 12 Player (Non-commercial use only)
File Edit View Search Terminal Help
q25_fix_data_in_otherwindow.R x
Source on Save Go to file/function R Data Editor
1 # View very large ata frame whi
2
3 a = 51:100
4 b = 401:450
5 c = 1:50
6
7 df = data.frame(a,b,c)
8
9 fix(df)
10
10:1 (Top Level) + R Data Editor
a b c
1 51 401 1
2 52 402 2
3 53 403 3
4 54 404 4
5 55 405 5
6 56 406 6
7 57 407 7
8 58 408 8
9 59 409 9
10 60 410 10
11 61 411 11
12 62 412 12
13 63 413 13
14 64 414 14
15 65 415 15
16 66 416 16
17 67 417 17
18 68 418 18
19 69 419 19
20 70 420 20
21 71 421 21
22 72 422 22
23 73 423 23
24 74 424 24

```

The status bar at the bottom indicates the date and time: 28-07-2017 22:32.

Figure 28: q25 - View Data Frame using `fix()` function

b. Check structure

`str()` function is used to check structure of Data Frames.

Use:

```
str(object, ...)
```

```
str(object, max.level = NA, vec.len = strO$vec.len, digits.d =
strO$digits.d, nchar.max = 128, give.attr = TRUE, give.head = TRUE,
give.length = give.head, width =getOption("width"), nest.lev = 0,
indent.str = paste(rep.int(" ", max(0, nest.lev + 1)), collapse =
".." ), comp.str = "$ ", no.list = FALSE, envir = baseenv(), strict.width =
strO$strict.width, formatNum = strO$formatNum, list.len = 99, ...)
```

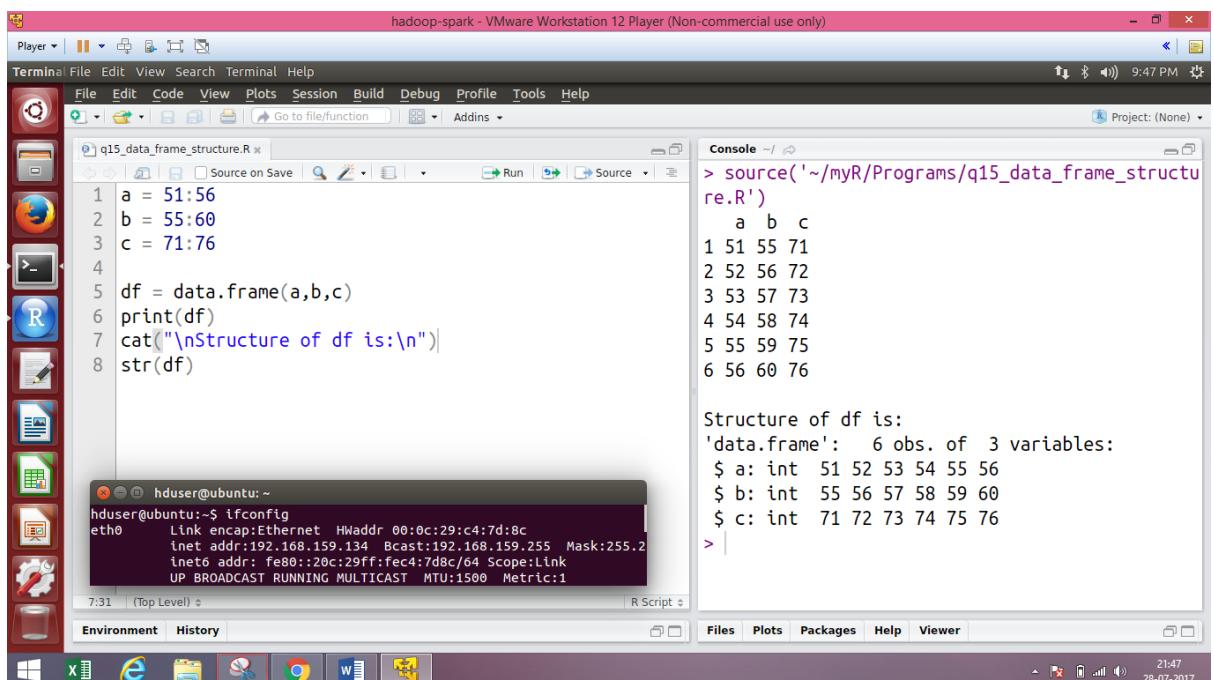


Figure 29: q15 - `str()` function to check Data Frame Structure

c. summary() of structure

Summary is a generic function used to produce result statistical summaries of the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument. (See [Function Overloading](#))

Use:

```
summary(object, ...)
```

```
## Default S3 method:  
summary(object, ..., digits = max(3,getOption("digits")-3))  
## S3 method for class 'data.frame'
```

```
summary(object, maxsum = 7,  
       digits = max(3,getOption("digits")-3), ...)
```

```
## S3 method for class 'factor'
```

```
summary(object, maxsum = 100, ...)

## S3 method for class 'matrix'
summary(object, ...)
```

The screenshot shows the RStudio interface with a terminal window open. The terminal window displays the output of the `summary()` function for a data frame named `df`:

```
a b c
1 51 55 71
2 52 56 72
3 53 57 73
4 54 58 74
5 55 59 75
6 56 60 76

Summary of df is:
Min. :51.00 1st Qu.:52.25 Median :53.50
Mean :53.50 3rd Qu.:54.75 Max. :56.00
00 Min. :55.00 1st Qu.:56.25 Median :57.50
7.50 Mean :57.50 3rd Qu.:58.75 Max. :60.00
n :73.50 Mean :73.50 3rd Qu.:74.75 Max. :76.00
. :76.00
```

Below the terminal window, there is a small window showing the output of the `ifconfig` command:

```
hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
```

Figure 30: q16 - summary() of Data Frame

d. Extract data from Data Frame

i. Get Row

Class of result set is data.frame.

The screenshot shows the RStudio interface with a terminal window open. The terminal window displays the output of extracting specific rows from a data frame:

```
> source('~/myR/Programs/q18_df_extract_rows.R')
)
a b c
1 51 55 71
a b c
1 51 55 71
2 52 56 72
> |
```

Below the terminal window, there is a small window showing the output of the `ifconfig` command:

```
hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
```

Figure 31: q18 - Extract Row from Data Frame

ii. Get Column

Class of result set is vector.

The screenshot shows the RStudio interface with a terminal window. The code in the script pane is:

```

1 # Access sign the data from the data frame
2 # Ans: The data frame which was created(df)
3 #   is composed of 3 vectors(a,b,c) which are
4 # 3 columns of the data frame
5
6 a = 51:56
7 b = 55:60
8 c = 71:76
9
10 df = data.frame(a,b,c)
11 print(df$b)
12 cat("\n Data from first two columns : \n",df$a,df$b)

```

The console output shows the results of the code execution:

```

> source('~/myR/Programs/q17_data_frame_access_data.R')
[1] 55 56 57 58 59 60

Data from first two columns :
51 52 53 54 55 56 55 56 57 58 59 60
>

```

A terminal window at the bottom shows the command `ifconfig` output:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

```

Figure 32: q17 - Access Column from Data Frame

iv. Get Specific Data

Get data from exact row and column.

The screenshot shows the RStudio interface with a terminal window. The code in the script pane is:

```

1 a = 51:56
2 b = 55:60
3 c = 71:76
4
5 df = data.frame(a,b,c)
6 row1and5andcol3 = df[c(1,5),3]
7
8 print(df)
9 cat("\n (R,C) = (1,3) and (5,3): \n",row1and5andcol3)

```

The console output shows the results of the code execution:

```

> source('~/myR/Programs/q20_df_specific_row_col.R')
   a   b   c
1 51 55 71
2 52 56 72
3 53 57 73
4 54 58 74
5 55 59 75
6 56 60 76

(R,C) = (1,3) and (5,3):
71 75
>

```

A terminal window at the bottom shows the command `ifconfig` output:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

```

Figure 33: q20 - Access 3 column value data of 1st and 5th row

v. Head of the Data

View head data from vector, matrix, table, and data frame or function. By default it shows 6 rows but it can be set explicitly in function call.

Use:

`head(x, n = 6L, ...)`

The screenshot shows the RStudio interface with the following details:

- Title Bar:** hadoop-spark - VMware Workstation 12 Player (Non-commercial use only)
- Toolbar:** File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help
- File Explorer:** Shows files q26_df_transpose.R and q27_df_head_data.R.
- Code Editor:** Displays R code for generating a data frame and printing its head.
- Console:** Shows the output of the head() function, displaying rows 15 through 21 of the data frame.
- Terminal:** Shows the command ifconfig and its output for the eth0 interface.
- Status Bar:** Project: (None), 10:42 PM, 28-07-2017.

```

1 a = 25:45
2 b = 50:70
3
4 df = data.frame(a,b)
5 print(df)
6
7 h_df =head(df)
8 print(h_df)
9
10 h_df =head(df, n=2)
11 print(h_df)

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:255.255.255.0
          ...

```

Figure 34: q27 - head() function for Data Frame

vi. Tail of the Data

Returns the last parts of a vector, matrix, table, data frame or function.

Use:

`tail(x, n = 6L, ...)`

The screenshot shows the RStudio interface with the following details:

- Title Bar:** hadoop-spark - VMware Workstation 12 Player (Non-commercial use only)
- Toolbar:** File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help
- File Explorer:** Shows file q28_df_tail_data.R.
- Code Editor:** Displays R code for generating a data frame and printing its tail.
- Console:** Shows the output of the tail() function, displaying rows 16 through 21 of the data frame.
- Terminal:** Shows the command ifconfig and its output for the eth0 interface.
- Status Bar:** Project: (None), 10:46 PM, 28-07-2017.

```

1 # view last rows of data frame
2
3 a = 25:45
4 b = 50:70
5
6 df = data.frame(a,b)
7 print(df)
8
9 t_df =tail(df)
10 print(t_df)
11
12 t_df =tail(df, n=2)
13 print(t_df)

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:255.255.255.0
          ...

```

Figure 35: q28 - tail() function for Data Frame

vii. Data without NA (Omit NA)

`na.omit()` returns the object with incomplete cases removed.

Use:

`na.omit(object, ...)`

```

hadoop-spark - VMware Workstation 12 Player (Non-commercial use only)
Player Terminal File Edit Code View Plots Session Build Debug Profile Tools Help
q29_df.omit.NA.R x Addins Project: (None) 10:48 PM
File Edit Code View Plots Session Build Debug Profile Tools Help Go to file/function Addins
Source on Save Run Source Full-screen Snip
1 # omit missing values in a data fram(NA)
2 # The entire row which contains NA will be ignored
3
4 v1 = 1:5
5 v1[3] = NA
6 v2 = 6:10
7 v3 = 11:15
8 v3[4] = NA
9
10 mydf = data.frame(v1,v2,v3)
11 print(mydf)
12 print(na.omit(mydf))

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:255.255.255.0
          ...
12:21 (Top Level) R Script Environment History Files Plots Packages Help Viewer 22:48 28-07-2017

```

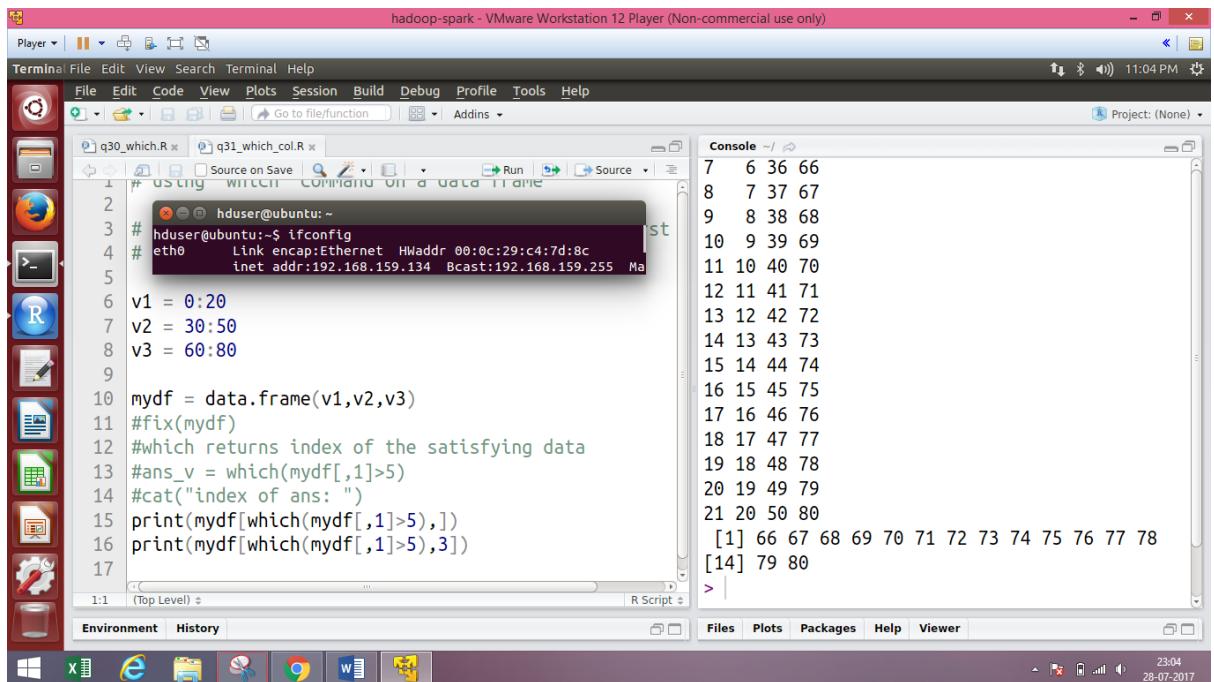
Figure 36: q29 - na.omit() for Data Frame

viii. Which Command

Get specific data which satisfies given condition.

```

hadoop-spark - VMware Workstation 12 Player (Non-commercial use only)
Player Terminal File Edit View Search Terminal Help
q30_which.R x Addins Project: (None) 11:00 PM
File Edit Code View Plots Session Build Debug Profile Tools Help Go to file/function Addins
Source on Save Run Source Full-screen Snip
1 # BUT FROM JUST LIST COLUMN WHICH IS greater than
2 v1 = 0:20
3 v2 = 30:50
4 v3 = 60:80
5 v4 = 90:110
6 v5 = 120:140
7 v6 = 150:170
8
9 mydf = data.frame(v1,v2,v3,v4,v5,v6)
10 #which returns index of the satisfying data
11 ans_v = which(mydf[,1]>5)
12 cat("\nindex of ans: ", ans_v, "\n", "Ans is: ")
13 for(i in ans_v){
14   cat(mydf[i,1], " ")
15 }
16 ans_v = which(mydf[,1]>5 & mydf[,2]<40)
17 cat("\nindex of ans: ", ans_v, "\n")
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
310
320
330
340
350
360
370
380
390
400
410
420
430
440
450
460
470
480
490
500
510
520
530
540
550
560
570
580
590
600
610
620
630
640
650
660
670
680
690
700
710
720
730
740
750
760
770
780
790
800
810
820
830
840
850
860
870
880
890
900
910
920
930
940
950
960
970
980
990
1000
1010
1020
1030
1040
1050
1060
1070
1080
1090
1100
1110
1120
1130
1140
1150
1160
1170
1180
1190
1200
1210
1220
1230
1240
1250
1260
1270
1280
1290
1300
1310
1320
1330
1340
1350
1360
1370
1380
1390
1400
1410
1420
1430
1440
1450
1460
1470
1480
1490
1500
1510
1520
1530
1540
1550
1560
1570
1580
1590
1600
1610
1620
1630
1640
1650
1660
1670
1680
1690
1700
1710
1720
1730
1740
1750
1760
1770
1780
1790
1800
1810
1820
1830
1840
1850
1860
1870
1880
1890
1895
1900
1905
1910
1915
1920
1925
1930
1935
1940
1945
1950
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
2030
2035
2040
2045
2050
2055
2060
2065
2070
2075
2080
2085
2090
2095
2100
2105
2110
2115
2120
2125
2130
2135
2140
2145
2150
2155
2160
2165
2170
2175
2180
2185
2190
2195
2200
2205
2210
2215
2220
2225
2230
2235
2240
2245
2250
2255
2260
2265
2270
2275
2280
2285
2290
2295
2300
2305
2310
2315
2320
2325
2330
2335
2340
2345
2350
2355
2360
2365
2370
2375
2380
2385
2390
2395
2400
2405
2410
2415
2420
2425
2430
2435
2440
2445
2450
2455
2460
2465
2470
2475
2480
2485
2490
2495
2500
2505
2510
2515
2520
2525
2530
2535
2540
2545
2550
2555
2560
2565
2570
2575
2580
2585
2590
2595
2600
2605
2610
2615
2620
2625
2630
2635
2640
2645
2650
2655
2660
2665
2670
2675
2680
2685
2690
2695
2700
2705
2710
2715
2720
2725
2730
2735
2740
2745
2750
2755
2760
2765
2770
2775
2780
2785
2790
2795
2800
2805
2810
2815
2820
2825
2830
2835
2840
2845
2850
2855
2860
2865
2870
2875
2880
2885
2890
2895
2900
2905
2910
2915
2920
2925
2930
2935
2940
2945
2950
2955
2960
2965
2970
2975
2980
2985
2990
2995
3000
3005
3010
3015
3020
3025
3030
3035
3040
3045
3050
3055
3060
3065
3070
3075
3080
3085
3090
3095
3100
3105
3110
3115
3120
3125
3130
3135
3140
3145
3150
3155
3160
3165
3170
3175
3180
3185
3190
3195
3200
3205
3210
3215
3220
3225
3230
3235
3240
3245
3250
3255
3260
3265
3270
3275
3280
3285
3290
3295
3300
3305
3310
3315
3320
3325
3330
3335
3340
3345
3350
3355
3360
3365
3370
3375
3380
3385
3390
3395
3400
3405
3410
3415
3420
3425
3430
3435
3440
3445
3450
3455
3460
3465
3470
3475
3480
3485
3490
3495
3500
3505
3510
3515
3520
3525
3530
3535
3540
3545
3550
3555
3560
3565
3570
3575
3580
3585
3590
3595
3600
3605
3610
3615
3620
3625
3630
3635
3640
3645
3650
3655
3660
3665
3670
3675
3680
3685
3690
3695
3700
3705
3710
3715
3720
3725
3730
3735
3740
3745
3750
3755
3760
3765
3770
3775
3780
3785
3790
3795
3800
3805
3810
3815
3820
3825
3830
3835
3840
3845
3850
3855
3860
3865
3870
3875
3880
3885
3890
3895
3900
3905
3910
3915
3920
3925
3930
3935
3940
3945
3950
3955
3960
3965
3970
3975
3980
3985
3990
3995
4000
4005
4010
4015
4020
4025
4030
4035
4040
4045
4050
4055
4060
4065
4070
4075
4080
4085
4090
4095
4100
4105
4110
4115
4120
4125
4130
4135
4140
4145
4150
4155
4160
4165
4170
4175
4180
4185
4190
4195
4200
4205
4210
4215
4220
4225
4230
4235
4240
4245
4250
4255
4260
4265
4270
4275
4280
4285
4290
4295
4300
4305
4310
4315
4320
4325
4330
4335
4340
4345
4350
4355
4360
4365
4370
4375
4380
4385
4390
4395
4400
4405
4410
4415
4420
4425
4430
4435
4440
4445
4450
4455
4460
4465
4470
4475
4480
4485
4490
4495
4500
4505
4510
4515
4520
4525
4530
4535
4540
4545
4550
4555
4560
4565
4570
4575
4580
4585
4590
4595
4600
4605
4610
4615
4620
4625
4630
4635
4640
4645
4650
4655
4660
4665
4670
4675
4680
4685
4690
4695
4700
4705
4710
4715
4720
4725
4730
4735
4740
4745
4750
4755
4760
4765
4770
4775
4780
4785
4790
4795
4800
4805
4810
4815
4820
4825
4830
4835
4840
4845
4850
4855
4860
4865
4870
4875
4880
4885
4890
4895
4900
4905
4910
4915
4920
4925
4930
4935
4940
4945
4950
4955
4960
4965
4970
4975
4980
4985
4990
4995
5000
5005
5010
5015
5020
5025
5030
5035
5040
5045
5050
5055
5060
5065
5070
5075
5080
5085
5090
5095
5100
5105
5110
5115
5120
5125
5130
5135
5140
5145
5150
5155
5160
5165
5170
5175
5180
5185
5190
5195
5200
5205
5210
5215
5220
5225
5230
5235
5240
5245
5250
5255
5260
5265
5270
5275
5280
5285
5290
5295
5300
5305
5310
5315
5320
5325
5330
5335
5340
5345
5350
5355
5360
5365
5370
5375
5380
5385
5390
5395
5400
5405
5410
5415
5420
5425
5430
5435
5440
5445
5450
5455
5460
5465
5470
5475
5480
5485
5490
5495
5500
5505
5510
5515
5520
5525
5530
5535
5540
5545
5550
5555
5560
5565
5570
5575
5580
5585
5590
5595
5600
5605
5610
5615
5620
5625
5630
5635
5640
5645
5650
5655
5660
5665
5670
5675
5680
5685
5690
5695
5700
5705
5710
5715
5720
5725
5730
5735
5740
5745
5750
5755
5760
5765
5770
5775
5780
5785
5790
5795
5800
5805
5810
5815
5820
5825
5830
5835
5840
5845
5850
5855
5860
5865
5870
5875
5880
5885
5890
5895
5900
5905
5910
5915
5920
5925
5930
5935
5940
5945
5950
5955
5960
5965
5970
5975
5980
5985
5990
5995
6000
6005
6010
6015
6020
6025
6030
6035
6040
6045
6050
6055
6060
6065
6070
6075
6080
6085
6090
6095
6100
6105
6110
6115
6120
6125
6130
6135
6140
6145
6150
6155
6160
6165
6170
6175
6180
6185
6190
6195
6200
6205
6210
6215
6220
6225
6230
6235
6240
6245
6250
6255
6260
6265
6270
6275
6280
6285
6290
6295
6300
6305
6310
6315
6320
6325
6330
6335
6340
6345
6350
6355
6360
6365
6370
6375
6380
6385
6390
6395
6400
6405
6410
6415
6420
6425
6430
6435
6440
6445
6450
6455
6460
6465
6470
6475
6480
6485
6490
6495
6500
6505
6510
6515
6520
6525
6530
6535
6540
6545
6550
6555
6560
6565
6570
6575
6580
6585
6590
6595
6600
6605
6610
6615
6620
6625
6630
6635
6640
6645
6650
6655
6660
6665
6670
6675
6680
6685
6690
6695
6700
6705
6710
6715
6720
6725
6730
6735
6740
6745
6750
6755
6760
6765
6770
6775
6780
6785
6790
6795
6800
6805
6810
6815
6820
6825
6830
6835
6840
6845
6850
6855
6860
6865
6870
6875
6880
6885
6890
6895
6900
6905
6910
6915
6920
6925
6930
6935
6940
6945
6950
6955
6960
6965
6970
6975
6980
6985
6990
6995
7000
7005
7010
7015
7020
7025
7030
7035
7040
7045
7050
7055
7060
7065
7070
7075
7080
7085
7090
7095
7100
7105
7110
7115
7120
7125
7130
7135
7140
7145
7150
7155
7160
7165
7170
7175
7180
7185
7190
7195
7200
7205
7210
7215
7220
7225
7230
7235
7240
7245
7250
7255
7260
7265
7270
7275
7280
7285
7290
7295
7300
7305
7310
7315
7320
7325
7330
7335
7340
7345
7350
7355
7360
7365
7370
7375
7380
7385
7390
7395
7400
7405
7410
7415
7420
7425
7430
7435
7440
7445
7450
7455
7460
7465
7470
7475
7480
7485
7490
7495
7500
7505
7510
7515
7520
7525
7530
7535
7540
7545
7550
7555
7560
7565
7570
7575
7580
7585
7590
7595
7600
7605
7610
7615
7620
7625
7630
7635
7640
7645
7650
7655
7660
7665
7670
7675
7680
7685
7690
7695
7700
7705
7710
7715
7720
7725
7730
7735
7740
7745
7750
7755
7760
7765
7770
7775
7780
7785
7790
7795
7800
7805
7810
7815
7820
7825
7830
7835
7840
7845
7850
7855
7860
7865
7870
7875
7880
7885
7890
7895
7900
7905
7910
7915
7920
7925
7930
7935
7940
7945
7950
7955
7960
7965
7970
7975
7980
7985
7990
7995
8000
8005
8010
8015
8020
8025
8030
8035
8040
8045
8050
8055
8060
8065
8070
8075
8080
8085
8090
8095
8100
8105
8110
8115
8120
8125
8130
8135
8140
8145
8150
8155
8160
8165
8170
8175
8180
8185
8190
8195
8200
8205
8210
8215
8220
8225
8230
8235
8240
8245
8250
8255
8260
8265
8270
8275
8280
8285
8290
8295
8300
8305
8310
8315
8320
8325
8330
8335
8340
8345
8350
8355
8360
8365
8370
8375
8380
8385
8390
8395
8400
8405
8410
8415
8420
8425
8430
8435
8440
8445
8450
8455
8460
8465
8470
8475
8480
8485
8490
8495
8500
8505
8510
8515
8520
8525
8530
8535
8540
8545
8550
8555
8560
8565
8570
8575
8580
8585
8590
8595
8600
8605
8610
8615
8620
8625
8630
8635
8640
8645
8650
8655
8660
8665
8670
8675
8680
8685
8690
8695
8700
8705
8710
8715
8720
8725
8730
8735
8740
8745
8750
8755
8760
8765
8770
8775
8780
8785
8790
8795
8800
8805
8810
8815
8820
8825
8830
8835
8840
8845
8850
8855
8860
8865
8870
8875
8880
8885
8890
8895
8900
8905
8910
8915
8920
8925
8930
8935
8940
8945
8950
8955
8960
8965
8970
8975
8980
8985
8990
8995
9000
9005
9010
9015
9020
9025
9030
9035
9040
9045
9050
9055
9060
9065
9070
9075
9080
9085
9090
9095
9100
9105
9110
9115
9120
9125
9130
9135
9140
9145
9150
9155
9160
9165
9170
9175
9180
9185
9190
9195
9200
9205
9210
9215
9220
9225
9230
9235
9240
9245
9250
9255
9260
9265
9270
9275
9280
9285
9290
9295
9300
9305
9310
9315
9320
9325
9330
9335
9340
9345
9350
9355
9360
9365
9370
9375
9380
9385
9390
9395
9400
9405
9410
9415
9420
9425
9430
9435
9440
9445
9450
9455
9460
9465
9470
9475
9480
9485
9490
9495
9500
9505
9510
9515
9520
9525
9530
9535
9540
9545
9550
9555
9560
9565
9570
9575
9580
9585
9590
9595
9600
9605
9610
9615
9620
9625
9630
9635
9640
9645
9650
9655
9660
9665
9670
9675
9680
9685
9690
9695
9700
9705
9710
9715
9720
9725
9730
9735
9740
9745
9750
9755
9760
9765
9770
9775
9780
9785
9790
9795
9800
9805
9810
9815
9820
9825
9830
9835
9840
9845
9850
9855
9860
9865
9870
9875
9880
9885
9890
9895
9900
9905
9910
9915
9920
9925
9930
9935
9940
9945
9950
9955
9960
9965
9970
9975
9980
9985
9990
9995
10000
10005
10010
10015
10020
10025
10030
10035
10040
10045
10050
10055
10060
10065
10070
10075
10080
10085
10090
10095
10100
10105
10110
10115
10120
10125
10130
10135
10140
10145
10150
10155
10160
10165
10170
10175
10180
10185
10190
10195
10200
10205
10210
10215
10220
10225
10230
10235
10240
10245
1025
```



The screenshot shows an RStudio interface with a terminal window and a code editor. The terminal window displays the output of the 'ifconfig' command. The code editor contains the following R script:

```

1 #> using WITTEN
2
3 # hduser@ubuntu:~$ ifconfig
4 # eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
#           inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:255.255.255.0
5
6 v1 = 0:20
7 v2 = 30:50
8 v3 = 60:80
9
10 mydf = data.frame(v1,v2,v3)
11 #fix(mydf)
12 #which returns index of the satisfying data
13 #ans_v = which(mydf[,1]>5)
14 #cat("index of ans: ")
15 print(mydf[which(mydf[,1]>5),])
16 print(mydf[which(mydf[,1]>5),3])
17

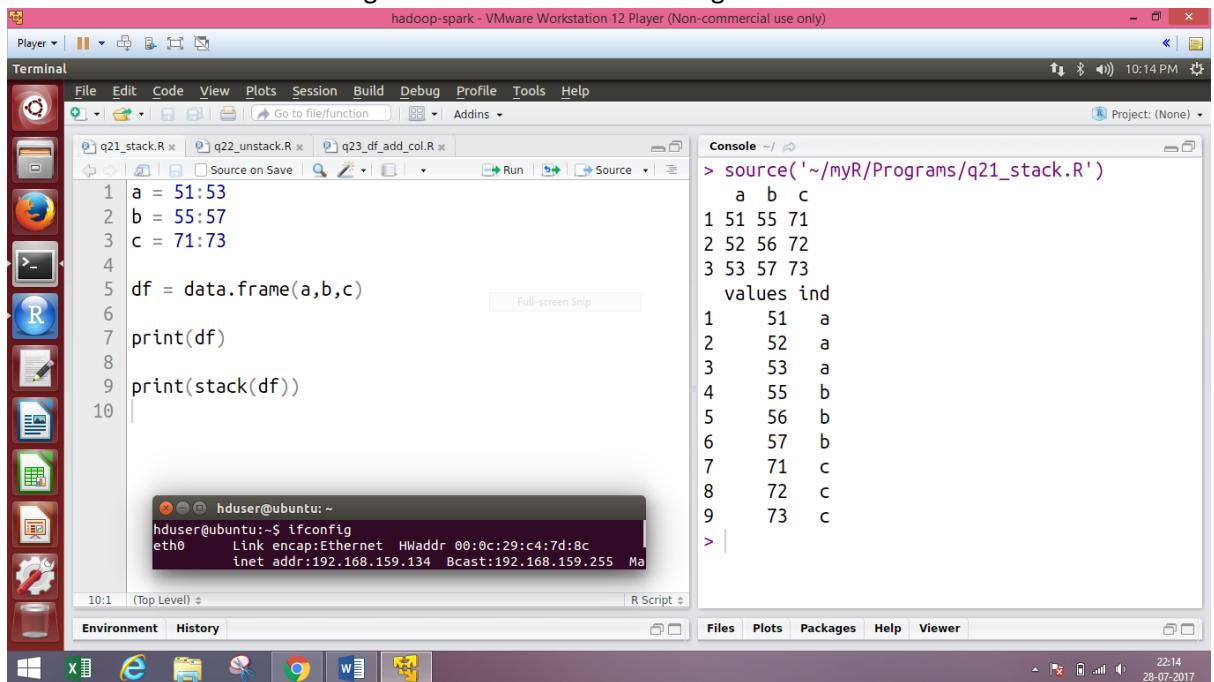
```

The console window shows the output of the script, including the creation of a data frame 'mydf' and its printing. The output also includes the results of the 'which()' function applied to the first column and then to the third column.

Figure 38: which() function on column and row of Data Frame

e. stack() function

The stack command will stack all the vectors in a data frame into a single column along with a factor indicating where each observation originated.



The screenshot shows an RStudio interface with a terminal window and a code editor. The terminal window displays the output of the 'ifconfig' command. The code editor contains the following R script:

```

1 a = 51:53
2 b = 55:57
3 c = 71:73
4
5 df = data.frame(a,b,c)
6
7 print(df)
8
9 print(stack(df))
10

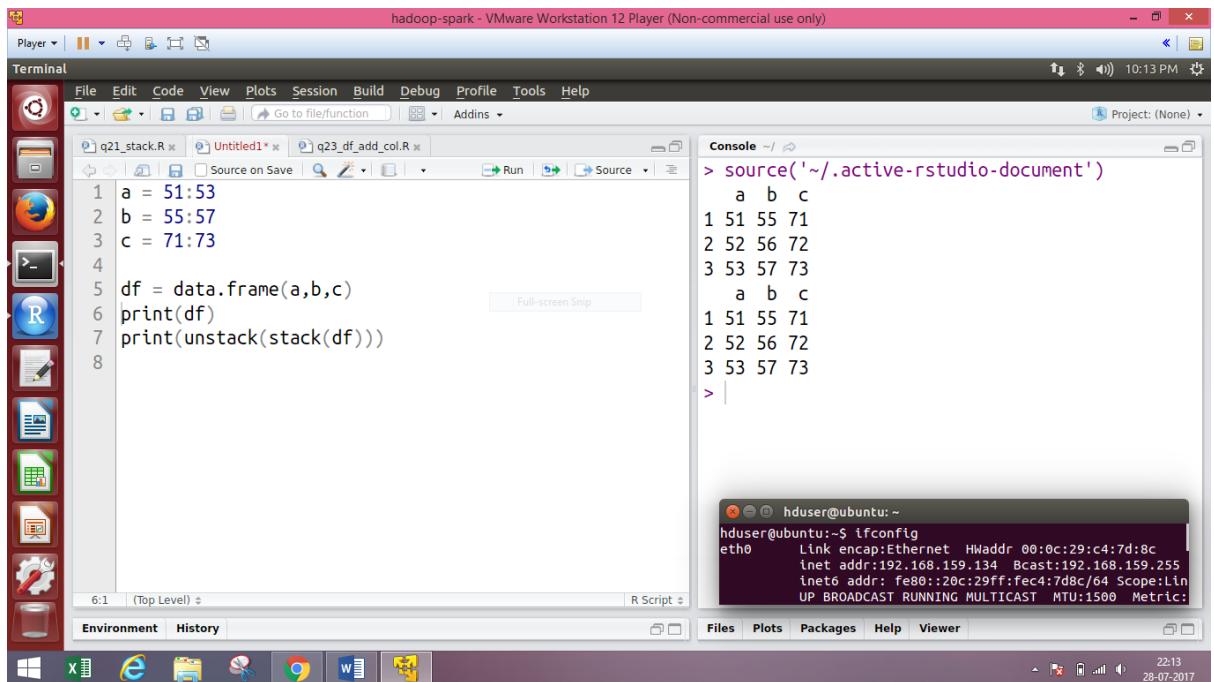
```

The console window shows the output of the script, including the creation of a data frame 'df' and its printing. The output also includes the results of the 'stack()' function applied to the data frame.

Figure 39: q21 - Stack data from Data Frame

f. unstack() function

Unstacking reverses operation of stack.



The screenshot shows an RStudio interface running on a Windows host. The left pane displays an R script named 'q21_stack.R' with the following code:

```

1 a = 51:53
2 b = 55:57
3 c = 71:73
4
5 df = data.frame(a,b,c)
6 print(df)
7 print(unstack(stack(df)))
8

```

The right pane shows the R console output:

```

> source('~/active-rstudio-document')
a b c
1 51 55 71
2 52 56 72
3 53 57 73
a b c
1 51 55 71
2 52 56 72
3 53 57 73
>

```

Below the RStudio window, a terminal window is open showing system information:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          ...

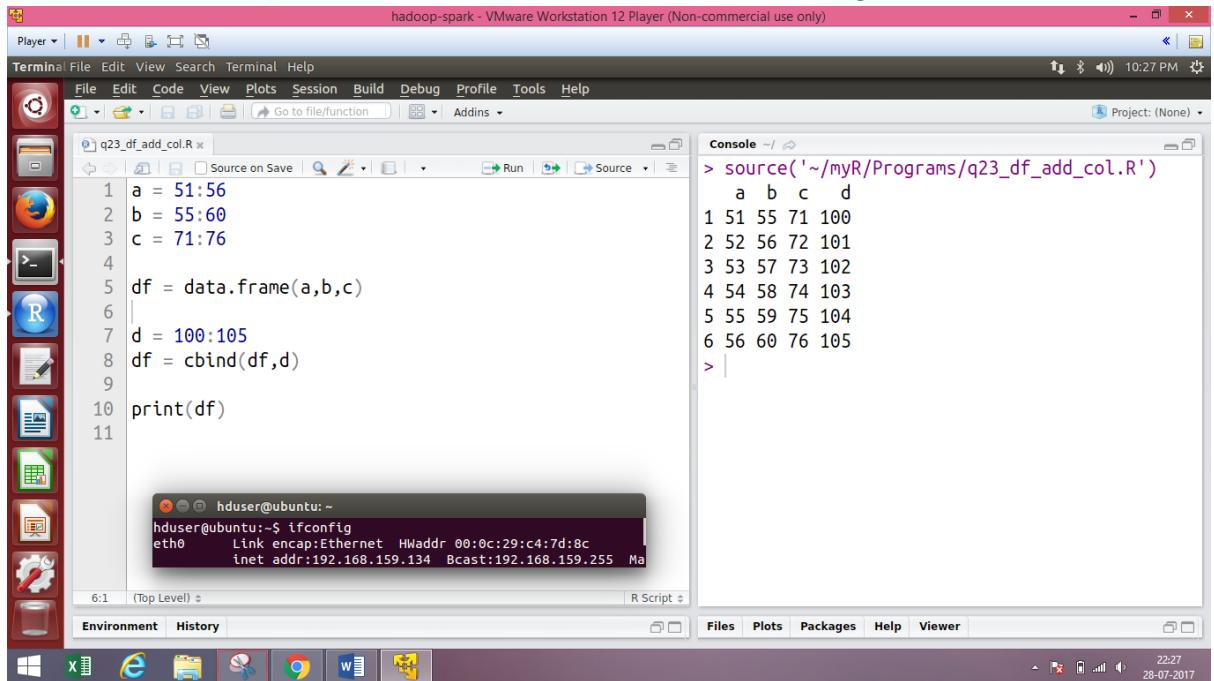
```

Figure 40: q22 - Unstack vector back to Data Frame

g. Expand Data Frame

i. Add Column using cbind() function

`cbind()` function is used to add more columns to existing Data Frame.



The screenshot shows an RStudio interface running on a Windows host. The left pane displays an R script named 'q23_df_add_col.R' with the following code:

```

1 a = 51:56
2 b = 55:60
3 c = 71:76
4
5 df = data.frame(a,b,c)
6
7 d = 100:105
8 df = cbind(df,d)
9
10 print(df)
11

```

The right pane shows the R console output:

```

> source('~/myR/Programs/q23_df_add_col.R')
a b c d
1 51 55 71 100
2 52 56 72 101
3 53 57 73 102
4 54 58 74 103
5 55 59 75 104
6 56 60 76 105
>

```

Below the RStudio window, a terminal window is open showing system information:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          ...

```

Figure 41: q23 - cbind() function to add column in Data Frame

ii. Add Row using rbind() function

`rbind()` function is used to add rows to an existing data frame.

The screenshot shows the RStudio interface. On the left, there's a file browser with icons for various files. In the center-left, there's a code editor window with two tabs: 'q23_df_add_col.R' and 'q24_df_add_row.R'. The code in 'q24_df_add_row.R' is:

```

1 a = 51:56
2 b = 55:60
3 c = 71:76
4
5 df = data.frame(a,b,c)
6
7 row6 = 1000:1003
8 df = rbind(df, row6)
9
10 print(df)
11

```

On the right, the 'Console' tab shows the output of running the script:

```

> source('~/myR/Programs/q24_df_add_row.R')
      a     b     c
1  51    55    71
2  52    56    72
3  53    57    73
4  54    58    74
5  55    59    75
6  56    60    76
7 1000 1001 1002
>

```

Below the console, a terminal window shows the command 'ifconfig' output:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:

```

Figure 42: q24 - rbind() function to add row to existing Data Frame

h. Order Data Frame using order() function

order() function returns a permutation which rearranges its first argument into ascending or descending order, breaking ties by further arguments.

sort.list() is the same, using only one argument.

Use:

```
order(..., na.last = TRUE, decreasing = FALSE)
```

```
sort.list(x, partial = NULL, na.last = TRUE, decreasing = FALSE, method =
c("shell", "quick", "radix"))
```

The screenshot shows the RStudio interface. On the left, there's a file browser with icons for various files. In the center-left, there's a code editor window with three tabs: 'q38_df_inbuilt_pick_selecive.R', 'q39_df_subset.R', and 'q40_df_order.R'. The code in 'q40_df_order.R' is:

```

1 #Order the new-arrest data frame by the population
2
3 #Note: for descending ordering of the data frame
4
5 newArrest = subset(USArrests, UrbanPop & Rape >=25,
6                     select = c(UrbanPop, Rape))
7 sorted = newArrest[order(newArrest$UrbanPop, ),]
8 print(sorted)
9
10 sorted=newArrest[order(newArrest$UrbanPop, decreasing = TRUE),
11                  print(sorted)

```

On the right, the 'Console' tab shows the output of running the script:

	UrbanPop	Rape
California	91	40.6
New York	86	26.1
Nevada	81	46.0
Arizona	80	31.0
Florida	80	31.9
Texas	80	25.5
Colorado	78	38.7
Michigan	74	35.1
Washington	73	26.2
Missouri	70	28.2
New Mexico	70	32.1
Maryland	67	27.8
Oregon	67	29.3
Georgia	60	25.8
Tennessee	59	26.9
Alaska	48	44.5

Below the console, a terminal window shows the command 'ifconfig' output:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:

```

Figure 43: Order Data Frame

i. Transpose of Data Frame

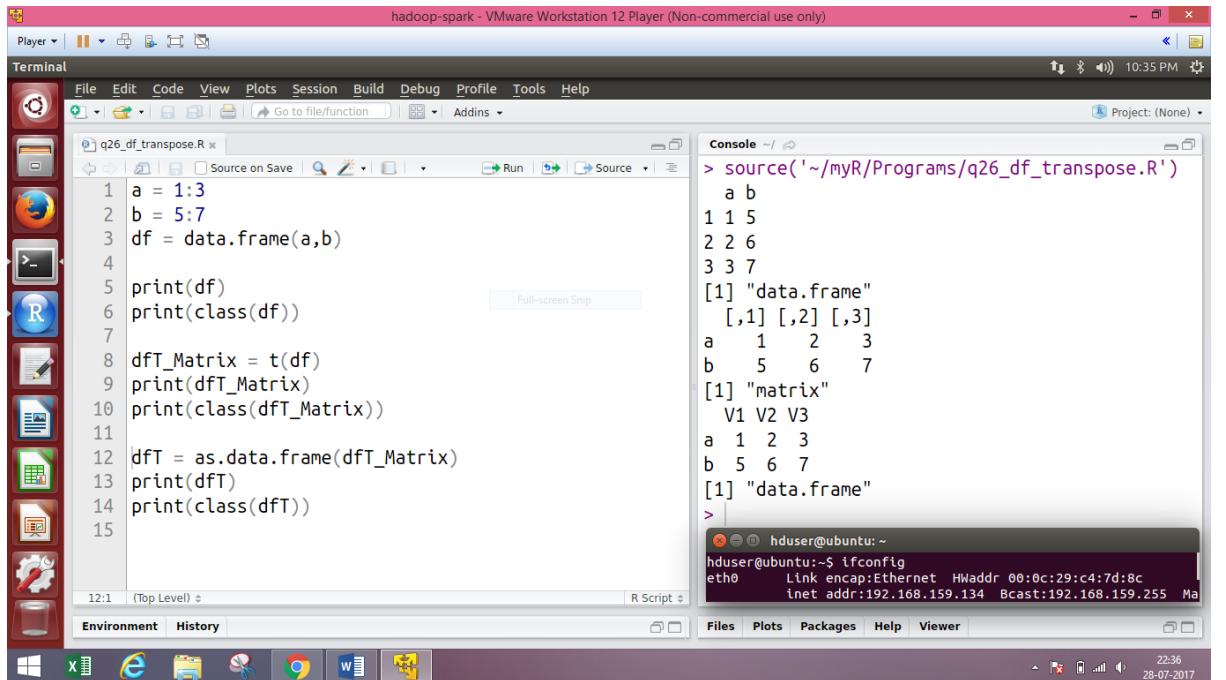
Transpose using `t()` function which interchanges row and columns of Data Frame.

Result of this is matrix instead of data frame. See [Data Frame from Matrix](#)

To convert it back to data.frame(result_matrix_object_name) is used.

Use:

`t(x)`



The screenshot shows the RStudio interface with the following details:

- Terminal:** Shows the R script `q26_df_transpose.R` and its output. The output shows the creation of a data frame `df` from vectors `a` and `b`, then its transpose `dft` as a matrix, and finally `dft` converted back to a data frame.
- Console:** Shows the command `> source('~/myR/Programs/q26_df_transpose.R')` and the resulting output. It prints the data frame `df` (rows 1, 2, 3), its transpose `dft` as a matrix (rows V1, V2, V3), and the transpose `dft` converted back to a data frame (rows a, b).
- Environment:** Shows the variables `a` and `b` defined in the workspace.
- History:** Shows the history of commands run in the console.
- Bottom Status Bar:** Shows the date and time as 28-07-2017 22:36.

Figure 44: q26 - `t()` function to transpose the Data Frame

j. Functions on Vector

i. `range()` function

`range()` function returns a vector containing the minimum and maximum of all the given arguments.

Use:

`range(..., na.rm = FALSE)`

Default S3 method:

`range(..., na.rm = FALSE, finite = FALSE)`

The screenshot shows the RStudio interface. On the left, there's a file browser with icons for various files and a project pane labeled '(None)'. The main area has tabs for 'Source on Save' and 'Console'. The 'Console' tab is active, displaying the output of running an R script named 'q32_range.R'. The script contains the following code:

```

1 #range()
2 #Produce lowest and highest values of the vector v
3 
4 a = 1:10
5 cat("\n Range of A is : ",range(a))
6 
7 b = c(57,70,5,75,9,17,21,22,8,4)
8 cat("\n Range of B is : ",range(b))
9 
10 c = c("Sima","Gazal","Kavita","Samay","Ravindra")
11 cat("\n Range of C is : ",range(c))

```

The console output shows the ranges for each vector:

```

Range of A is :  1 10
Range of B is :  4 75
Range of C is :  Gazal Sima
> |

```

Figure 45: q32 - range() function

ii. pretty() function

It produces a new factor containing the elements from the original vector, with the specified intervals

Compute a sequence of about $n+1$ equally spaced ‘round’ values which cover the range of the values in x . The values are chosen so that they are 1, 2 or 5 times a power of 10.

Use:

```
pretty(x, ...)
```

```
## Default S3 method:
pretty(x, n = 5, min.n = n %/% 3, shrink.sml = 0.75, high.u.bias = 1.5,
u5.bias = .5 + 1.5*high.u.bias, eps.correct = 0, ...)
```

```

#Pretty function: produces a new vector containing the
# from the original vector,
# with the specified intervals
#
# Pretty(x,y) # y is the number of intervals in x
#
a = 1:10
cat(pretty(a,2))
cat("\n",pretty(a,5))

```

Console ~ /

```

> source('~/myR/Programs/q34_pretty.R')
0 5 10
0 2 4 6 8 10
>

```

Files Plots Packages Help Viewer

R: Pretty Breakpoints Find in Topic

pretty {base}

Pretty Breakpoints

Description

Compute a sequence of about n+1 equally spaced 'round' values which cover the range of the values in x. The values are chosen so that they are 1, 2 or 5 times a power of 10.

Usage

Figure 46: q33 - pretty() function

7. Working with Factors

A type vector contains a set of numeric codes with character valued levels. The function factor is used to encode a vector as a factor (the terms ‘category’ and ‘enumerated type’ are also used for factors). Factors are used in plotting graphs while working with categorical data. The default method combines its arguments to form a vector. All arguments are coerced to a common type which is the type of the returned value, and all attributes except names are removed. It automatically gives numeric value to all categories available in data so mode of factor returns numeric type.

```

# Will character valued levels
#
# example: a family of two girls(1) and four boys(0)
# kids = factor(c(1,0,1,0,0,0), levels = c(0,1), labels = c("boy", "girl"))
#
kids = factor(c(1,0,1,0,0,0),levels = c(0,1), labels = c("boy", "girl"))
print(kids)
print(class(kids))
print(mode(kids))
f1 = factor(c("pig", "hive", "hbase", "pig"))
print(nlevels(f1))

```

Console ~ /

```

> source('~/myR/Programs/practice_factor.R')
[1] girl boys girl boys boys
Levels: boys girl
[1] "factor"
[1] "numeric"
[1] 3
>

```

hduser@ubuntu:~

ifconfig

eth0 Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:

Figure 47: q34 - Factors

- **(Q35) c() Function**

c() in (c("pig","hive","hbase","pig")) stands for concatenation function. This is a generic function which combines its arguments.

- **nlevel() Function**

This function returns the number of unique data in list.

- **append() Function**

It is used to add factor data in factor. Including data directly using append won't work as expectedly because mode of factor is numeric and auto generated so

The screenshot shows the RStudio interface with the following details:

- Code Editor:** Displays the R script `q36_append_factor.R` containing the following code:

```

4 f1 = factor(c("hello", "world", 11, "hello", 11))
5 print(f1)
6 cat("\n")
7 print(levels(f1))
8 cat("\n")
9 f2 = append(f1,c("xyz", "hello", 11, 22))
10 print(f2)
11 cat("\n")
12 print(levels(f2))
13 cat("\n")
14 f3 = append(f1,factor(c("xyz", "hello", 11, 22)))
15 print(f3)
16 cat("\n")
17 print(levels(f3))
18 cat("\n")
19
20

```
- Console:** Shows the output of the R code:

```

> source('~/myR/Programs/q36_append_factor.R')
[1] hello world 11    hello 11
Levels: 11 hello world

[1] "11"    "hello" "world"
[1] "2"     "3"     "1"     "2"     "1"
[6] "xyz"   "hello" "11"   "22"

NULL

[1] 2 3 1 2 1 4 3 1 2

NULL
>

```
- Terminal:** Shows the command `ifconfig` being run in the terminal window:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255

```

Figure 48: q36 - append() in Data Frame

- **unclass() function**

It generates number sequence which are assigned to values inside the factor [q] the numbers are assigned based on dictionary order of sorting

The screenshot shows the RStudio interface with the following details:

- Terminal:** Shows the R script `q36_append_factor.R` and its output in the Console.
- Console:** Displays the execution of `unclass()` on a factor object, showing the underlying integer representation and the levels.
- Environment:** Shows the current environment variables.
- History:** Shows the command history.
- System:** Shows the system tray and taskbar.

```

1 # unclass() it generates number sequence which are as
2 # to values inside the factor [q] the numbers are assigned
3 # based on dictionary order of sorting
4
5 sizes = c("small", "medium", "large", "small")
6 f = factor(sizes)
7 print(f)
8 cat("\n unclassing ---> \n")
9 unf = unclass(f)
10 print(unf)
11 cat("\n total levels ---> \n")
12 nf = nlevels(f)
13 print(nf)

> source('~/myR/Programs/practice_factor_unclass_levels.R')
[1] small medium large small
Levels: large medium small

unclassing --->
[1] 3 2 1 3
attr(,"levels")
[1] "large" "medium" "small"

total levels --->
[1] 3
>

```

Figure 49: `unclass()` function for factors

- **table() function**

Create table from factors.

The screenshot shows the RStudio interface with the following details:

- Terminal:** Shows the R script `practice_factor_table.R` and its output in the Console.
- Console:** Displays the creation of two factor objects `f1` and `f2`, and their corresponding tables `t1` and `t2`.
- Environment:** Shows the current environment variables.
- History:** Shows the command history.
- System:** Shows the system tray and taskbar.

```

> source('~/myR/Programs/practice_factor_table.R')
f1
hbase    hive    pig
      1      1      2
f2
large   medium   small
      1      2      1
>

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:
          ...

```

Figure 50: `table()` to make table from factors

8. Subset

It is function that returns subsets of vectors, matrices or data frames which meet conditions.

Use:

`subset(x, ...)`

`## Default S3 method:`

```

subset(x, subset, ...)

## S3 method for class 'matrix'
subset(x, subset, select, drop = FALSE, ...)

## S3 method for class 'data.frame'
subset(x, subset, select, drop = FALSE, ...)

```

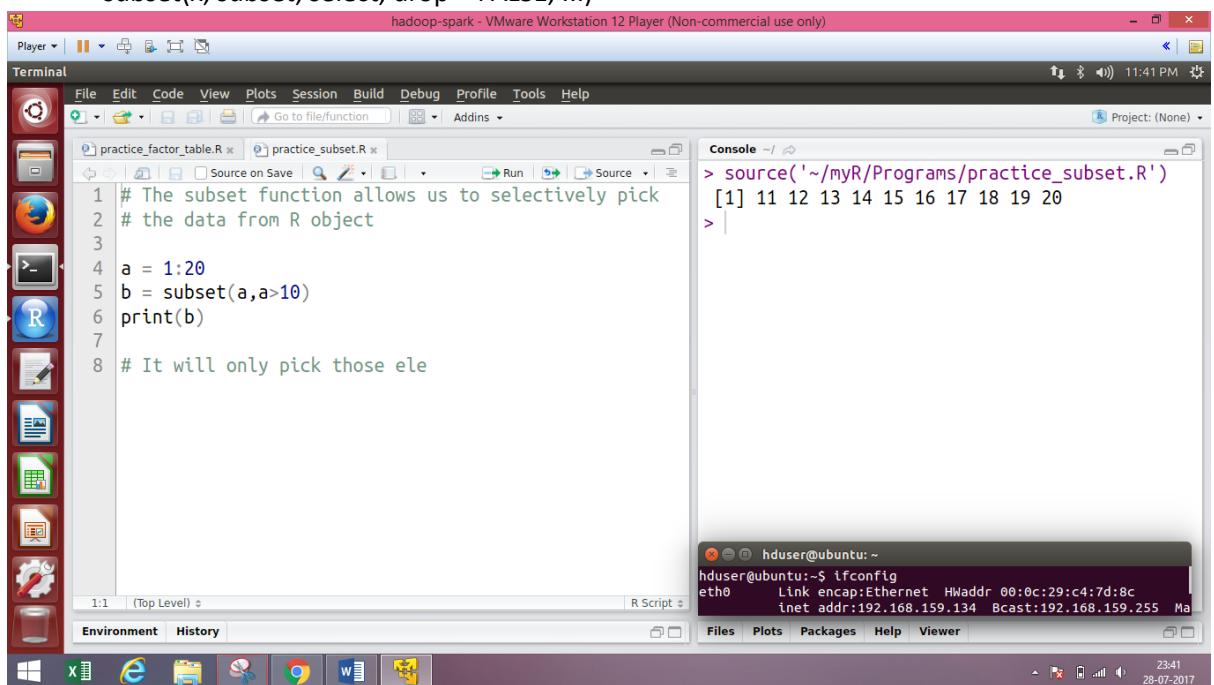


Figure 51: q - subset() function

a. Create Subset from Data Frame

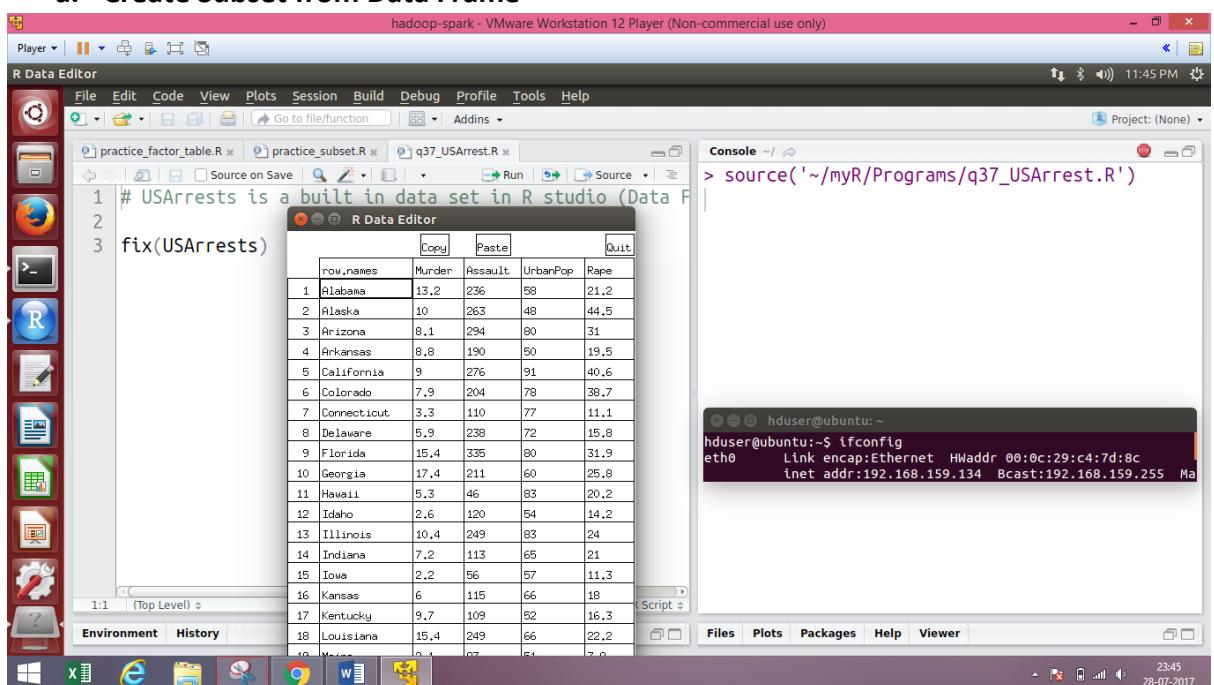


Figure 52: q37 - subset() function on Data Frame

b. Selection from Data Frame

The screenshot shows the RStudio interface running on a Windows host. The left pane displays an R script named 'q38_df_inbuilt_pick_selecive.R' with the following code:

```

1 # Pick all columns based on the condition that the pop
2 # is greater than 70 million(3rd column)
3
4 newArrest = subset(USArrests, UrbanPop>=70,
5   select = c(Murder, Assault, UrbanPop, Rape))
6 print(newArrest)
7 #fix(USArrests)

```

The right pane shows the R console output, which includes a table of US state arrest statistics and a terminal window showing network configuration:

	UrbanPop	Rape
Delaware	5.9	238
Florida	15.4	335
Hawaii	5.3	46
Illinois	10.4	249
Massachusetts	4.4	149
Michigan	12.1	255
Missouri	9.0	178
Nevada	12.2	252
New Jersey	7.4	159
New Mexico	11.4	285
New York	11.1	254
Ohio	7.3	120
Pennsylvania	6.3	106
Rhode Island	3.4	174
Texas	12.7	201
Utah	3.2	120
Washington	4.0	145

Terminal output:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:255.255.255.0

```

Figure 53: q38 - Pick up columns based on the condition using subset() function on Data Frame

The screenshot shows the RStudio interface running on a Windows host. The left pane displays an R script named 'q39_df_subset.R' with the following code:

```

1 #skipset = c(2,3,12,14:21,23)
2 #newArrest = subset(USArrests, UrbanPop & Rape >=25,
3 #                     select = c(UrbanPop, Rape), drop
4
5 newArrest = subset(USArrests, UrbanPop & Rape >=25,
6                     select = c(UrbanPop, Rape))
7 print(newArrest)
8 #fix(USArrests)

```

The right pane shows the R console output, which includes a table of US state arrest statistics and a terminal window showing network configuration:

	UrbanPop	Rape
Alaska	48	44.5
Arizona	80	31.0
California	91	40.6
Colorado	78	38.7
Florida	80	31.9
Georgia	60	25.8
Maryland	67	27.8
Michigan	74	35.1
Missouri	70	28.2
Nevada	81	46.0
New Mexico	70	32.1
New York	86	26.1
Oregon	67	29.3
Tennessee	59	26.9
Texas	80	25.5
Washington	73	26.2

Terminal output:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:255.255.255.0

```

Figure 54: q39 - Subsetting Data Frame

9. Table

The Structure of data in tables and data frames are the same however operations like join, in better way than that of data frames.

Function table() uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels and gives a frequency count of how many times each number is repeated in the above vector.

Use:

```
table(..., exclude = if (useNA == "no") c(NA, NaN), useNA = c("no",
  "ifany", "always"), dnn = list.names(...), deparse.level = 1)
```

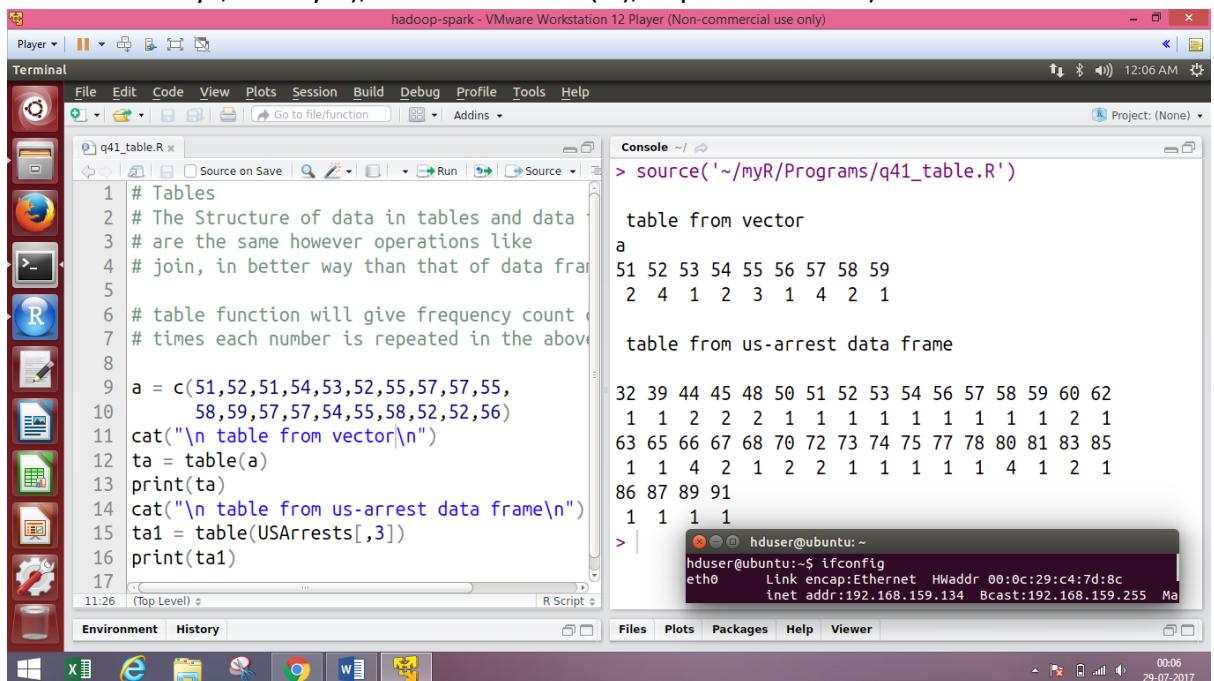


Figure 55: q41 - `table()` function, Create table from vector

Function `table()` with multiple arguments.

For example,

```
table(USArrests[,3], USArrests[,4])
```

First argument becomes column names, and second row names.

The table elements will be the frequency count.

The screenshot shows an RStudio interface running on a Windows host. The left pane displays an R script with code to create a frequency table from two vectors 'a' and 'b'. The right pane shows the console output where the table is printed. Below the RStudio window, a terminal window shows network configuration details.

```

source('~/active-rstudio-document')
table from vector
  b
a   15 25 35 45 55 57 58 65 95
51  1  1  0  0  0  0  0  0  0
52  1  1  1  0  0  0  0  0  1  0
53  0  0  0  1  0  0  0  0  0  0
54  0  1  0  1  0  0  0  0  0  0
55  0  0  0  0  3  0  0  0  0  0
56  0  1  0  0  0  0  0  0  0  0
57  0  0  0  0  0  4  0  0  0  0
58  0  0  0  0  0  0  2  0  0  0
59  0  0  0  0  0  0  0  0  0  1

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134  Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:L
          
```

Figure 56: Table using `table()` function indicating frequency

(Q45) How to infer below table?

Number 'a' = 51 has occurred 1 times while other number was 'b' = 15
 Number 'a' = 51 has occurred 1 times while other number was 'b' = 25
 Number 'a' = 52 has occurred 1 times while other number was 'b' = 15
 Number 'a' = 52 has occurred 1 times while other number was 'b' = 25
 Number 'a' = 52 has occurred 1 times while other number was 'b' = 35
 Number 'a' = 53 has occurred 1 times while other number was 'b' = 45
 Number 'a' = 54 has occurred 1 times while other number was 'b' = 25
 Number 'a' = 54 has occurred 1 times while other number was 'b' = 45
 Number 'a' = 55 has been repeated 3 times while other number was 'b' = 55
 Number 'a' = 56 has occurred 1 times while other number was 'b' = 25
 Number 'a' = 57 has been repeated 4 times while other number was 'b' = 57
 Number 'a' = 58 has been repeated 2 times while other number was 'b' = 57
 Number 'a' = 56 has occurred 1 times while other number was 'b' = 25

Sum of particular column or row can be referred as occurrence of that data item in data item.

For example,

Number 'a' = 52 has occurred total 4 times in complete 'a' data set.

Number 'a' = 15 has occurred total 2 times in complete 'b' data set.

The screenshot shows the RStudio interface with the following details:

- Left Panel (Code Editor):** Contains two tabs: "q44_table_from_df.R" and "practice_airquality.R". The "practice_airquality.R" tab contains the following R code:

```
1 # airquality
2 # Data Frame consisting of Daily air quality
3 # measurements in New York, May to September
4 print(class(airquality))
5 print(head(airquality,5))
```
- Right Panel (Console):** Shows the output of the R code:

```
> source('~/myR/Programs/practice_airquality.R')
[1] "data.frame"
[1] "Ozone Solar.R Wind Temp Month Day
1   41    190  7.4  67   5   1
2   36    118  8.0  72   5   2
3   12    149 12.6  74   5   3
4   18    313 11.5  62   5   4
5   NA     NA 14.3  56   5   5
> |
```
- Bottom Panel (Terminal):** Shows a terminal window titled "hduser@ubuntu:~" with the command "ifconfig" run, displaying network interface information.

Figure 57: Table from Data Frame

The screenshot shows the RStudio interface with the following details:

- File Menu:** File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help.
- Console Tab:** Shows the command `source('~/myR/Programs/q42_table_from_matrix.R')` and its output:

```
> source('~/myR/Programs/q42_table_from_matrix.R')
      col1 col2 col3
row1     1    2    3
row2     4    5    6
row3     7    8    9
> |
```
- Code Editor:** Two files are open: `q41_table.R` and `q42_table_from_matrix.R`. The code in `q42_table_from_matrix.R` is as follows:

```
1 mymatrix = matrix(c(1:9),nrow = 3,byrow = T)
2
3 rownames(mymatrix) = c("row1","row2","row3")
4 colnames(mymatrix) = c("col1","col2","col3")
5
6 #`colnames<-`(`mymatrix, c("col1","col2","col3"))
7
8 mytable = as.table(mymatrix)
9 print(mytable)
```
- Terminal:** A terminal window titled "hduser@ubuntu:" is open, showing the output of the command `ifconfig`:

```
hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134  Bcast:192.168.159.255
          netmask:255.255.255.0  Brd:192.168.159.255
          inet6 addr: fe80::20c:29ff:fe4c:7d8c%64 Scope:L...
```
- Help:** A tooltip for the `table` function is displayed, providing usage information and arguments.

Figure 58: q42 - Table from Matrix

10. Difference

a. R vs. Non-R Programming

R Programming	Non-R Programming
<ul style="list-style-type: none">• In R, vector and list and arrays are different and has different properties.• List can be named or un-named.	<ul style="list-style-type: none">• Un-named list are Arrays• Named lists are dictionary• However Array and Dictionary can contain another Array or a

	dictionary, but it's still called Arrays and dictionaries.
--	--

b. Matrix vs. Data Frame

Matrix	Data Frame
<ul style="list-style-type: none"> • 2 dimensional data structure • Homogeneous since a matrix could contain only identical data type. 	<ul style="list-style-type: none"> • 2 dimensional data structure • Can contain more than one data type, It is made of collection of vectors which can contain different type of data objects

c. Vector vs. List vs. Array vs. Matrix vs. Data Frame

Vector
<ul style="list-style-type: none"> ○ Simplest form of R object ○ All values in a vector should be of same object/class ○ However they can't contain another vector or array or list. ○ (non-R programmers terms) it's like array but with all values in the array having same data-types
List
<ul style="list-style-type: none"> ○ It is like a vector but can contain any object/class. ○ Class also have named attributes. ○ It can contain list within a list etc. ○ (non-R programmers terms) it's like array (if not named) and dictionaries(if named)
Array
<ul style="list-style-type: none"> ○ It's a list with one, two or more dimensions ○ Dimensions can be named. ○ (non-R programmer's terms) it's like array within another array.
Matrix
<ul style="list-style-type: none"> ○ It's more like an array but with only 2 dimensions. ○ (non-R programmer's terms) it's still a matrix :)
Data Frame
<ul style="list-style-type: none"> ○ Data in table format, with rows and each column is an attribute. ○ when you read a csv file, you normally read that as dataFrames ○ (non-R programmers terms) Imagine table with rows and columns (with columnnames)

11.Package

R packages are a collection of R functions, complied code and sample data. They are stored under a directory called "library" in the R environment. By default, R installs a set of packages during installation. More packages are added later, when they are needed for some specific purpose.

When we start the R console, only the default packages are available by default. Other packages which are already installed have to be loaded explicitly to be used by the R program that is going to use them.

- Check Available Packages

Use:

.libPaths()

- **(Q46) Get The List Of All Installed Packages**

Use:

```
library()
```

For Example:

```
library(XML)
```

```
library(methods)
```

- **Install Package Manually**

Use:

```
install.packages(file_name_with_path, repos = NULL, type = "source")
```

- **(Q47) Install Packages from CRAN**

Use:

```
install.packages("Package Name")
```

For Example:

```
install.packages("slidify")
```

Note: An archive internet connection is necessary.

- **Load Package from Library**

Use:

```
library("package Name", lib.loc = "path to library")
```

12.File IO Operations

a. CSV File

i. **Read.csv()**

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Use:

```
read.csv(file, header = TRUE, sep = ",", quote = "\"\"",  
dec = ".", fill = TRUE, comment.char = "", ...)
```

```

1 r1 = read.csv("/home/hduser/myR/Program_Data/csvnew.csv")
2 print(r1)

> source('~/myR/Programs/practice_csv_file.R')
      Name   Gender BirthDay BirthMonth
1  Gazal Female      5     August
2 Kavita Female     21 December
3  Samay  Male      17     August
4  Sima Female      9      April
5   Ravi  Male     13      May
>

```

Figure 59: `read.csv()` function to read .CSV file

1. Default Header from file

```

1 csvPath="/home/hduser/myR/Program_Data/csvnew.csv"
2 r1 = read.csv(csvPath)
3 print(r1)
4
5 r1 = read.csv(csvPath,header = F,
6                 stringsAsFactors = T)
7 print(r1)

> source('~/myR/Programs/practice_csv_file.R')
      V1   V2   V3   V4
1  Name Gender BirthDay BirthMonth
2  Gazal Female      05     August
3 Kavita Female     21 December
4  Samay  Male      17     August
5  Sima Female      09      April
6   Ravi  Male     13      May
>

```

Figure 60: q48 - Read csv with title header as data

2. Skip few lines from header

The screenshot shows the RStudio interface with a terminal window. The code in the script pane is:

```

1 # skip the first 3 lines from the csv while reading it
2 filenamepath = "/home/hduser/myR/Program_Data/writethiscsv.csv"
3 mydf = read.csv(filenamepath, skip = 4, header = F)
4 print(mydf)
5
6 #print(head(USArrests))
7 #print(head(mydf))
8 #Head function prints only the first 6 lines from the dataframe

```

The console output shows the data frame:

```

> source('~/myR/Programs/q55_csv_skip_read.R')
   V1      V2      V3
1 54 Ganesha Ganpati
2 55 Kishan Kanha
3 56 Mahadev Shankar
4 57 Vishnu Keshav
>

```

A terminal window at the bottom shows the command `ifconfig`:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fe:c47d:8c/64 Scope:Link

```

Figure 61: Skip Header lines

3. Read Only header

Note: .csv is compulsory in the file name argument

Row names will be added by default in the file being written.

Using the written using the write.csv command

The row names will be numbers starting from 1

Turn off this auto or default naming with row.names = F

The screenshot shows the RStudio interface with a terminal window. The code in the script pane is:

```

1 # Read only header in csv
2
3 filenamepath = "/home/hduser/myR/Program_Data/csvnew.csv"
4
5 header = read.csv(filenamepath, skip = 1, header = F
6           , nrow = 3, as.is = T)
7
8 print(header)

```

The console output shows the data frame:

```

> source('~/myR/Programs/q54_header.R')
   V1      V2      V3      V4
1 Gazal Female     5 August
2 Kavita Female    21 December
3 Samay  Male     17 August
>

```

A terminal window at the bottom shows the command `ifconfig`:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fe:c47d:8c/64 Scope:Link

```

Figure 62: q54 - Read csv file header

ii. Data type

It returns data.frame class object as result.

The screenshot shows the RStudio interface. On the left is a tool palette with icons for various functions. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help, and Addins. A toolbar below the menu has icons for file operations like Open, Save, and Run. The main workspace contains two tabs: q49_write_csv.R and q48_read_csv.R*. The q48_read_csv.R* tab displays the following R code:

```

1 # Print the contents of "a" and check data type of object a
2 # It will be a data frame and the first row from the file will be missing
3 # Since it will be considered as header, to include this,
4 # set the header option during import.
5
6 # header = F means, the first line of the file will be considered as data
7 cpath="/home/hduser/myR/Program_Data/AssetsImportCompleteSample.csv"
8 a = read.csv(cpath, fill = T, header = F, stringsAsFactors = F)
9 print(class(a))
10 #print(a)

```

Below the code is a terminal window showing the command ifconfig and its output:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fe4:7d8c/64 Scope:Link
          ...

```

At the bottom of the RStudio interface is a taskbar with icons for various applications.

Figure 63: Class of reader variable of csv file

iii. write.csv()

1. From Object

Note: Row names will be added by default starting from 1.

The screenshot shows the RStudio interface. The top menu bar and toolbar are identical to Figure 63. The workspace shows the q49_write_csv.R tab with the following R code:

```

1 # Write an data from data in file
2 # Note: .csv is compulsory in the file name argument
3 # Row names will be added by default in the file being written
4 # Using the write.csv command
5 # The row names will be numbers starting from 1
6 # Turn off this auto or default naming with row.names = FALSE
7 b = 50:70
8 g = 70:90
9 csvPath="/home/hduser/myR/Program_Data/writethiscsv.csv"
10 write.csv(list(b,g),csvPath)
11 r1 = read.csv(csvPath)
12 fix(r1)

```

To the right of the code is a data editor window titled "R Data Editor" showing a table of data:

	X	X50,70	X70,90
1	1	50	70
2	2	51	71
3	3	52	72
4	4	53	73
5	5	54	74
6	6	55	75
7	7	56	76
8	8	57	77
9	9	58	78
10	10	59	79
11	11	60	80
12	12	61	81
13	13	62	82
14	14	63	83
15	15	64	84
16	16	65	85
17	17	66	86
18	18	67	87
19	19	68	88
20	20	69	89
21	21	70	90
22			
23			
24			

At the bottom of the RStudio interface is a taskbar with icons for various applications.

Figure 64: Write csv using object data

2. From Data Frame

Note: Row names are given otherwise it will be added by default starting from 1.

The screenshot shows the RStudio interface with a terminal window. The terminal window displays R code and its execution results. The code creates a data frame `bg_df` from vectors `b` and `g`, writes it to a CSV file, reads it back, and then fixes it. The R Data Editor panel shows the contents of the CSV file. A terminal window in the background shows the output of the `ifconfig` command.

```

1 # Ignore the column names in a csv file created through R
2 # However you might not be able to ignore the column names at times
3 b = c(50:57,rep(NA,4),71:78)
4 g = c(70:83,rep(NA,2),89:92)
5 bg_df = data.frame(b,g)
6 wpath = "/home/hduser/myR/Program_Data/writethiscsv.csv"
7 rownames(bg_df) = c(101:120)
8 colnames(bg_df) = c("collll1","collll2")
9 write.csv(bg_df,wpath)
10 r1 = read.csv(wpath)
11 fix(r1)
12
9:22 (Top Level) +

```

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link
          Brdaddr:192.168.159.134
          Mask:255.255.255.0
          MTU:1500 Metric:1
          RX packets:103 errors:0 dropped:0 overruns:0 frame:0
          TX packets:102 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14640 (14.6 KB) TX bytes:13200 (13.2 KB)
          Interrupt:15

13:113 (Top Level) +

```

```

> source('~/myR/Programs/q49_write_csv_DF.R')

```

Figure 65: Write Data from Data Frame in .csv file

3. Without Row names

Note: Row name can explicitly remove but column names cannot be deleted with same function. Other operations like extraction-modification-update is more demanding for large size data.

The screenshot shows the RStudio interface with a terminal window. The terminal window displays R code and its execution results. The code is identical to Figure 65, but the `write.csv` function includes `row.names = F` and `col.names = F` parameters. The R Data Editor panel shows the contents of the CSV file, which now has an additional column `var3` filled with empty values. A terminal window in the background shows the output of the `ifconfig` command.

```

1 # Ignore the column names in a csv file created through R
2 # However you might not be able to ignore the column names at times
3 b = c(50:57,rep(NA,4),71:78)
4 g = c(70:83,rep(NA,2),89:92)
5 bg_df = data.frame(b,g)
6 wpath = "/home/hduser/myR/Program_Data/writethiscsv.csv"
7 rownames(bg_df) = c(101:120)
8 colnames(bg_df) = c("collll1","collll2")
9 write.csv(bg_df,wpath, row.names = F, col.names = F)
10 r1 = read.csv(wpath)
11 fix(r1)
12
10:20 (Top Level) +

```

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link
          Brdaddr:192.168.159.134
          Mask:255.255.255.0
          MTU:1500 Metric:1
          RX packets:103 errors:0 dropped:0 overruns:0 frame:0
          TX packets:102 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14640 (14.6 KB) TX bytes:13200 (13.2 KB)
          Interrupt:15

13:113 (Top Level) +

```

```

> source('~/myR/Programs/q49_write_csv_DF.R')

```

b. Import Text File

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Use:

```

read.table(file, header = FALSE, sep = "", quote = "\"\"", dec = ".", row.names,
col.names, as.is = !stringsAsFactors, na.strings = "NA", colClasses = NA,
nrows = -1, skip = 0, check.names = TRUE, fill = !blank.lines.skip, strip.white =
FALSE, blank.lines.skip = TRUE, comment.char = "#", allowEscapes = FALSE,
flush = FALSE, stringsAsFactors = default.stringsAsFactors(), fileEncoding =
"", encoding = "unknown", text)

```

The screenshot shows the RStudio interface. On the left is a tool palette with icons for file operations, browser, R, and other tools. The main workspace contains a script editor with the following code:

```

1 # Tab delimited file
2 # Note: Check the class of t1 and it will be a data frame
3 # stores whole sentence in one row
4 filenamepath = "/home/hduser/myR/Program_Data/hello.txt"
5 t1 = read.table(filenamepath, sep = "\t")
6 fix(t1)
7 print(class(t1))

```

Below the script editor is a terminal window showing the command `ifconfig` output:

```

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:L

```

To the right is the R Data Editor pane, which lists 24 items under the variable `Cons`:

- > s e_i
- 1 Every moment matters
- 2 Do small things with great love
- 3 Never stop dreaming
- 4 Conquer from within
- 5 Hold on, pain ends
- 6 Take risk, or loose
- 7 Worry less, smile more
- 8 Find a way or fade away
- 9 Focus on the good
- 10 Be your own kind of beautiful
- 11 Weakness is a choice
- 12 Do what you love
- 13 Small steps every day
- 14 Celebrate small victories
- 15 Choose kindness and laugh often
- 16 Be a game changer
- 17 Life is tough but so are you
- 18 Let your light shine
- 19 No rain, no flower
- 20 Be here now
- 21
- 22
- 23
- 24

The status bar at the bottom right shows the date and time: 29-07-2017 02:35.

Figure 66: q56 - Read Text File

c. Import XML file

This function can be used to extract data from an XML document (or sub-document) that has a simple, shallow structure that does appear reasonably commonly. The idea is that there is a collection of nodes which have the same fields (or a subset of common fields) which contain primitive values, i.e. numbers, strings, etc. Each node corresponds to an "observation" and each of its sub-elements correspond to a variable.

`xmlToDataFrame()` function then builds the corresponding data frame, using the union of the variables in the different observation nodes. This can handle the case where the nodes do not all have all of the variables.

Use:

```

xmlToDataFrame(doc, colClasses = NULL, homogeneous = NA, collectNames
= TRUE, nodes = list(), stringsAsFactors = default.stringsAsFactors())

```

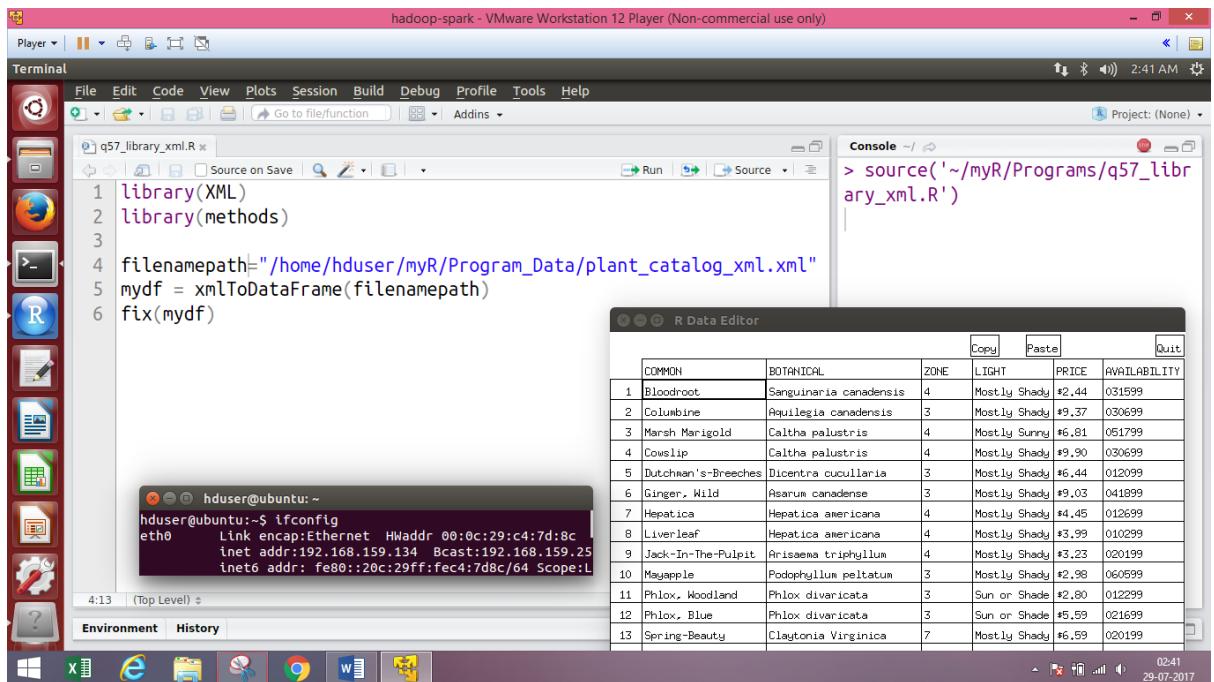


Figure 67: Import .XML file

13. Write A Program Questions:

i. WAP to print the first 30 odd numbers

```

vect = 3:60
ans_vect = c(1)
for(i in vect){ #1
  if(i%%2 != 0){ #2
    ans_vect = append(ans_vect,i)
  } #1.1 if
} #1 for
print(ans_vect)

```

```

hadoop-spark - VMware Workstation 12 Player (Non-commercial use only)
Player Terminal File Edit Code View Plots Session Build Debug Profile Tools Help
q3.R wap0_for.R Source on Save Go to file/function Run Source
1 #WAP0: To print the first 30 odd numbers
2
3 vect = 3:60
4 ans_vect = c(1)
5 for(i in vect){ #1
6   if(i%%2 != 0){ #2
7     ans_vect = append(ans_vect,i)
8   } #1.1 if
9 } #1 for|
10 print(ans_vect)

Console ~ / 
> source('~/myR/Programs/wap0_for.R')
[1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27
[15] 29 31 33 35 37 39 41 43 45 47 49 51 53 55
[29] 57 59
> |
```

hduser@ubuntu:~\$ ifconfig
eth0 Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
inet addr:192.168.159.134 Bcast:192.168.159.255 Mask:255.255.255.0
inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link

Figure 68: WAP 0 : For Loop

ii. Modify the repeat code to make infinite loop

```
i = 0
repeat{ #1
  cat(i, " ")
  i = i+1
} #repeat
```

```

hadoop-spark - VMware Workstation 12 Player (Non-commercial use only)
Player Terminal File Edit Code View Plots Session Build Debug Profile Tools Help
q3.R wap1.repeat.R Source on Save Go to file/function Run Source
1 #WAP1: Modify the repeat code to make infinite loop
2 i = 0
3 repeat{ #1
4   cat(i, " ")
5   i = i+1
6 } #repeat

Console ~ / 
53 54 55 56 57 58 59 60 61 62 63 64
65 66 67 68 69 70 71 72 73 74 75 76 77
78 79 80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99 100 101
102 103 104 105 106 107 108 109 110 111
112 113 114 115 116 117 118 119 120 121
122 123 124 125 126 127 128 129 130 131
132 133 134 135 136 137 138 139 140 141
142 143 144 145 146 147 148 149 150 151
152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171
172 173 174 175 176 177 178 179 180 181
182 183 184 185 186 187 188 189 190 191
192 193 194 195 196 197 198 199 200 201
202 203 204 205 206 207 208 209 210 211
212 213 214 215 216 217 218 219 220 221
... <truncated>
```

hduser@ubuntu:~\$ ifconfig
eth0 Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
inet addr:192.168.159.134 Bcast:192.168.159.255
inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link

Figure 69: WAP 1 Repeat Loop

iii. WAP to modify the above code to print the first 20 numbers divisible by 5

```
i = 0
```

```

ans_vect = c(i)
counter = 1
repeat{ #1
  i = i+1
  if(counter == 20){ #1.1
    break
  } #1.1 if
  if(i %% 5 == 0){ #1.2
    counter = counter + 1
    ans_vect = append(ans_vect,i)
  } #1.2 if
} #1 repeat
print(ans_vect)

```

The screenshot shows the RStudio interface with the R script `wap2_repeat.R` open. The code defines a function that prints the first 20 natural numbers, skipping every 5th number. The console output shows the resulting vector. Below the RStudio window, a terminal window displays the output of the `ifconfig` command.

Figure 70: WAP 2 Repeat

- iv. Write a function to print the squares of the first N natural numbers, where N would be the argument the user passes to the function.

```

square = function(a,n){ #1 function
  if(mode(a) == "numeric"){ #1.1
    i = 1 #pointer for vector
    cat("\n Squares of first ", n,
        values: \n")
    while(i<=n){ #1.1.1
      cat(i*i, " ")
      i = i+1
    } #1.1.1 while
  } #1.1 if
} #1 End Function
vect_input = c(1:15)

```

```
#print(mode(vect_input))
cat("Input vector: \n",vect_input)
square(vect_input,10)
```

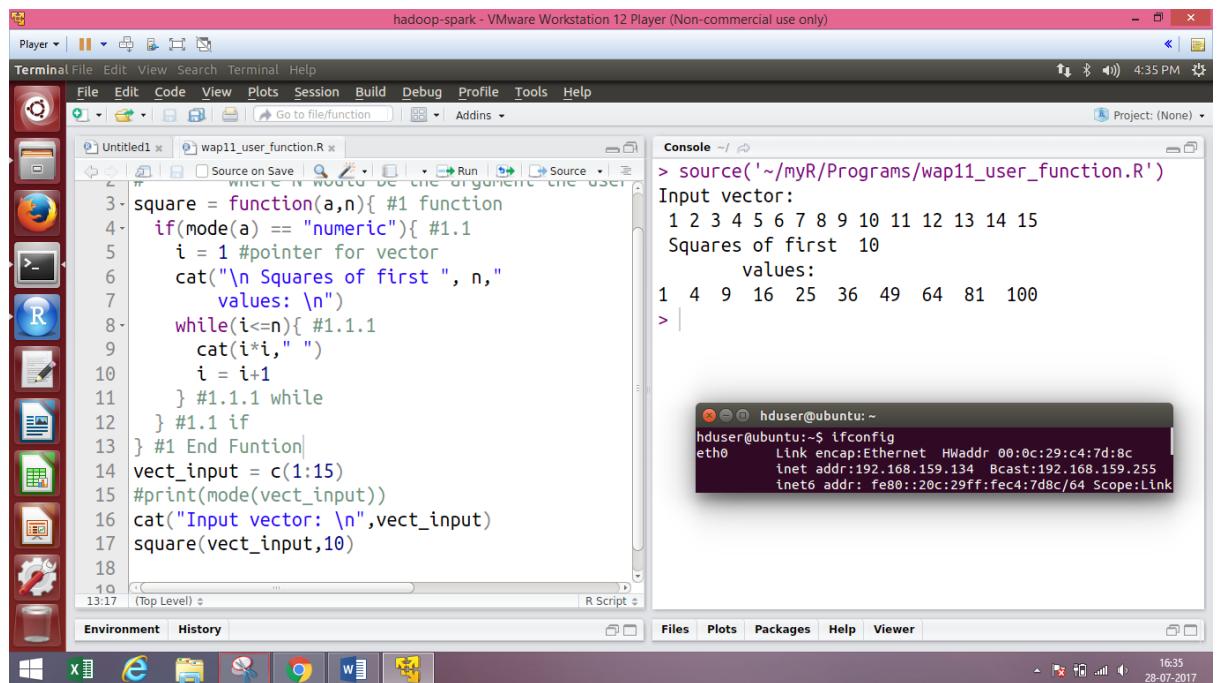


Figure 71: WAP 11 Function

- v. **WAP where a vector with 10 numbers is passed on to a user defined function, increment every number in the vector and return it back to display the contents.**

```
increaseVector = function(v,x)
{
  #cat("vector: ",vect,"\\n")
  ans_vect = vect + x
}

vect = sample(30:100, size = 10)
cat("vector: ",vect,"\\n")
cat(increaseVector(vect,5))
```

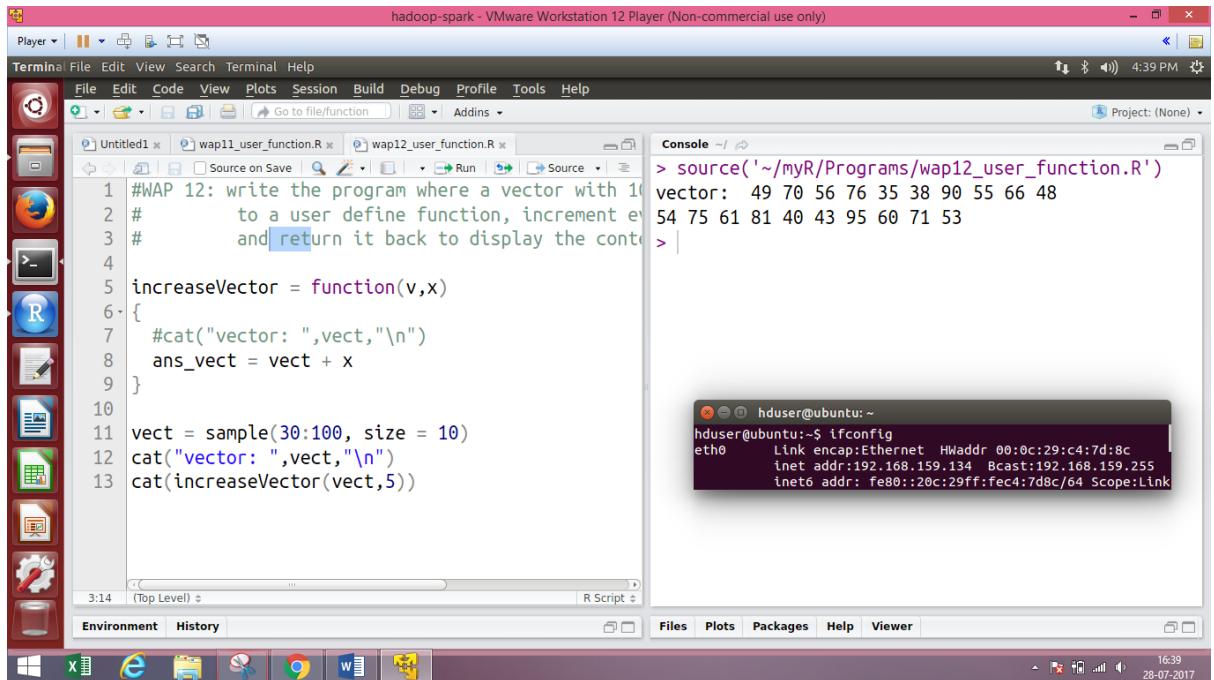


Figure 72: WAP 12 function

- vi. To concatenate 2 strings by passing the strings as an argument to a function and then returning it back from the function and display the result

```
conc = function(s1,s2)
{
  ans = paste(s1,s2)
}
str1 = "Gazal"
str2 = "Patel"
cat(conc(str1,str2))
```

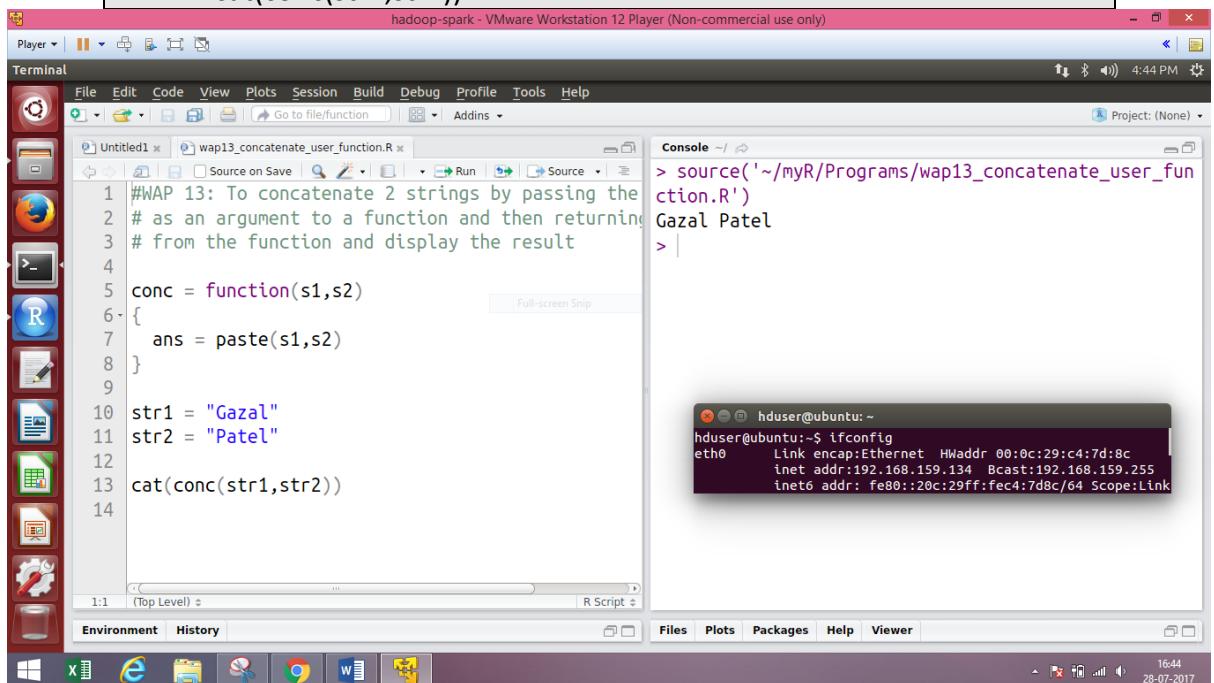


Figure 73: WAP 13 function

vii. **WAP to remove space from string and measure length of it**

```
library(stringr)

str = "Hello how are you"
#print(str)
#print(gsub("[[:space:]]", "$", str))
#m = regexpr(pattern = ex, text = str)

#METHOD1
str_wo_space = gsub("[[:space:]]", "", str)
cat("METHOD1: ",str_wo_space)
cat("\nlength of string without space is: "
    ,nchar(str_wo_space))

#METHOD2
str_wo_space1 = str_replace_all(string=a, pattern=" ", repl="")

cat("\n METHOD2: ",str_wo_space1)
cat("\nlength of string without space is: "
    ,nchar(str_wo_space1))
```

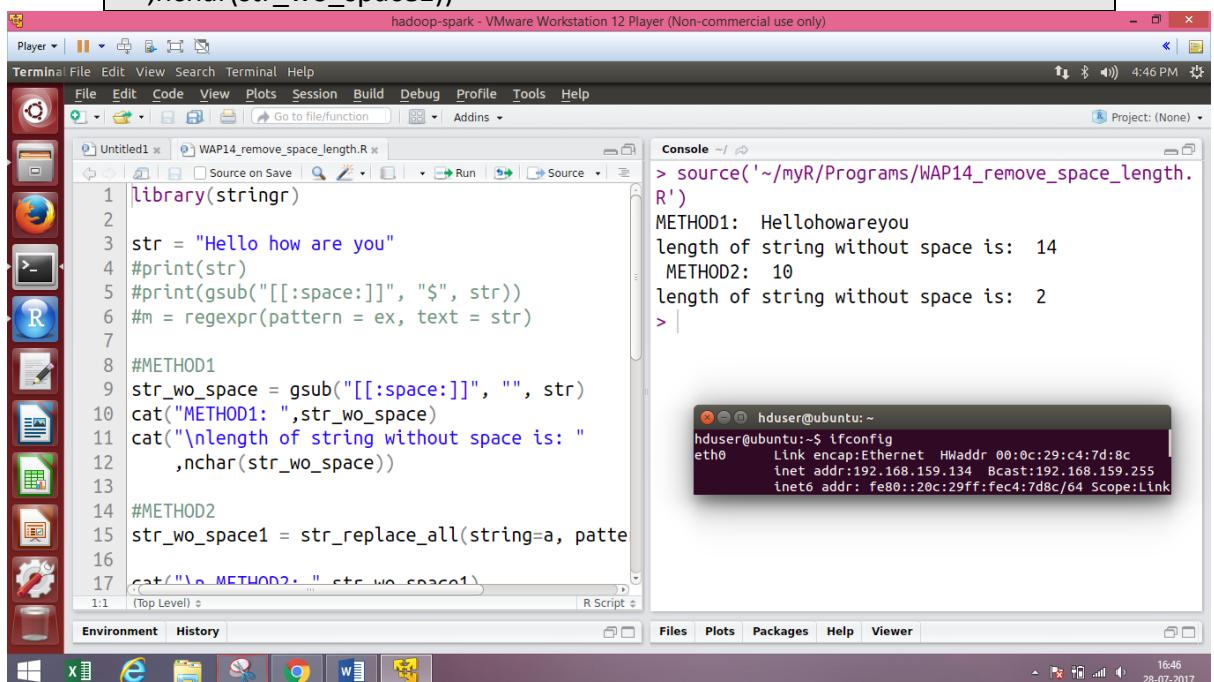


Figure 74: WAP 14 String

viii. **WAP to count space in string.**

```
CountOccurrence = function(str, sub_str)
{
  # gsub(x,y,z)
  # gsub replaces string x to string y in string z
  x <- gsub(sub_str,"",str)
  numof <- nchar(str) - nchar(x)
```

```

    return(numof)
}

s1 <- "blabla 23 mai 2000 blabla 18 mai 2004"
p1 <- "0"

ans1 =CountOccurrence(s1,p1)
cat("\nnumber of times ",p1," is repeated in ",s1," is: ",ans1)
s2 <- "How are you"
p2 <- " "

cat("\nnumber of times ",p2," is repeated in ",s2," is:
",CountOccurrence(s2,p2))

```

The screenshot shows the RStudio interface with the following details:

- Code Editor:** Displays the R script `wap15_string_gsub_count.R` containing the provided R code.
- Console:** Shows the command `source('~/myR/Programs/wap15_string_gsub_count.R')` followed by the output of the script execution. The output includes the counts of '0' and '' in the respective strings.
- Terminal:** A separate terminal window titled 'hduser@ubuntu:' shows the command `ifconfig` and its output, including network interface details like HWaddr and inet addresses.
- System Tray:** Shows the date and time as 28-07-2017 at 16:50.

Figure 75: WAP 15 String count

ix. A program to demonstrate a function calling another function

```

Find_VolumeOfCube = function(len)
{
  return(len*Find_AreaOfCube(len))
}
Find_AreaOfCube = function(len)
{
  return(len*len)
}

cubeLen1 = 5
cat("\nVolume of e= ",cubeLen1," is:
",Find_VolumeOfCube(cubeLen1))

```

```

cubeLen2 = 3
cat("\nVolume of e= ",cubeLen2," is:
",Find_VolumeOfCube(cubeLen2))
cubeLen3 = 4
cat("\nArea of e= ",cubeLen3," is: ",Find_AreaOfCube(cubeLen3))

```

The screenshot shows the RStudio interface. The top bar indicates it's running on a VMware Workstation 12 Player. The terminal window on the right displays the execution of R code and an ifconfig command. The R script editor on the left shows the source code for a program demonstrating function calling.

```

> source('~/myR/Programs/WAP16_inside_function_calling.R')

Volume of e= 5 is: 125
Volume of e= 3 is: 27
Area of e= 4 is: 16
>

hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fec4:7d8c/64 Scope:Link

```

Figure 76: WAP 16 inside Function Calling

x. WAP to pick only the elements from vector which divisible by 5 and 7

```
vect = 1:1000
```

```
ans_vect = subset(vect, vect%%5 == 0 & vect%%7 == 0)
```

```
print(ans_vect)
```

The screenshot shows the RStudio interface running on a Windows host (VMware Workstation 12 Player). The left sidebar contains icons for various applications like R, R Markdown, and GitHub. The main workspace has three tabs open: 'Untitled1', 'WAP16_inside_function_calling.R', and 'wap17_subset.R'. The 'wap17_subset.R' tab contains the following R code:

```
1 # WAP to pick only the elements from
2 # vector which divisible by 5 and 7
3
4 vect = 1:1000
5
6 ans_vect = subset(vect, vect%%5 == 0 & vect%%7 ==
7
8 print(ans_vect)
```

The 'Console' tab shows the output of the R script:

```
> source('~/myR/Programs/wap17_subset.R')
[1] 35 70 105 140 175 210 245 280 315 350 385
[12] 420 455 490 525 560 595 630 665 700 735 770
[23] 805 840 875 910 945 980
> |
```

Below the RStudio window, a terminal window is visible, showing the output of the 'ifconfig' command on an Ubuntu system:

```
hduser@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 00:0c:29:c4:7d:8c
          inet addr:192.168.159.134 Bcast:192.168.159.255
          inet6 addr: fe80::20c:29ff:fe:c47d%eth0/64 Scope:Link
```

The taskbar at the bottom shows icons for various Windows applications.

Figure 77: WAP 17 subset