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.

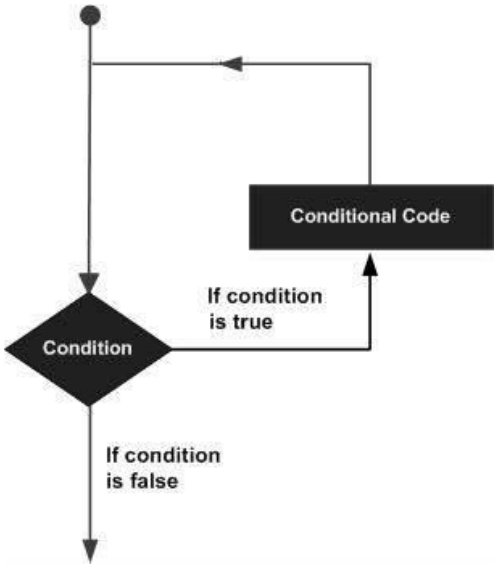


Figure 1: Loop Structure Flow Chart

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

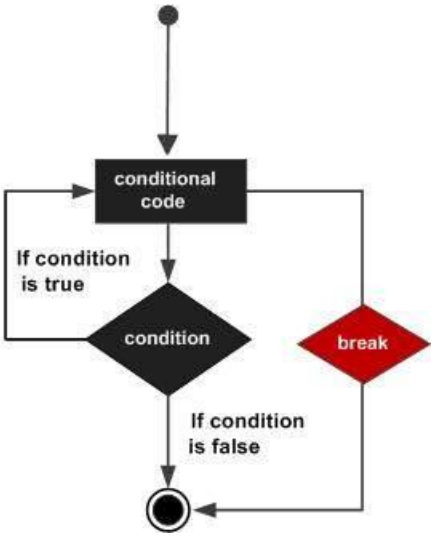


Figure 2: Repeat Loop Flow Chart

Use:

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

```
    }
}
```

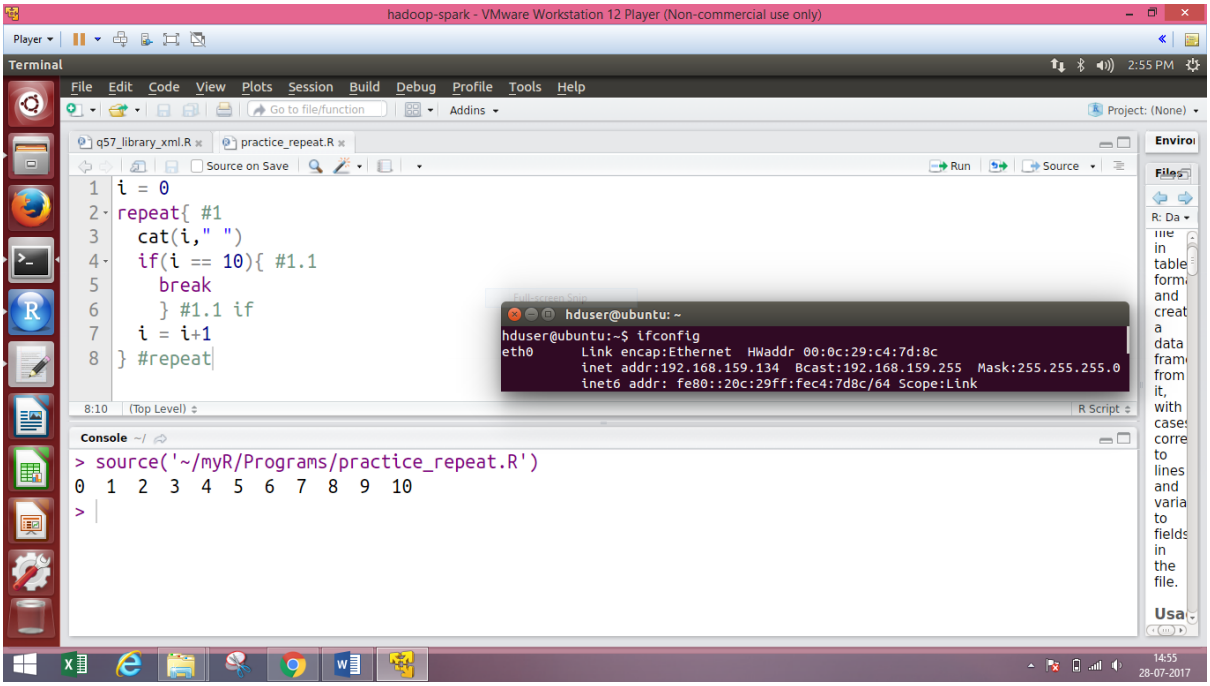


Figure 3: Repeat Loop Program

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

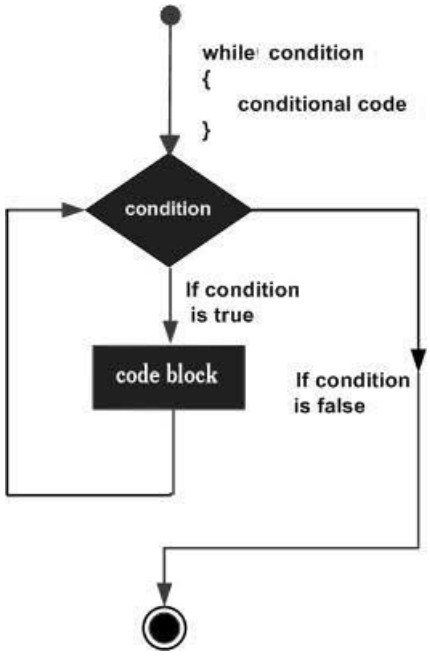


Figure 4: While Loop Flow Chart

Use:

```
while (test_expression) {
  statement
}
```

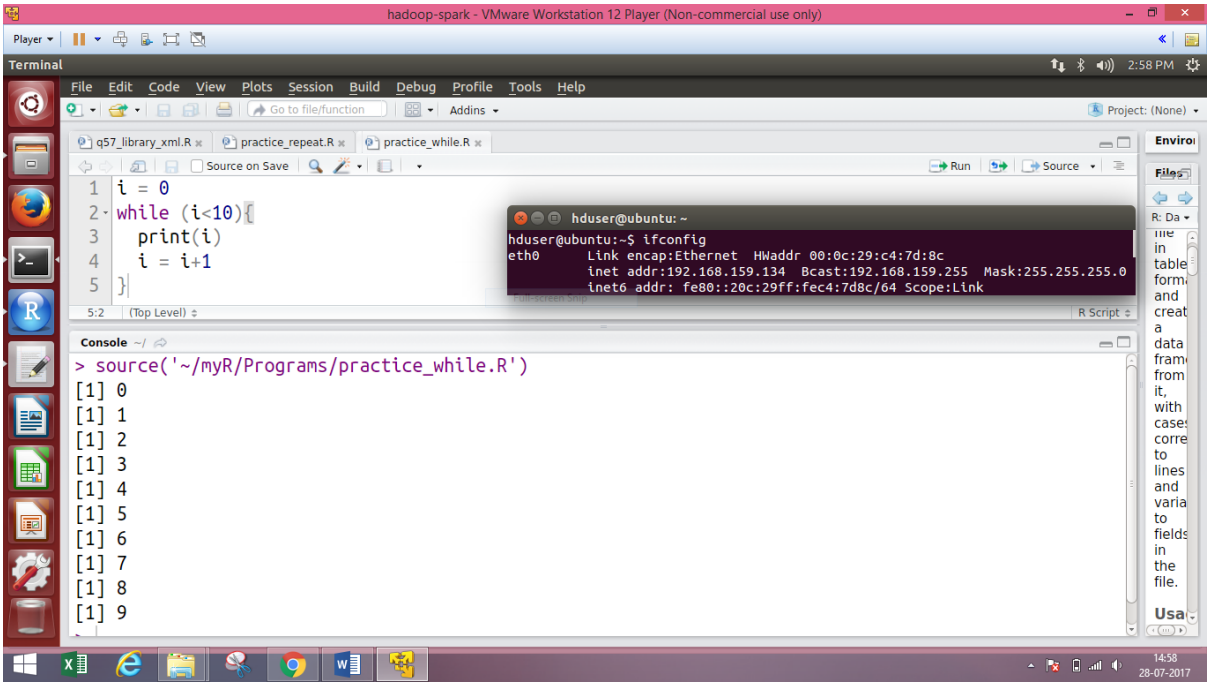


Figure 5: While Loop Program

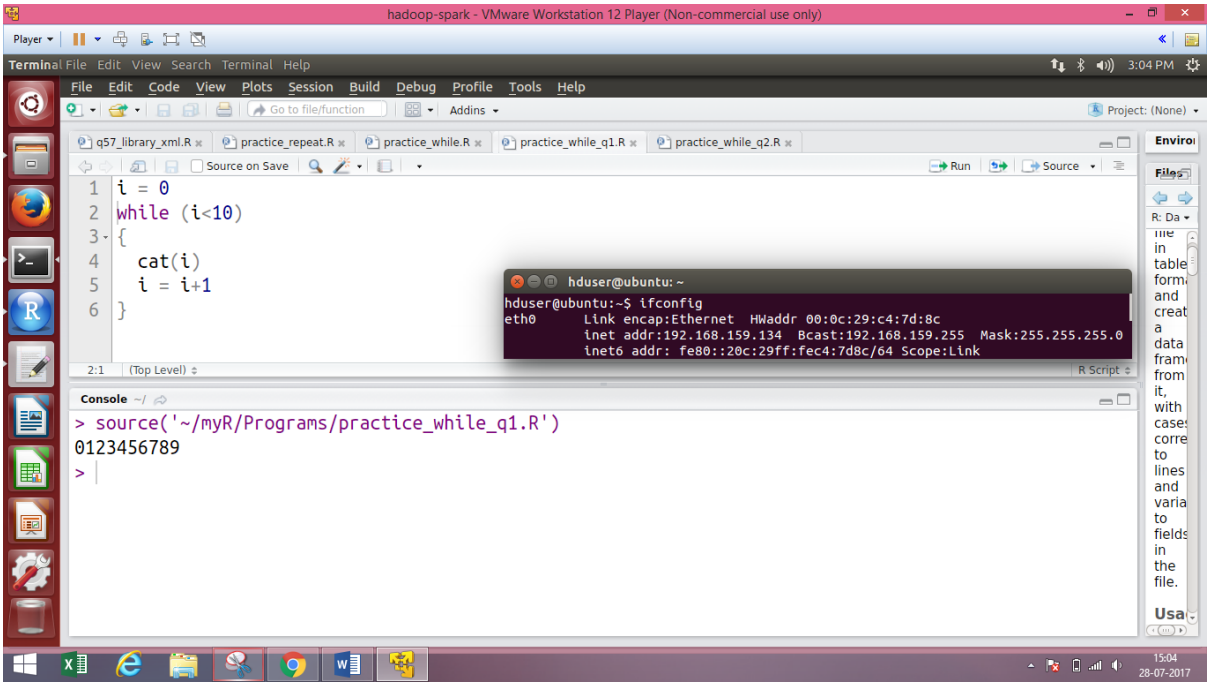


Figure 6: q1 - While Loop Cat Program

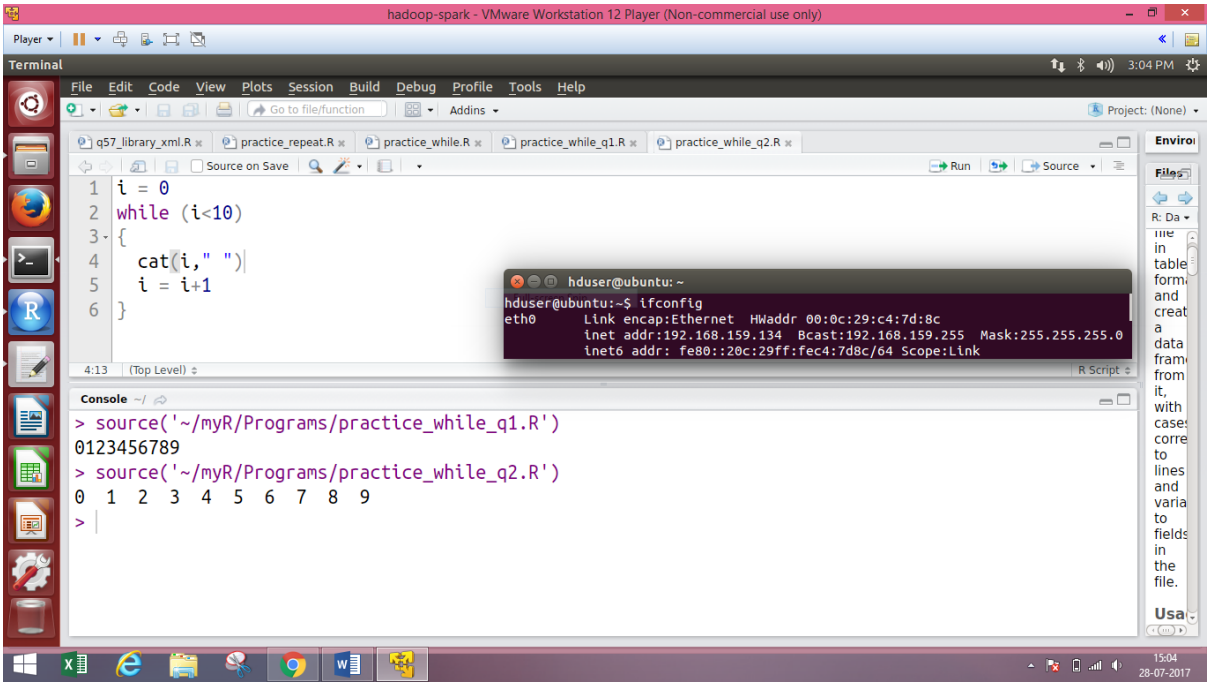


Figure 7: q2 - While Loop Program

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
}
```

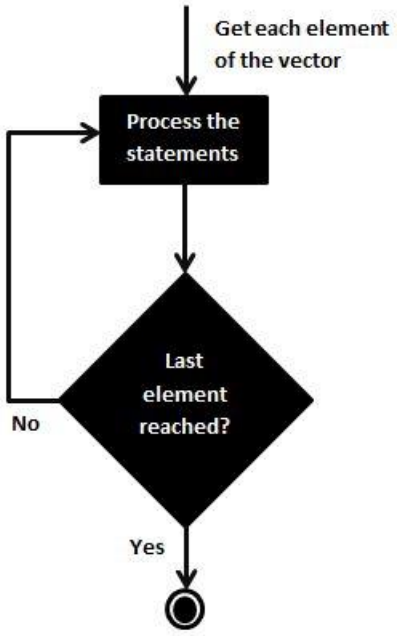


Figure 8: For Loop Flow Chart

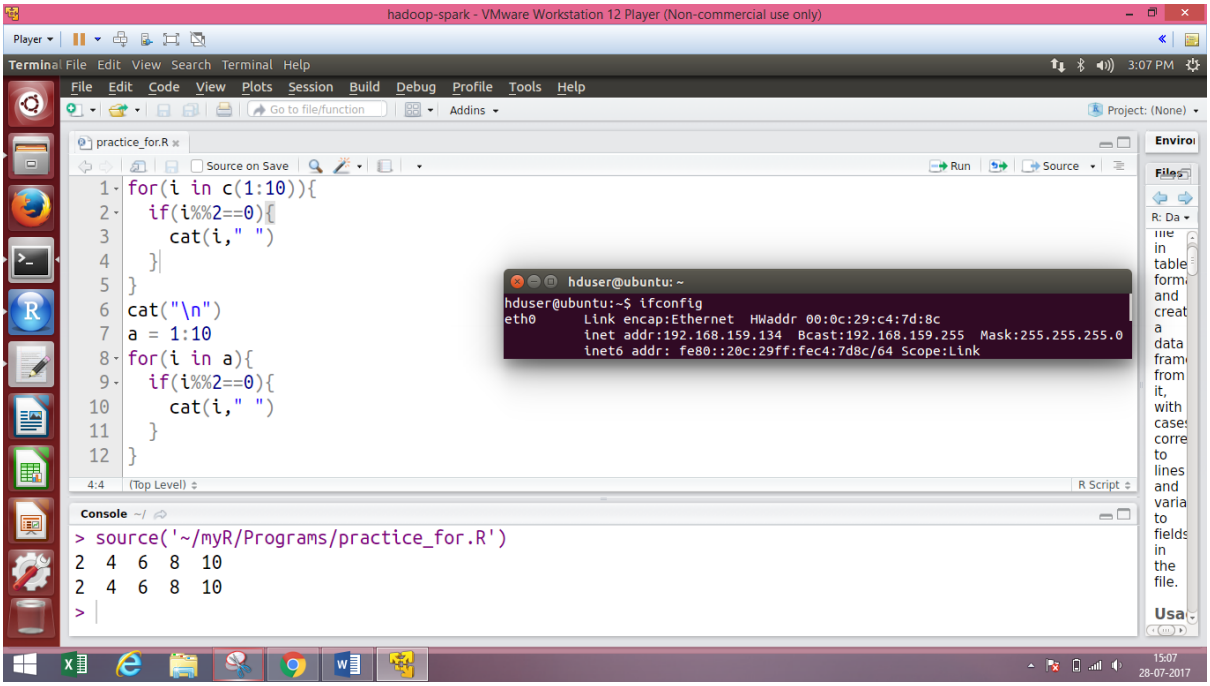


Figure 9: For Loop Program

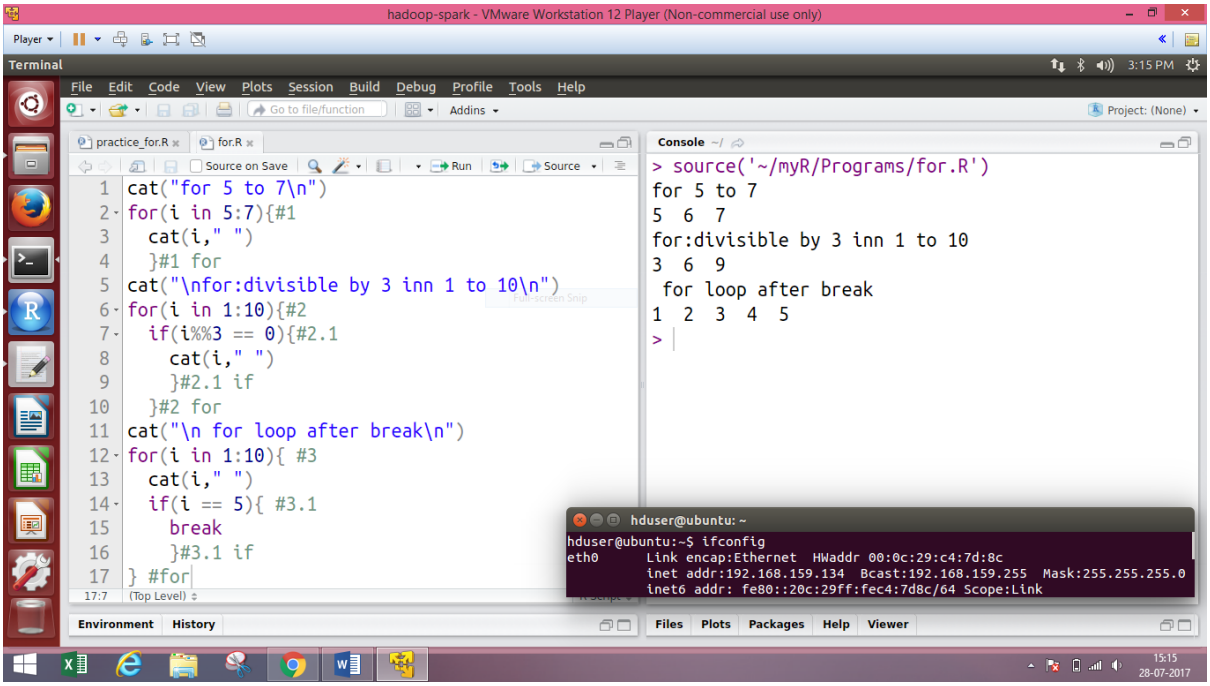


Figure 10: For Loop Program and Break

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;
```

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

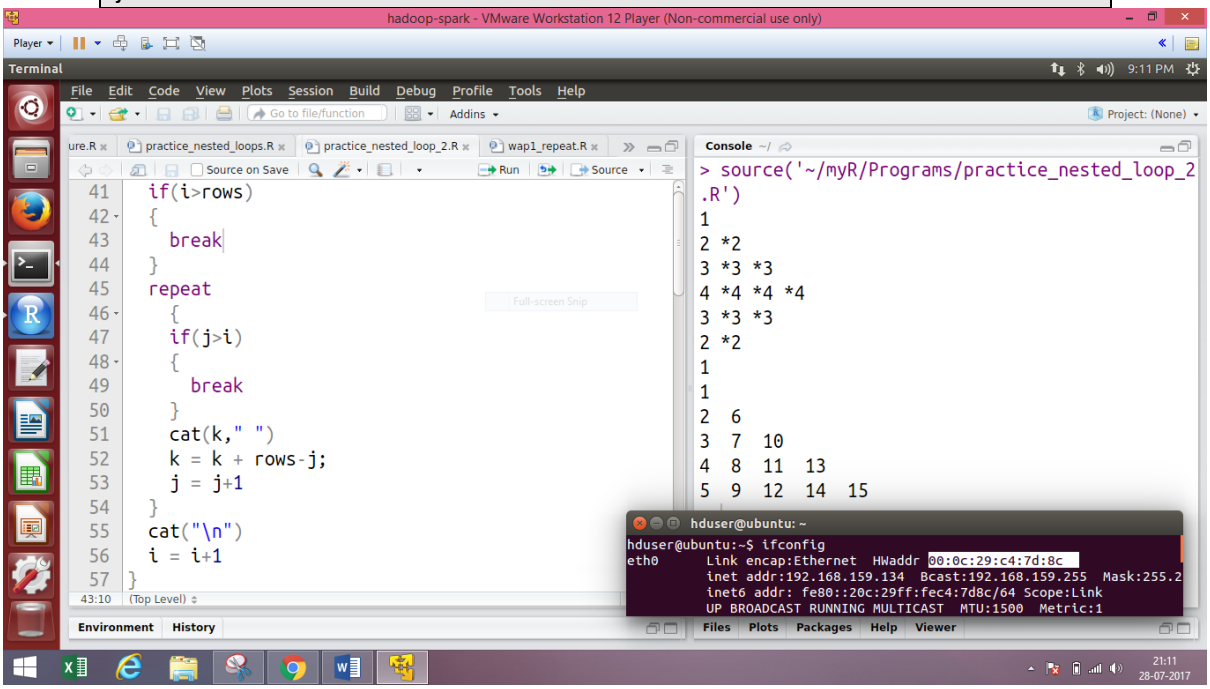


Figure 11: Nested Loop with break

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.

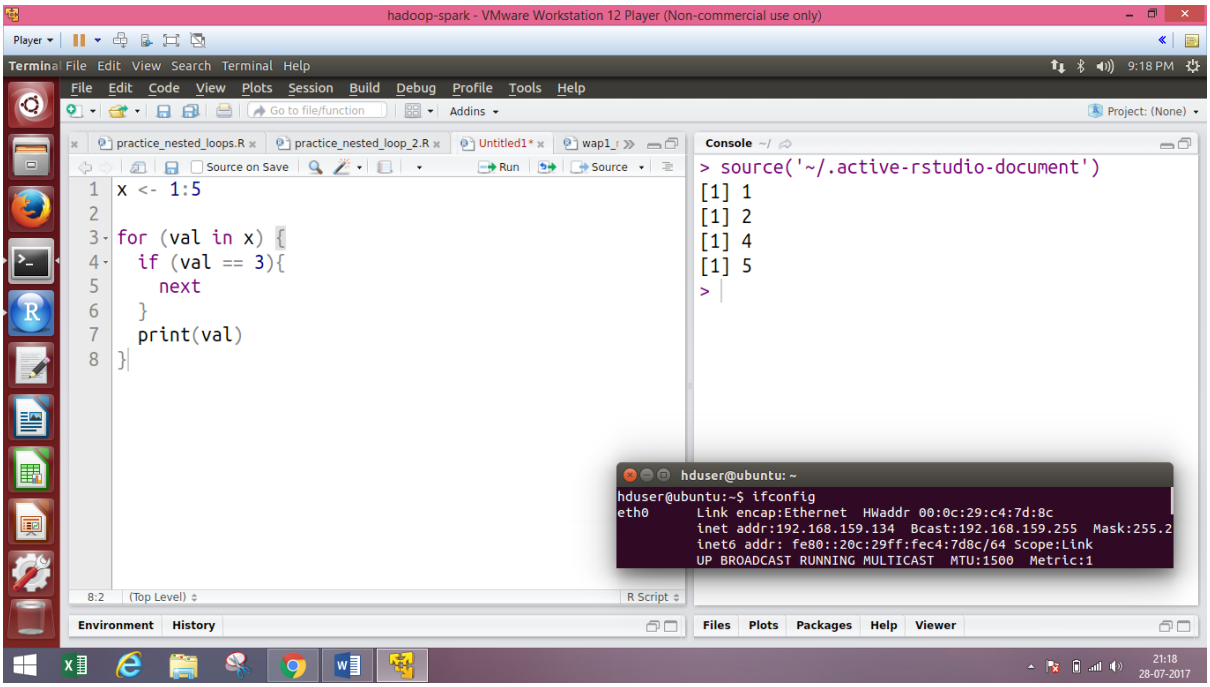


Figure 12: Next in Loops

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.

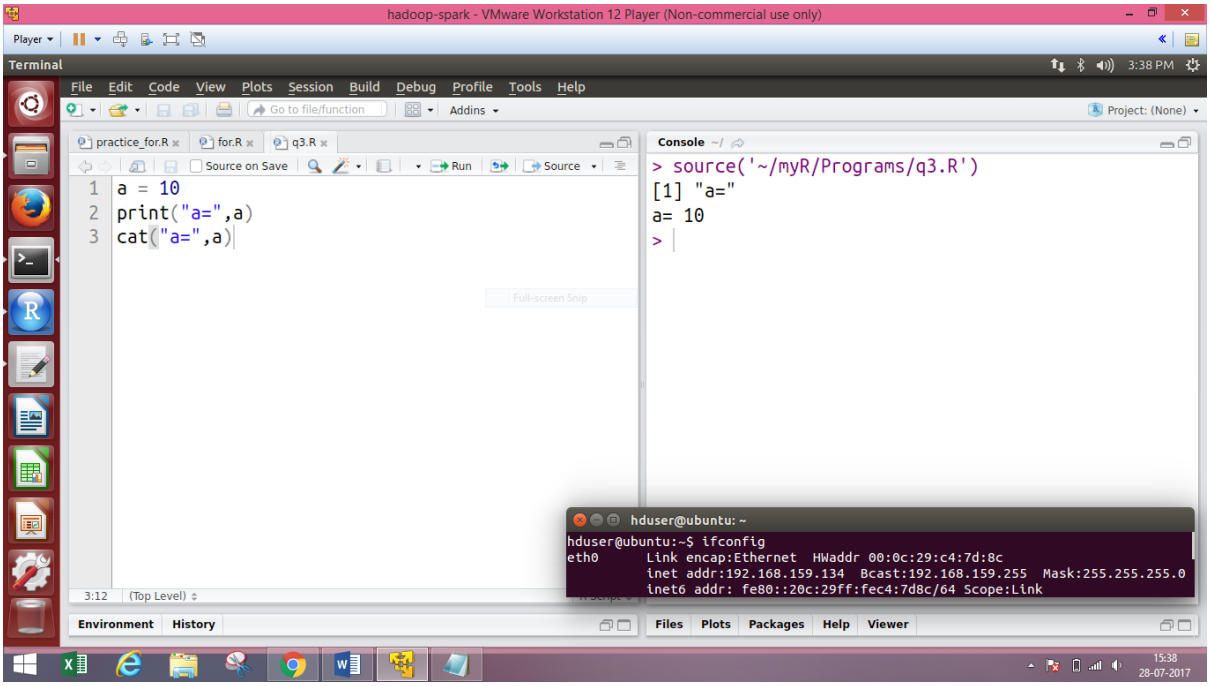


Figure 13: q3 - cat() and print() functions

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

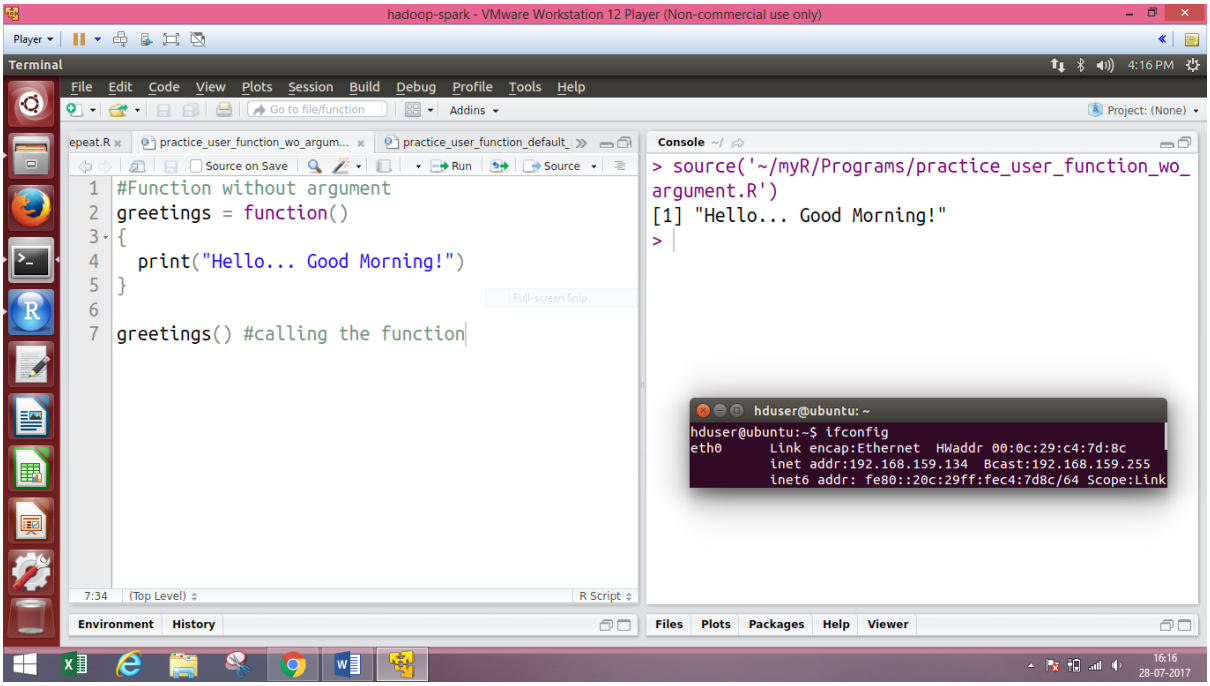


Figure 14: Calling a Function without an Argument

b. Function with Parameter

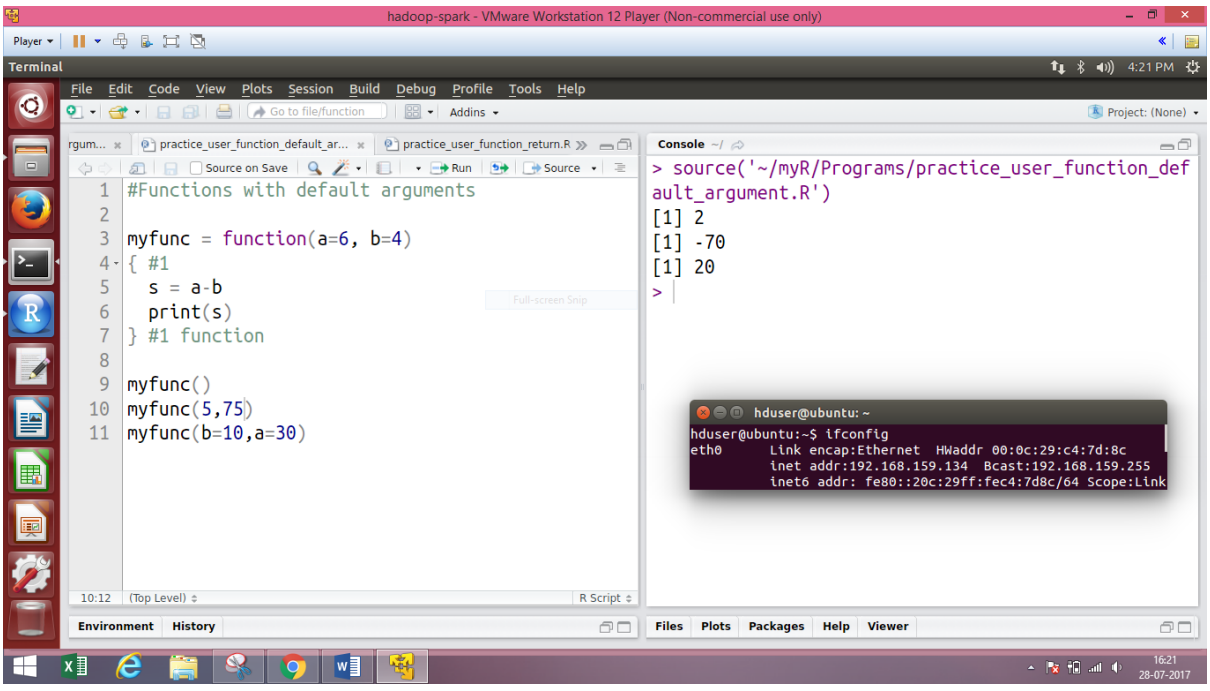


Figure 15: Calling a Function with Argument Values (by position and by name)

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**

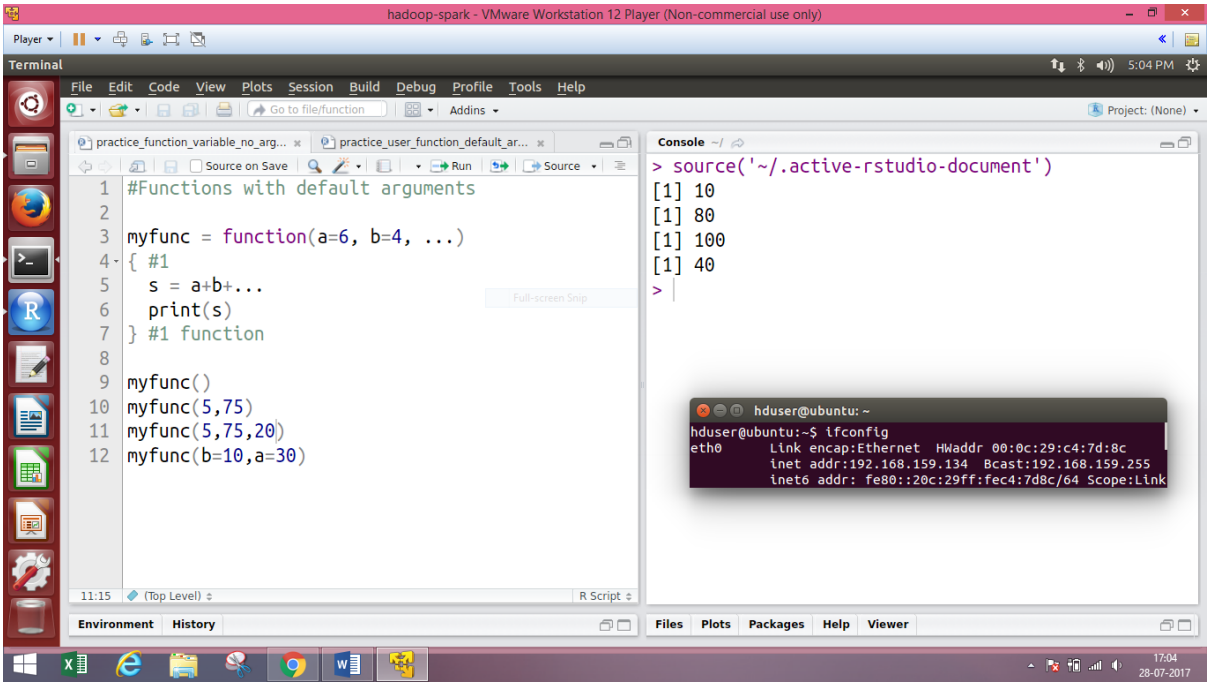


Figure 16: variable function parameter

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.

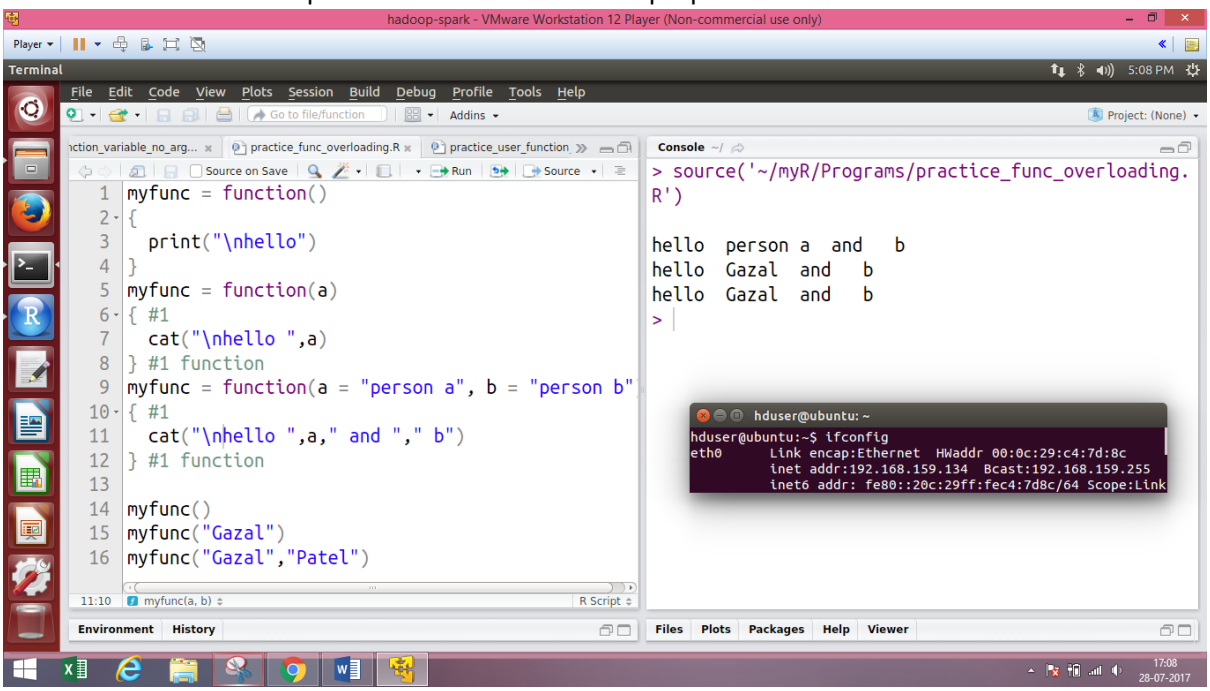


Figure 17: Function Overloading

c. Return from function

i. return() function

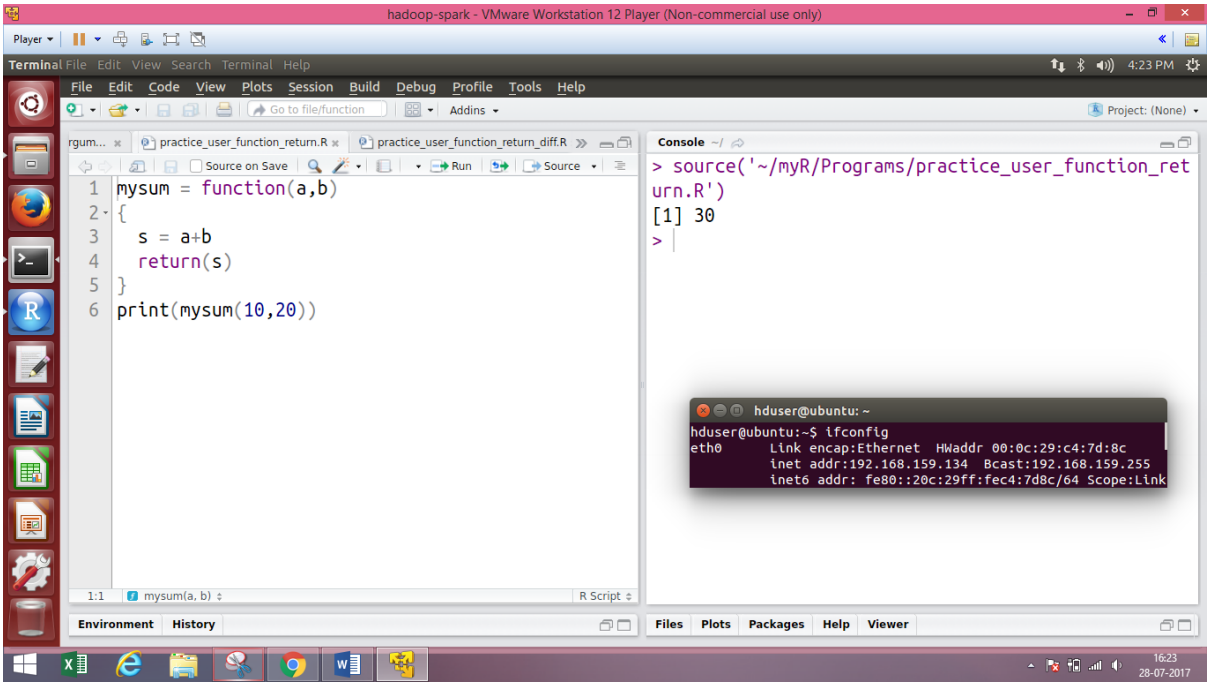


Figure 18: Function with return() funciton

ii. Auto return

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

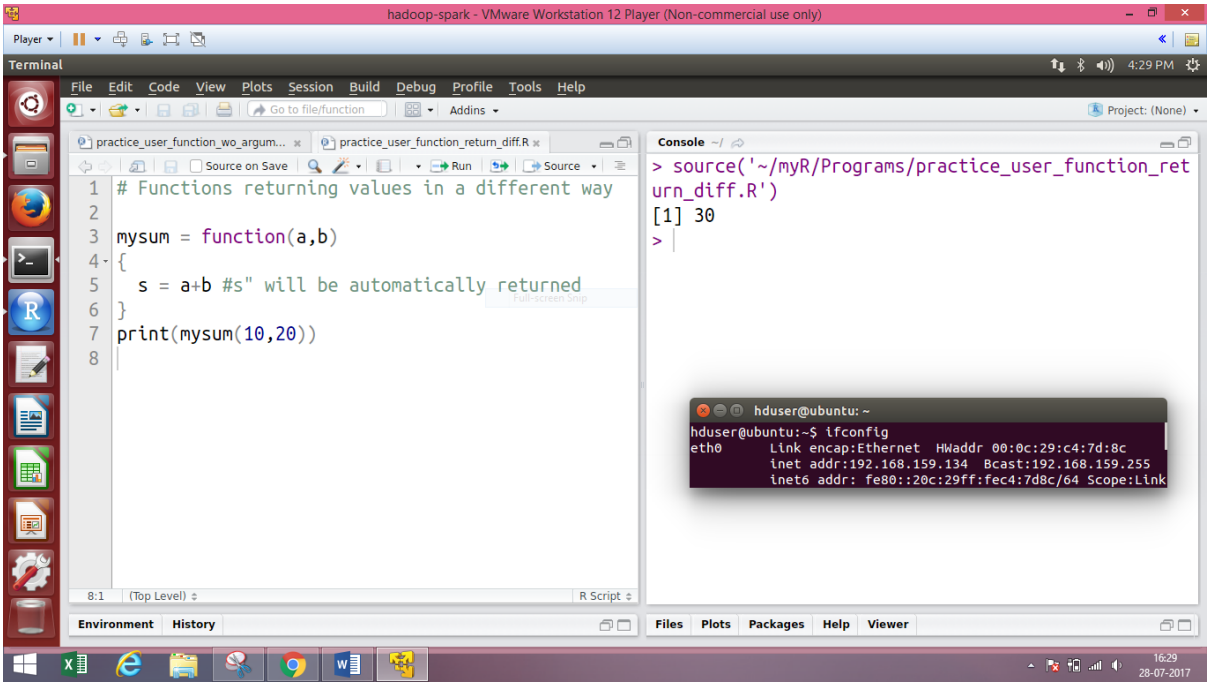


Figure 19: Default function return

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)

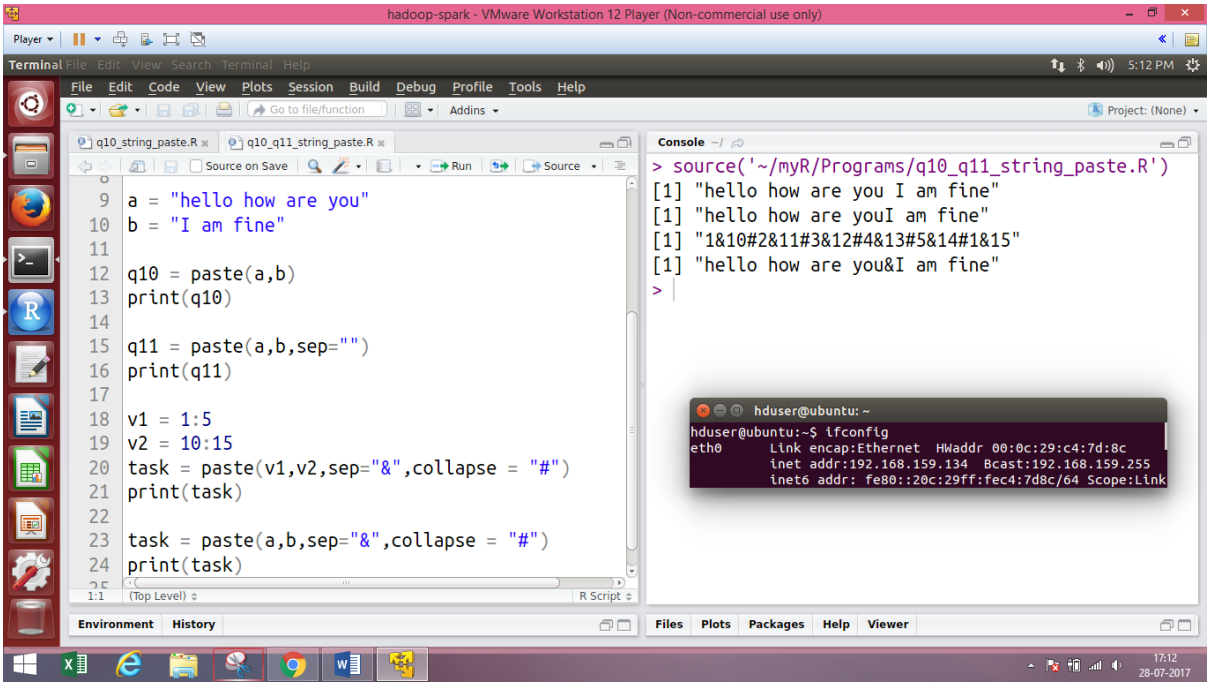


Figure 20: q10 and q11 - string paste

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)

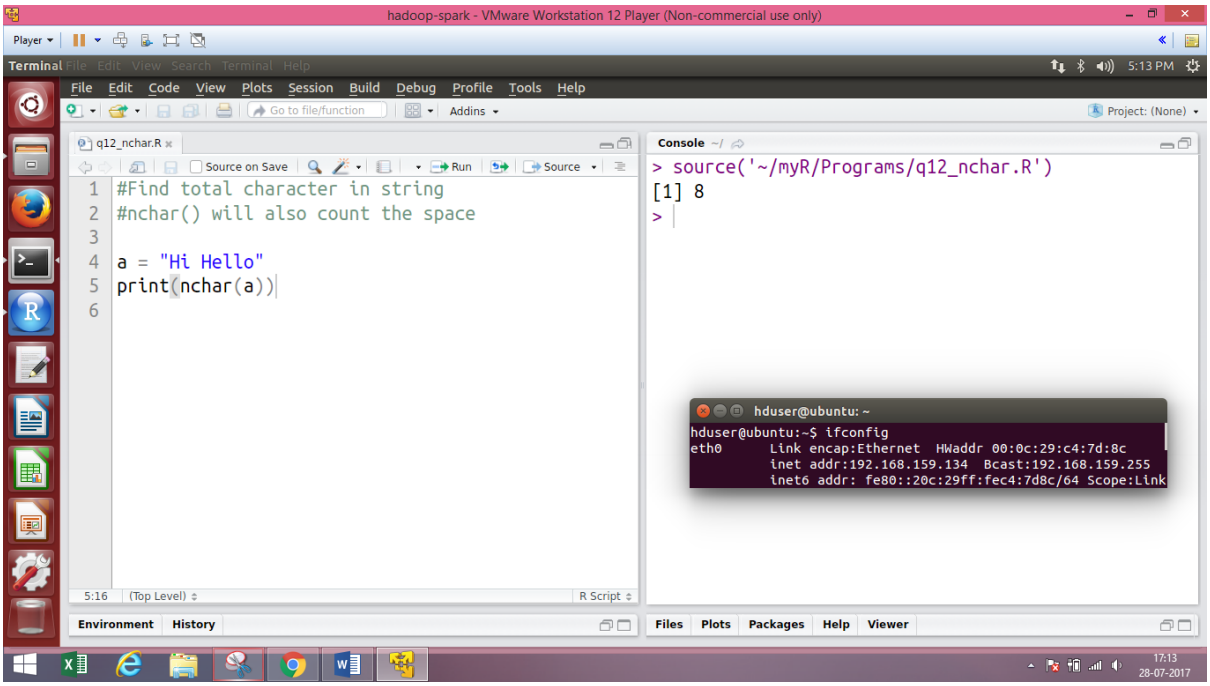


Figure 21: q12 - nchar() function

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)

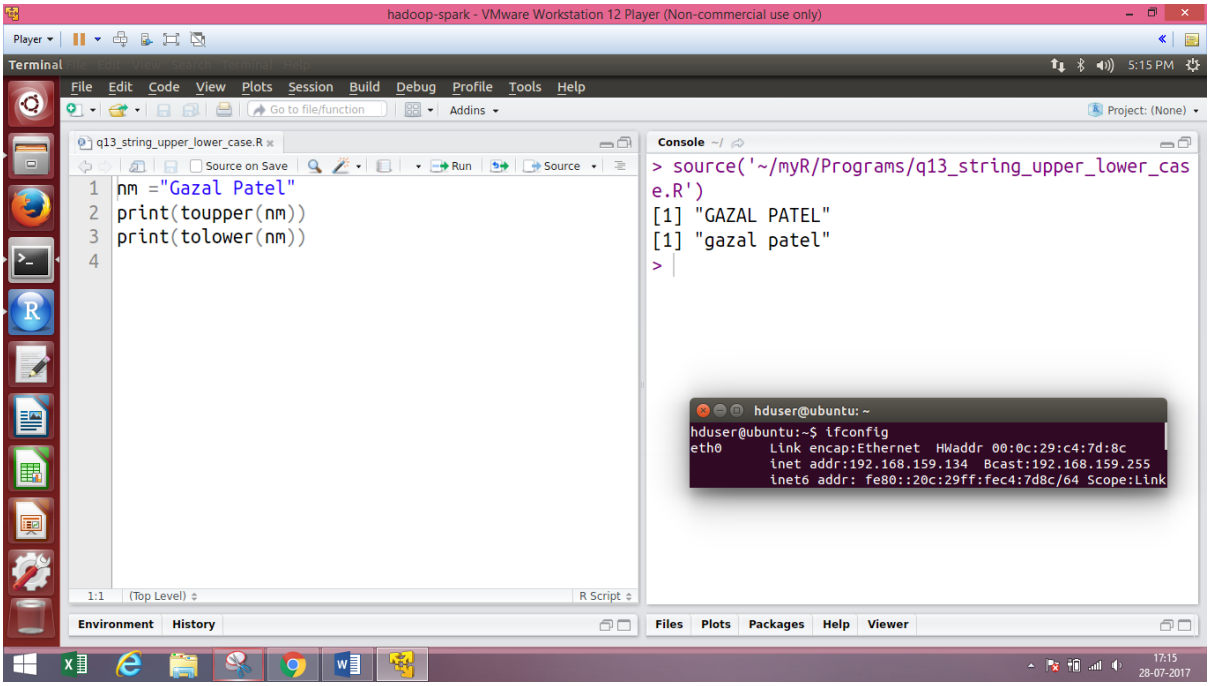


Figure 22: q13 - upper and lower case string

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)

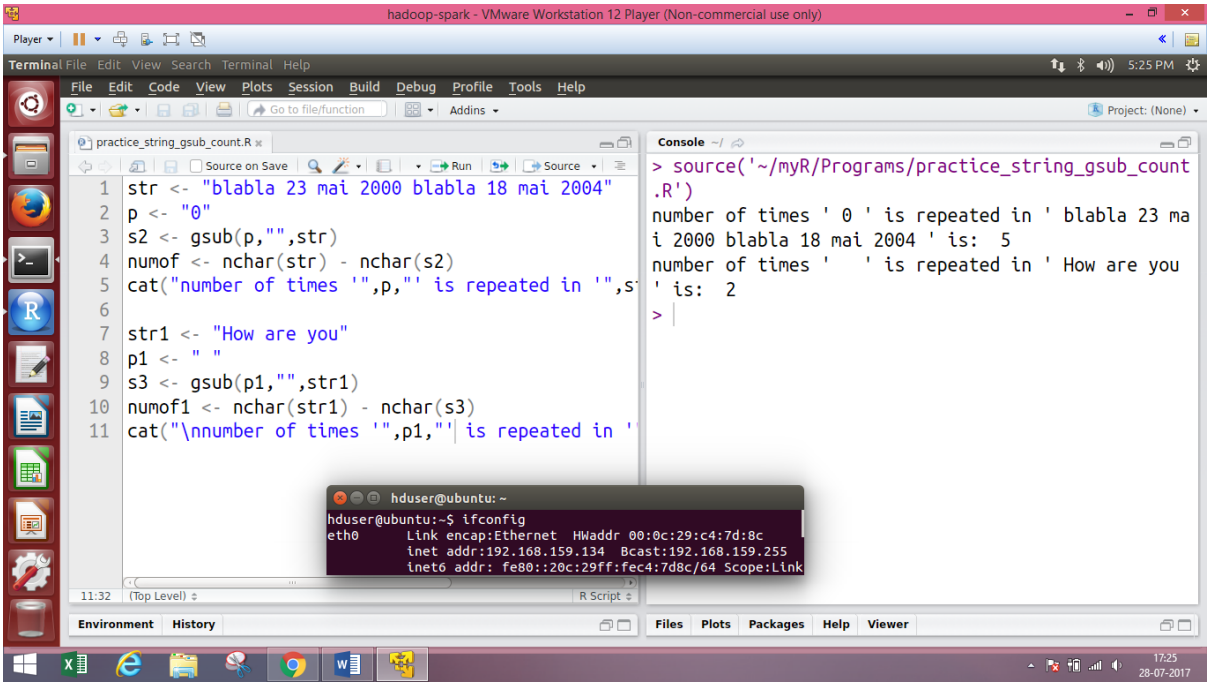


Figure 23: q14 - sub() and gsub() functions

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)

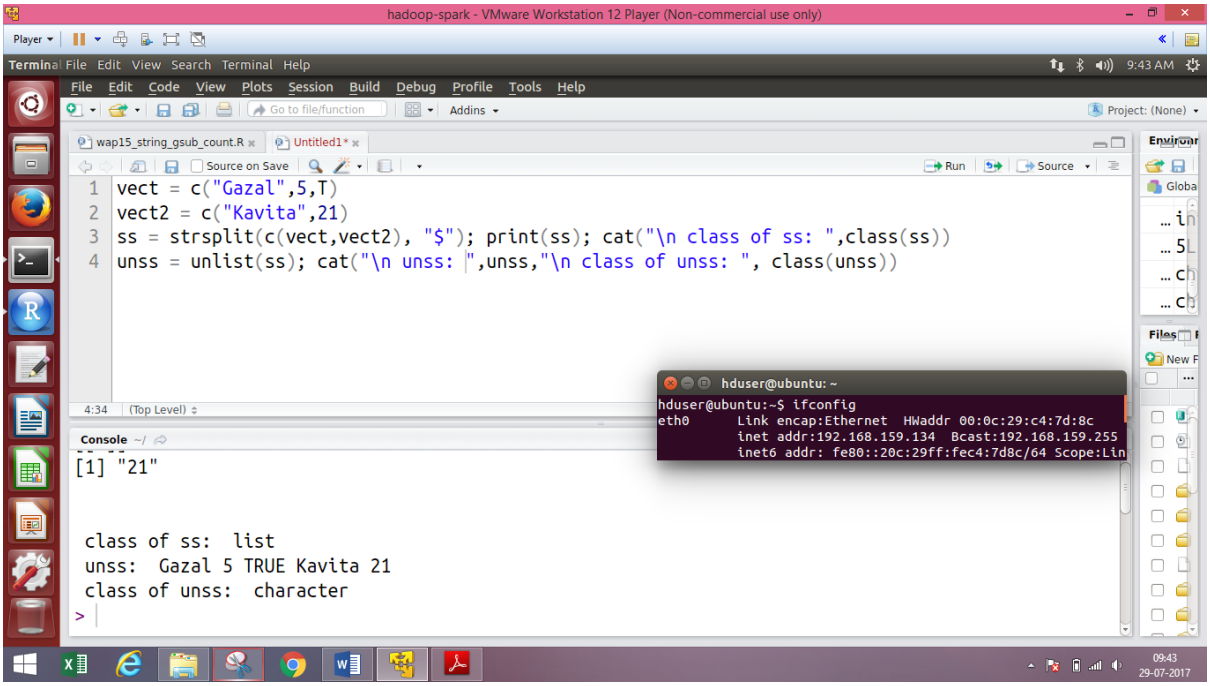


Figure 24: strsplit() and unlist() functions

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)

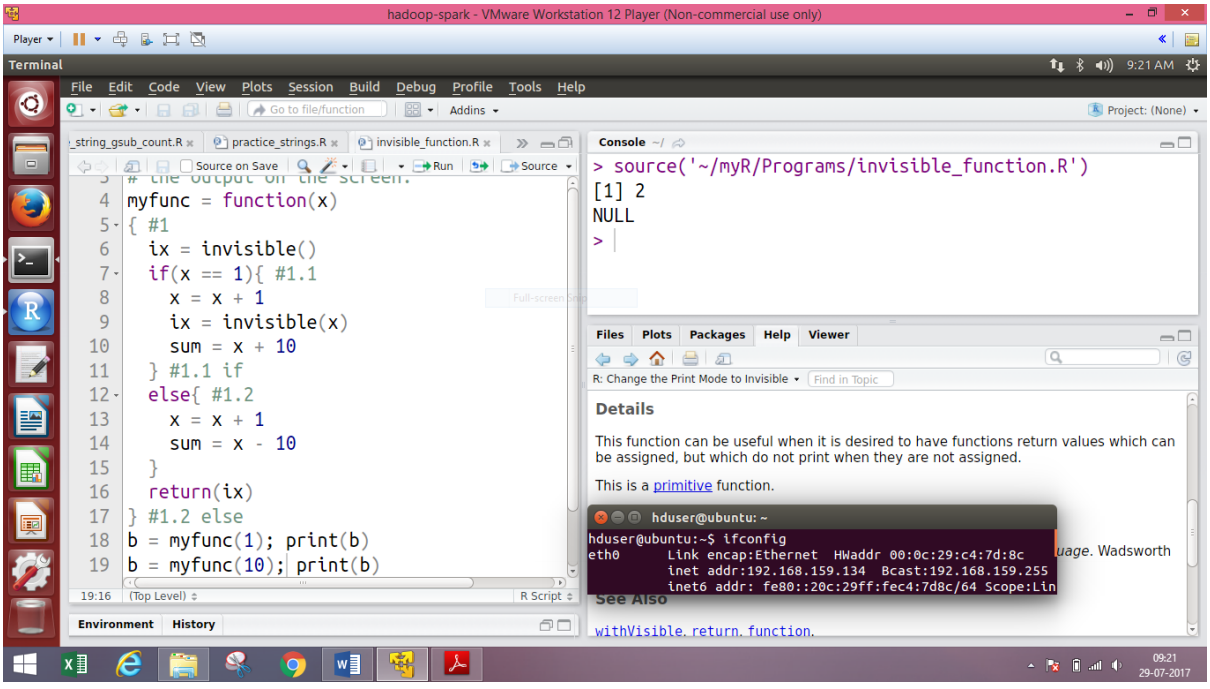


Figure 25: invisible() function

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

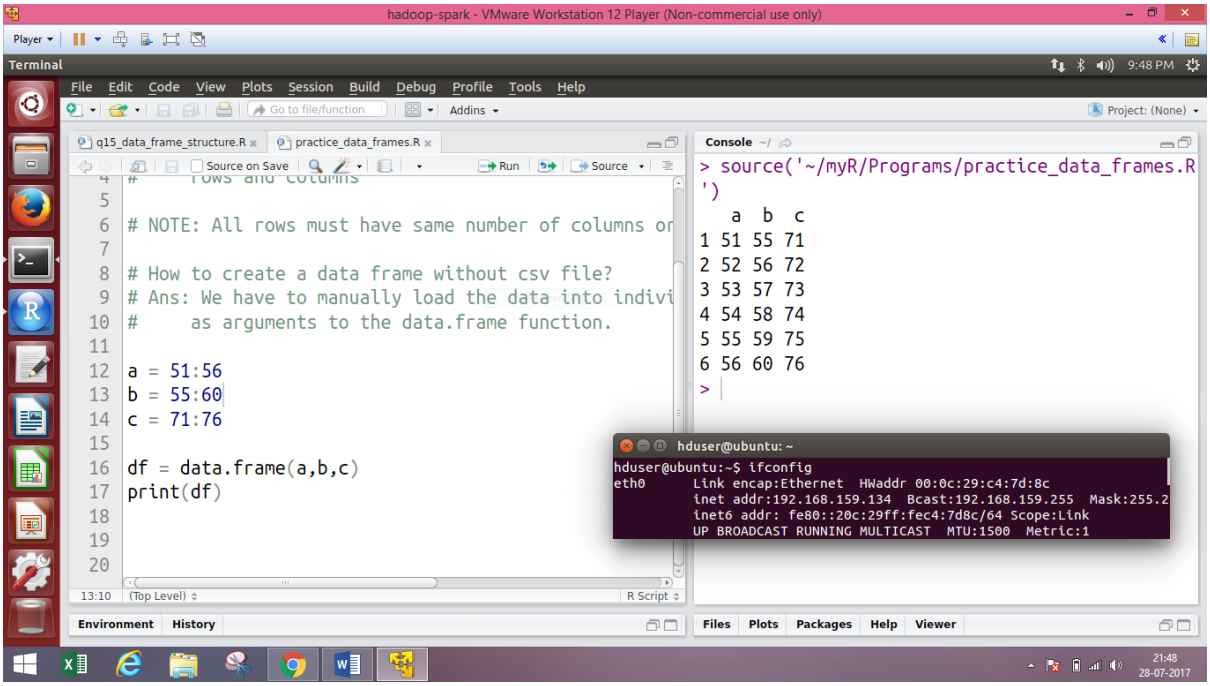


Figure 26: data.frame() function to create and print() function to view Data Frame

ii. Create Data Frame from Matrix

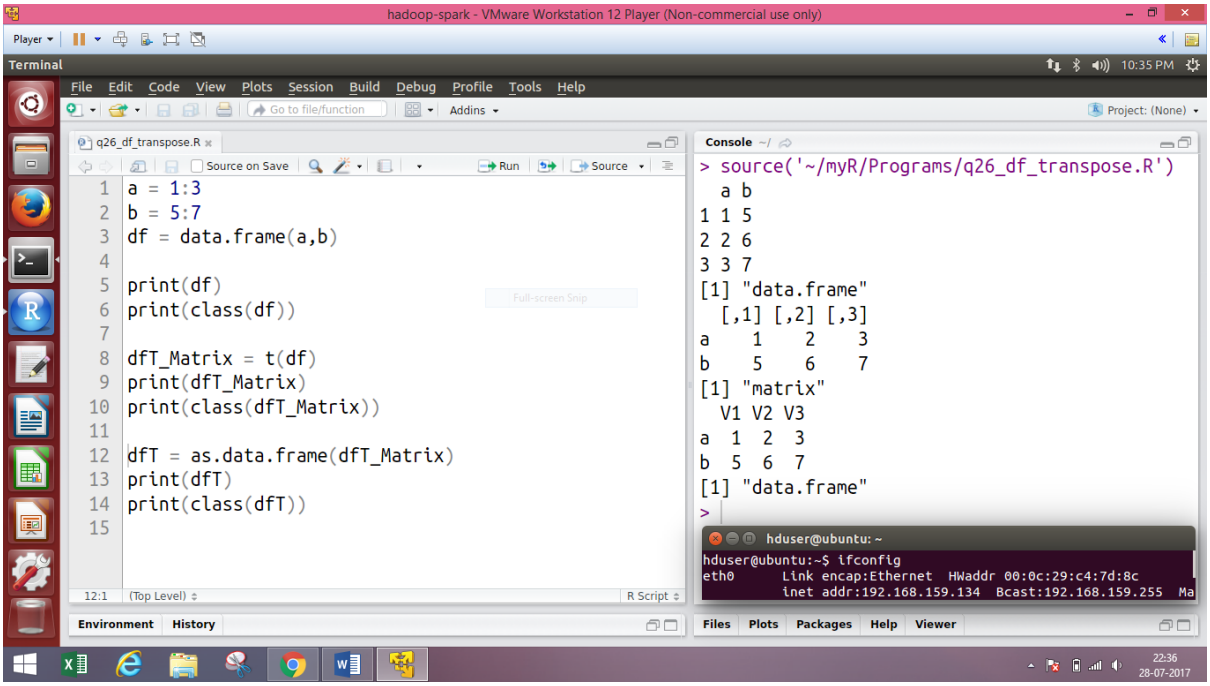


Figure 27: Matrix to Data Frame

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, ...)

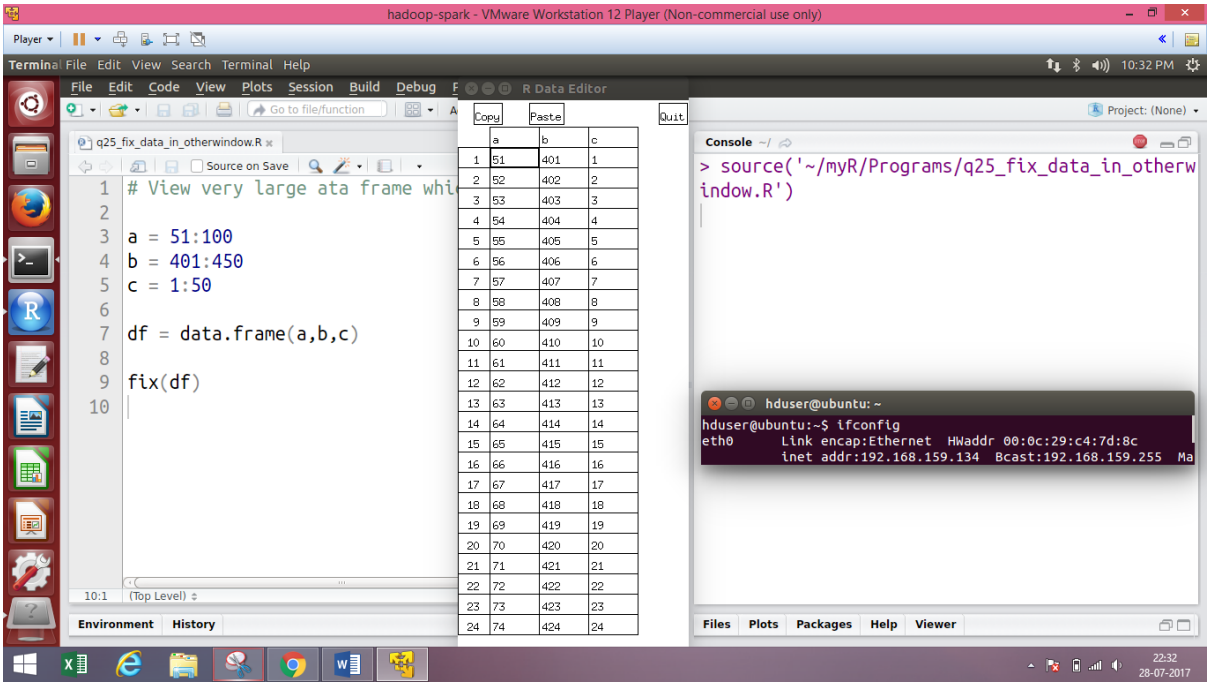


Figure 28: q25 - View Data Frame using fix() function

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, ...)
```

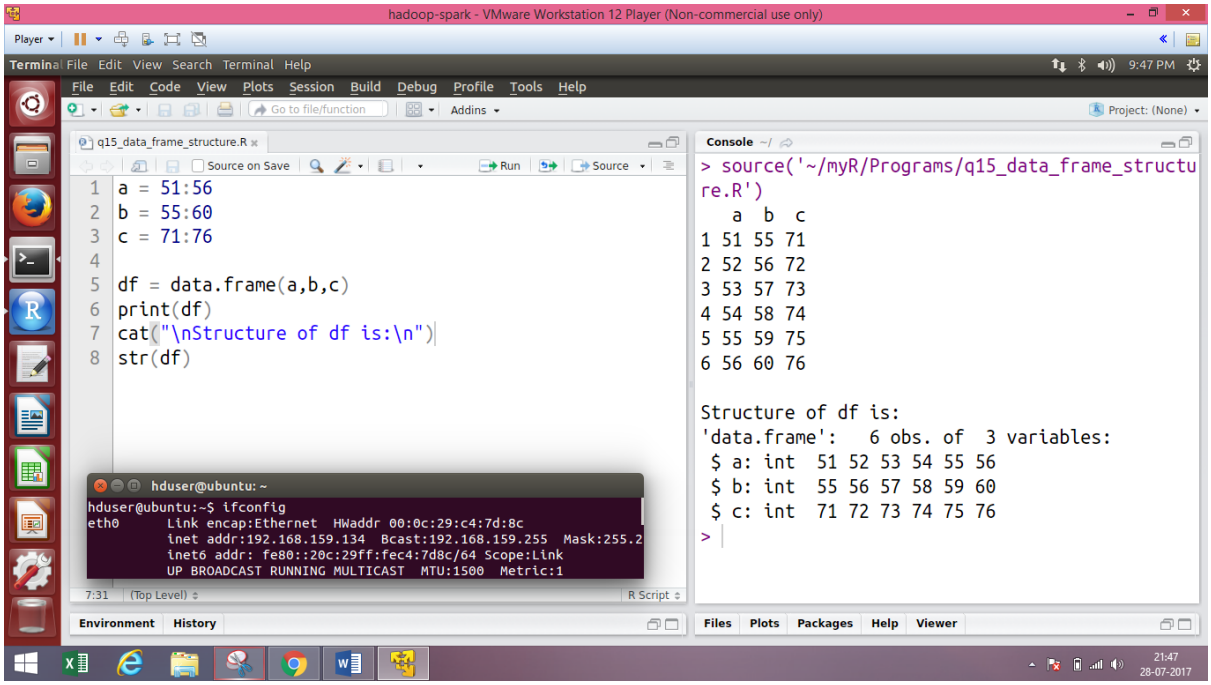


Figure 29: q15 - str() function to check Data Frame Structure

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, ...)

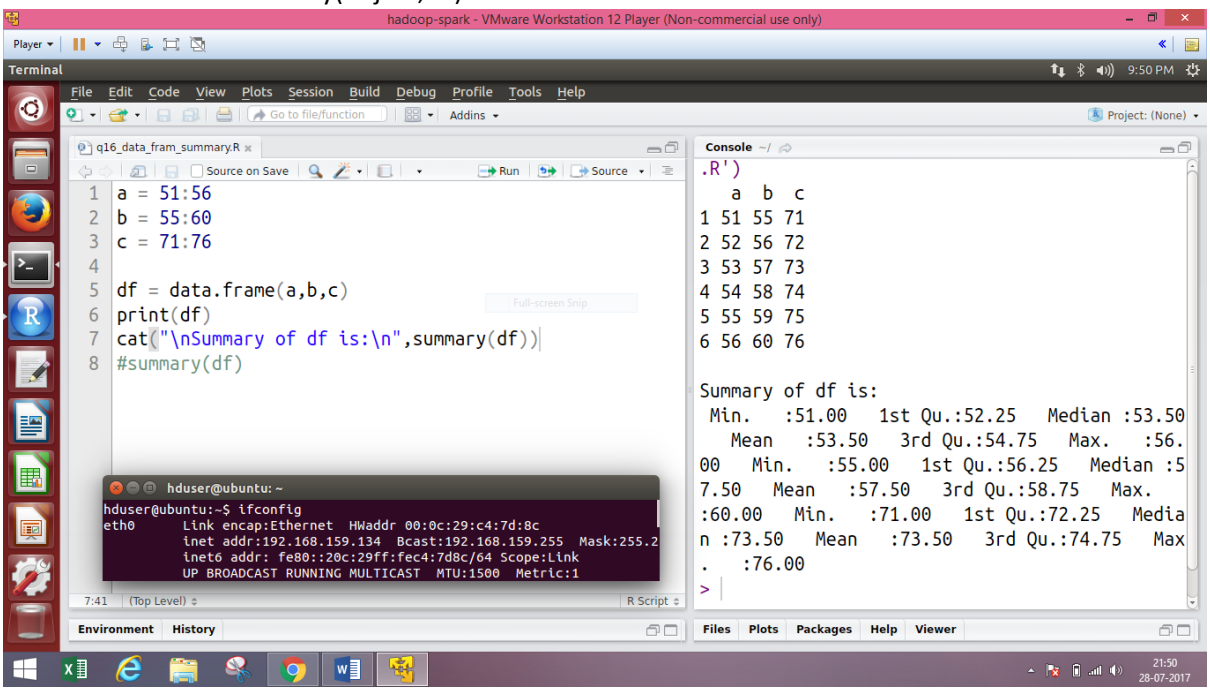


Figure 30: q16 - summary() of Data Frame

d. Extract data from Data Frame

i. Get Row

Class of result set is data.frame.

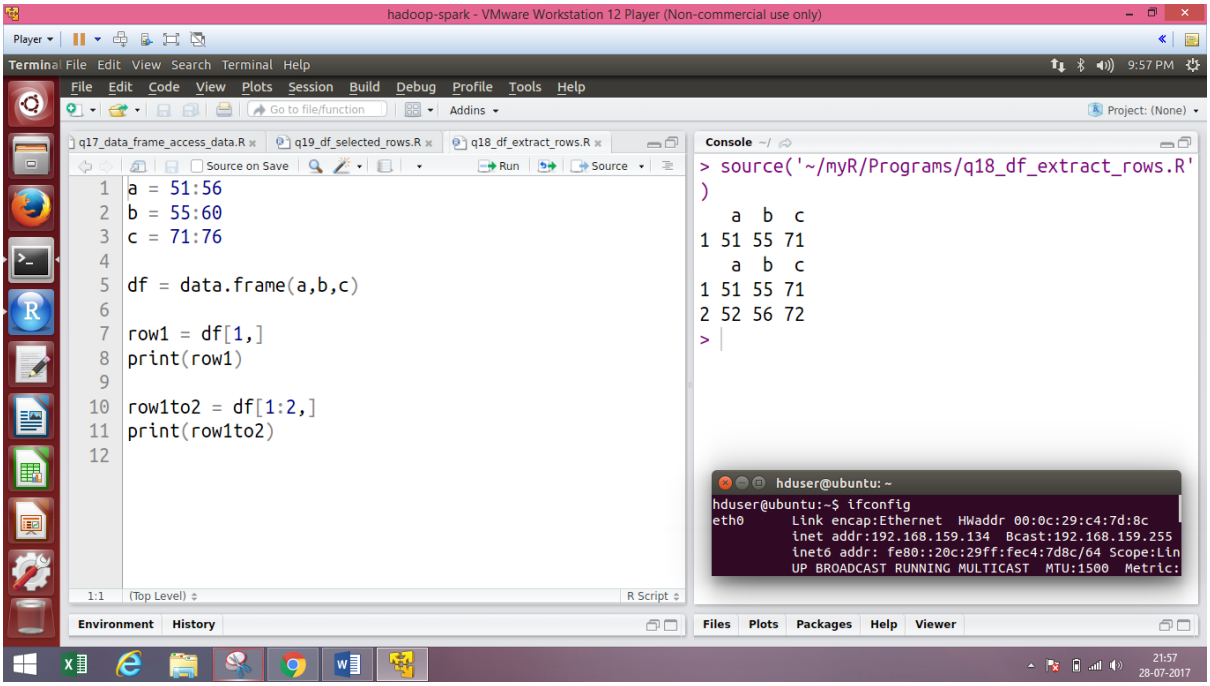


Figure 31: q18 - Extract Row from Data Frame

ii. Get Column

Class of result set is vector.

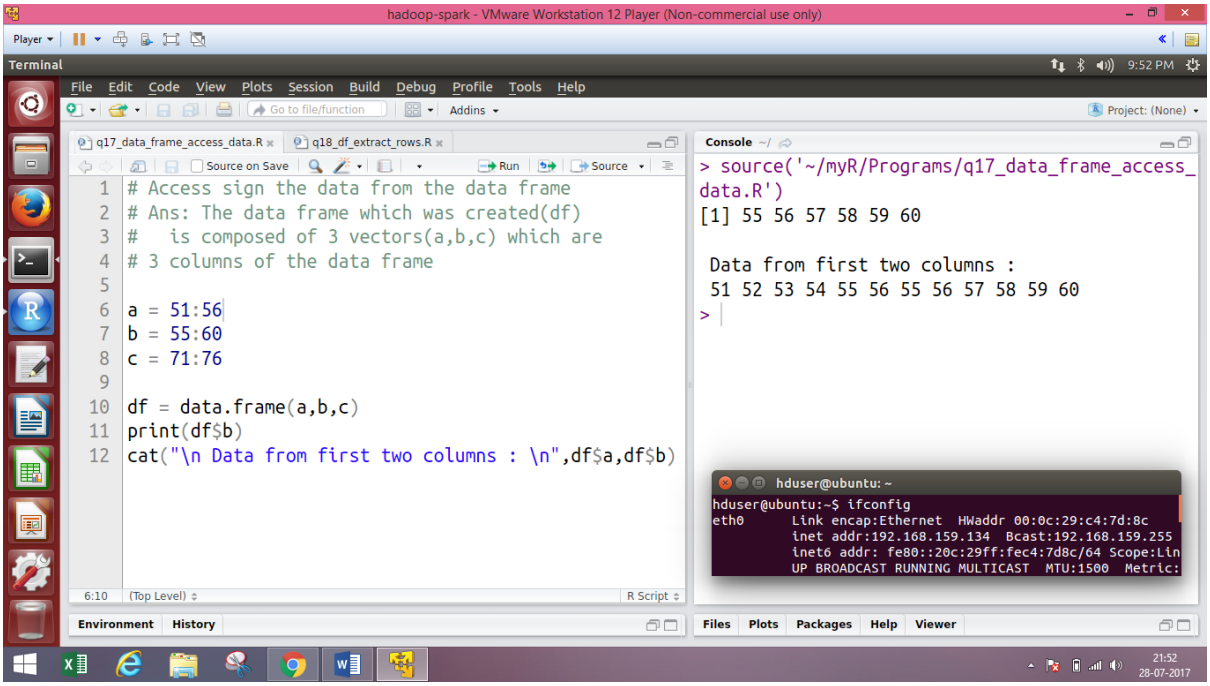


Figure 32: q17 - Access Column from Data Frame

iv. Get Specific Data

Get data from exact row and column.

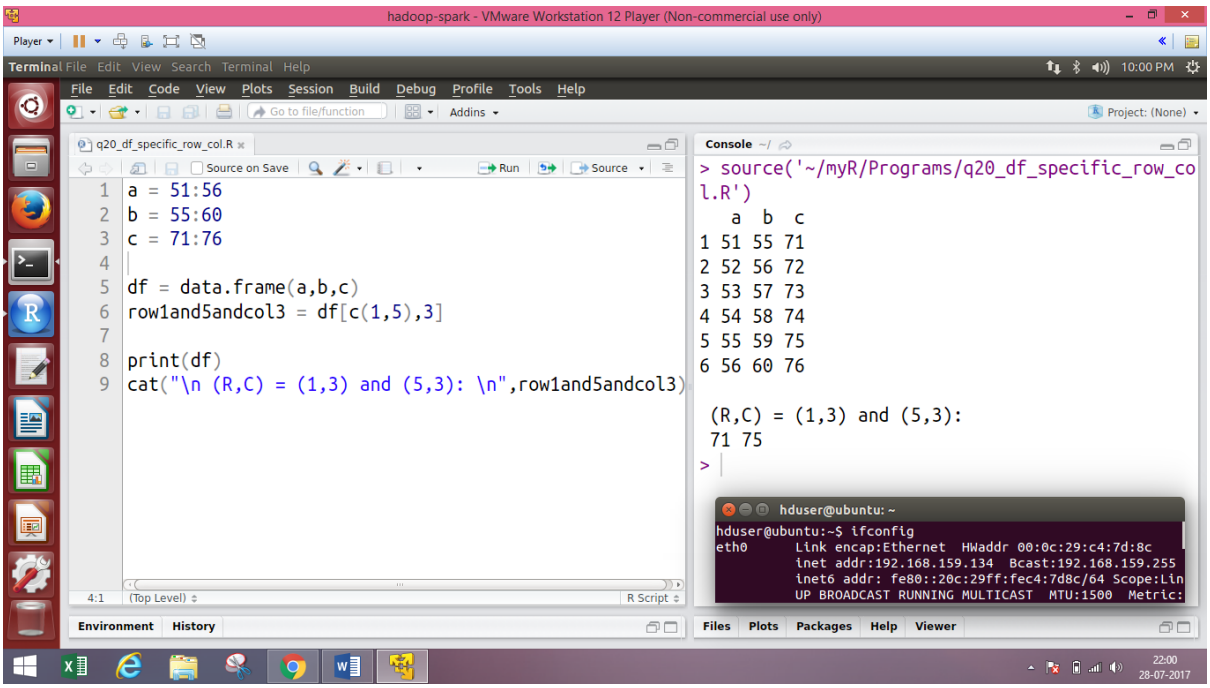


Figure 33: q20 - Access 3 column value data of 1st and 5th row

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, ...)

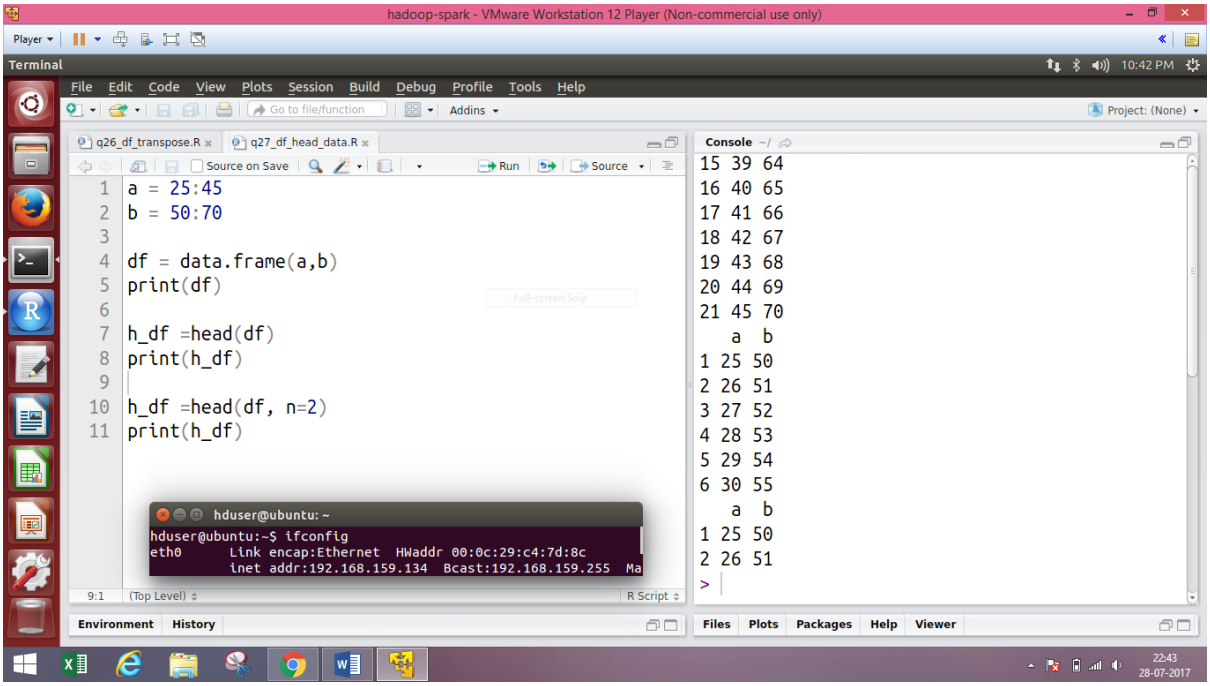


Figure 34: q27 - head() function for Data Frame

vi. Tail of the Data

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

Use:

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

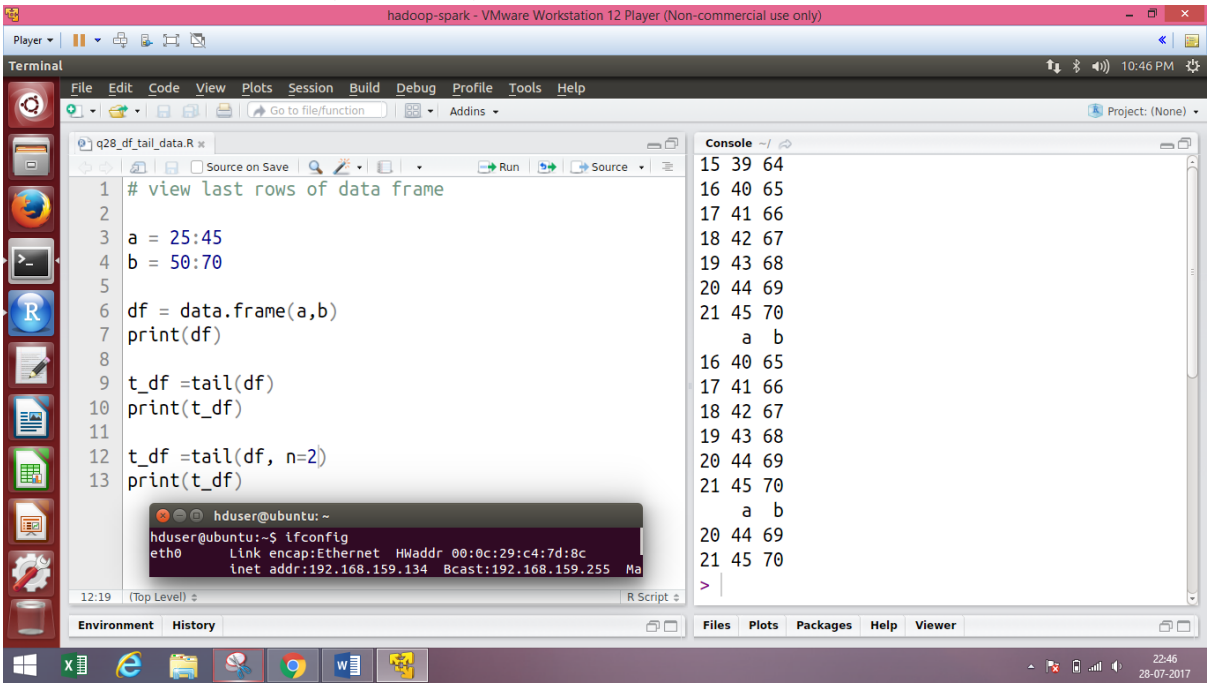


Figure 35: q28 - tail() function for Data Frame

vii. Data without NA (Omit NA)

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

Use:

na.omit(object, ...)

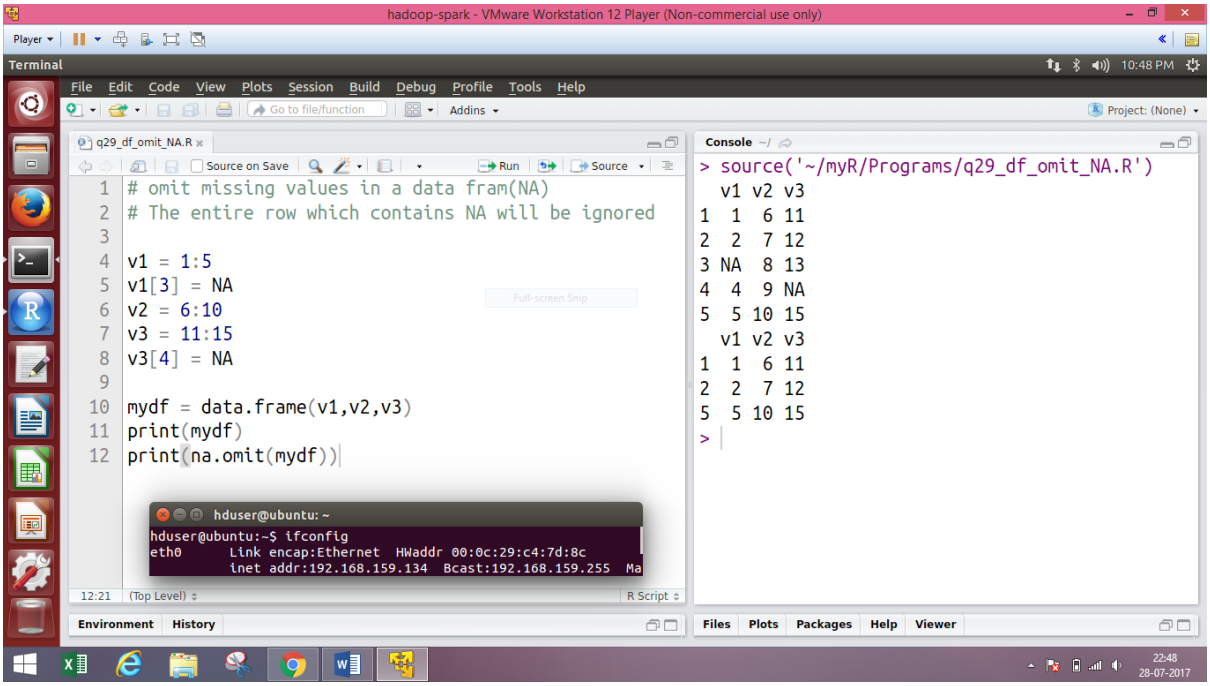


Figure 36: q29 - na.omit() for Data Frame

viii. Which Command

Get specific data which satisfies given condition.

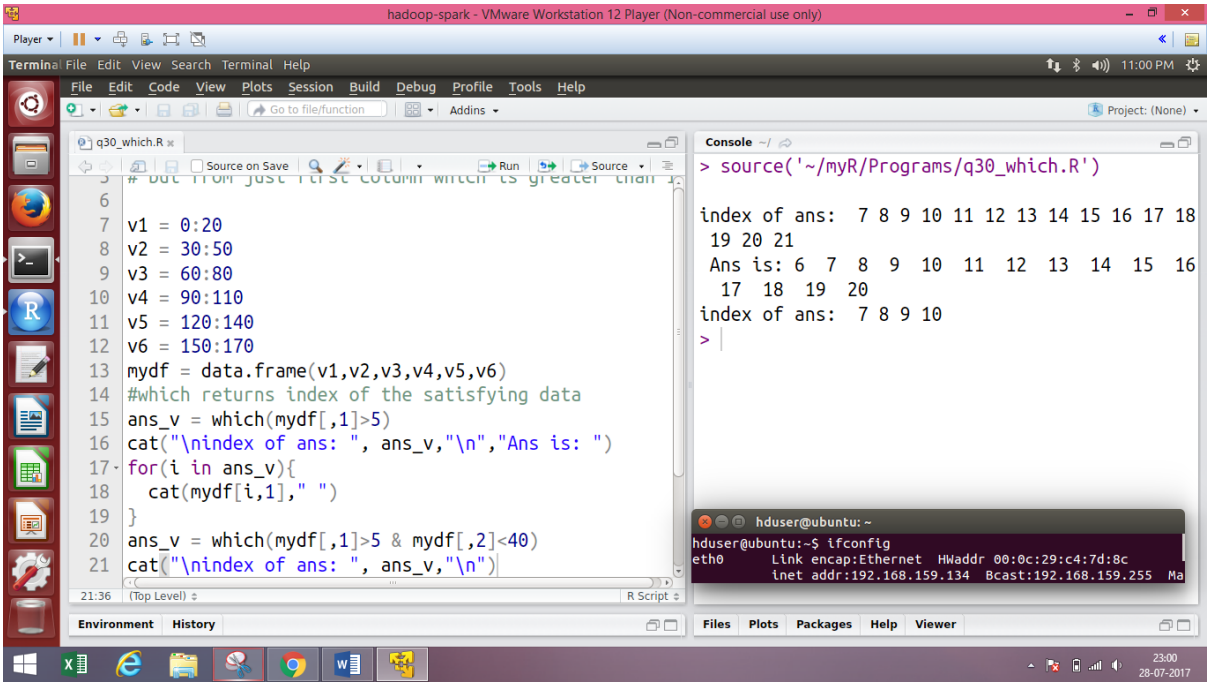


Figure 37: q30 - which() function to get specific data from Data Frame

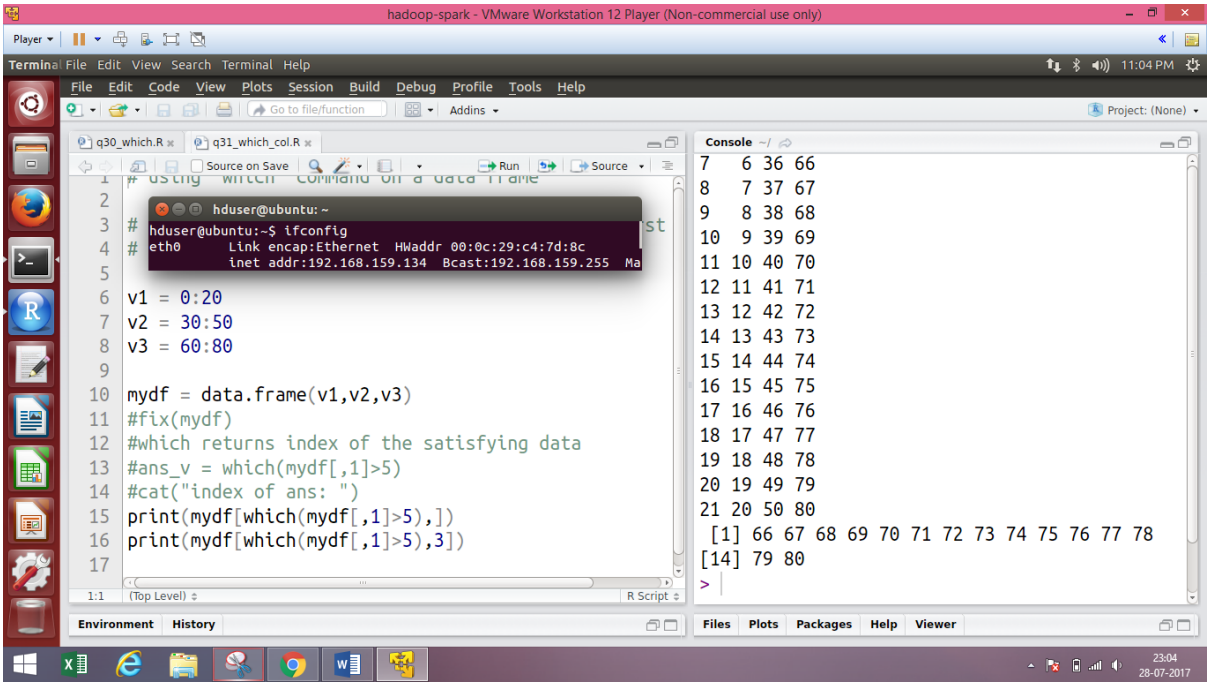


Figure 38: which() function on column and row of Data Frame

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.

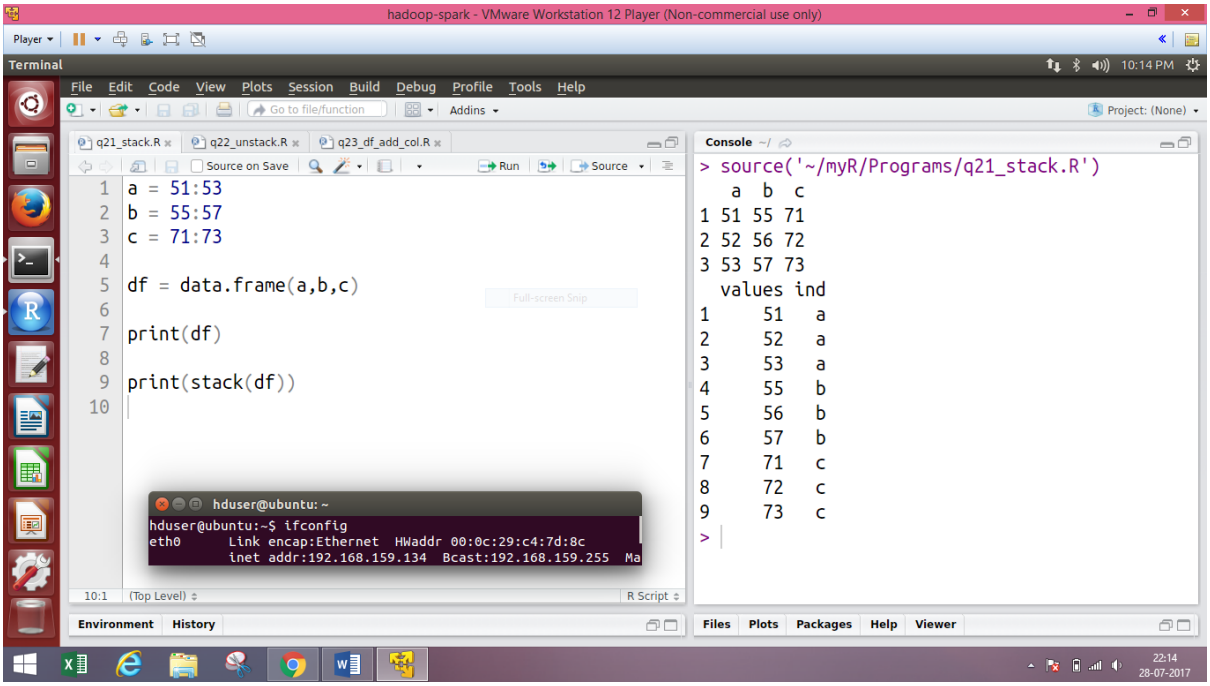


Figure 39: q21 - Stack data from Data Frame

f. unstack() function

Unstacking reverses operation of stack.

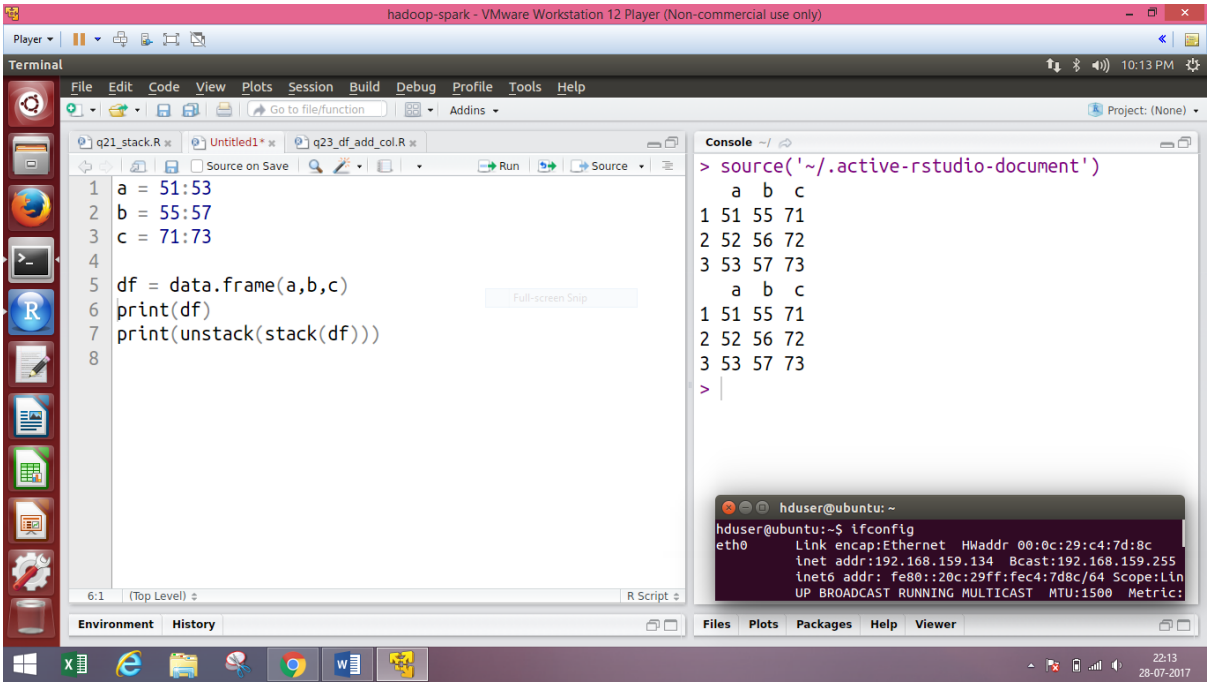


Figure 40: q22 - Unstack vector back to Data Frame

g. Expand Data Frame

i. Add Column using cbind() function

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

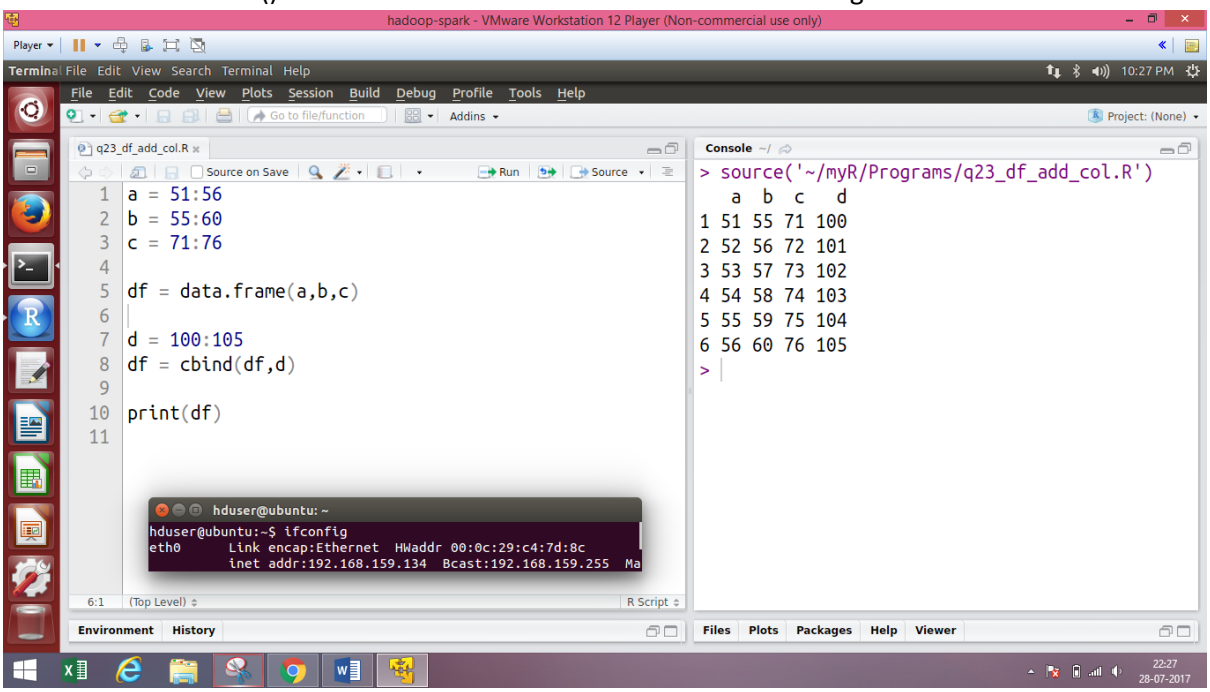


Figure 41: q23 - cbind() function to add column in Data Frame

ii. Add Row using rbind() function

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

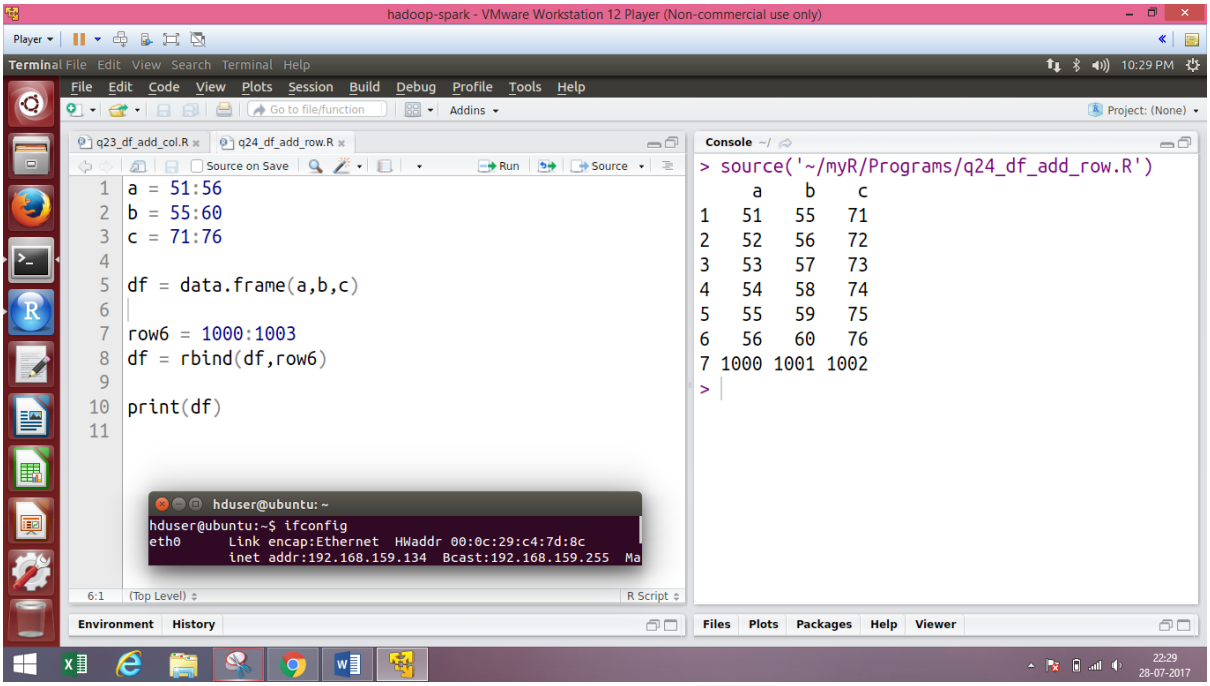


Figure 42: q24 - rbind() function to add row to existing Data Frame

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"))

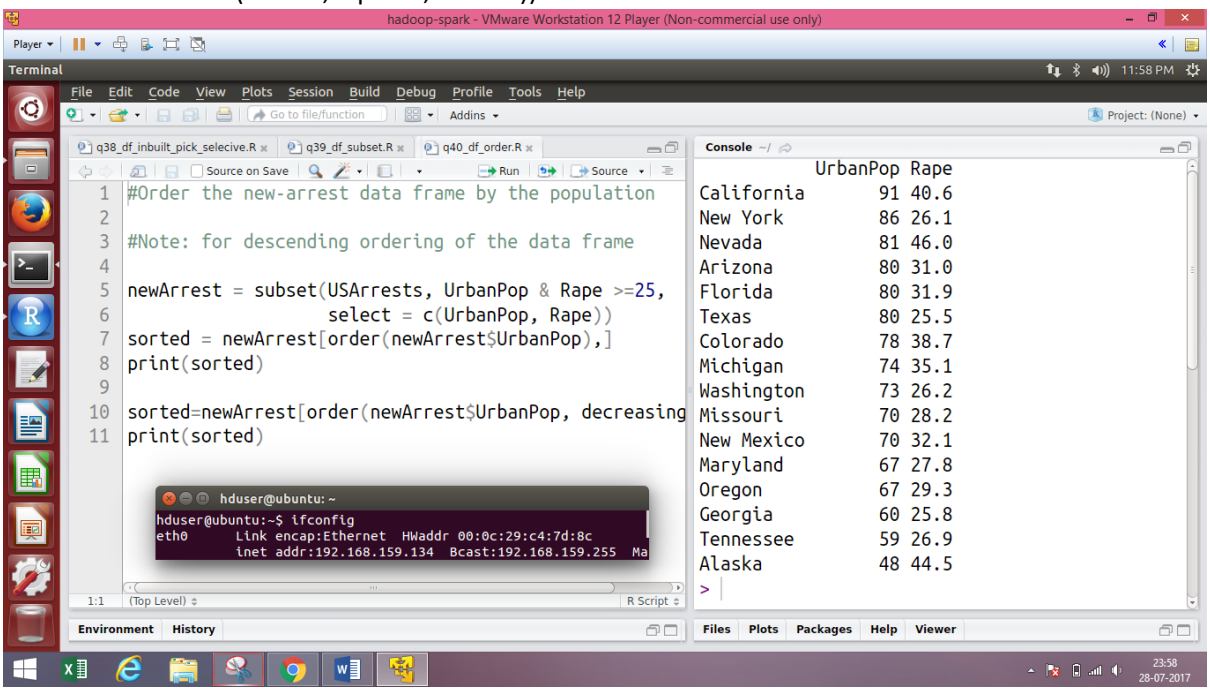


Figure 43: Order Data Frame

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)

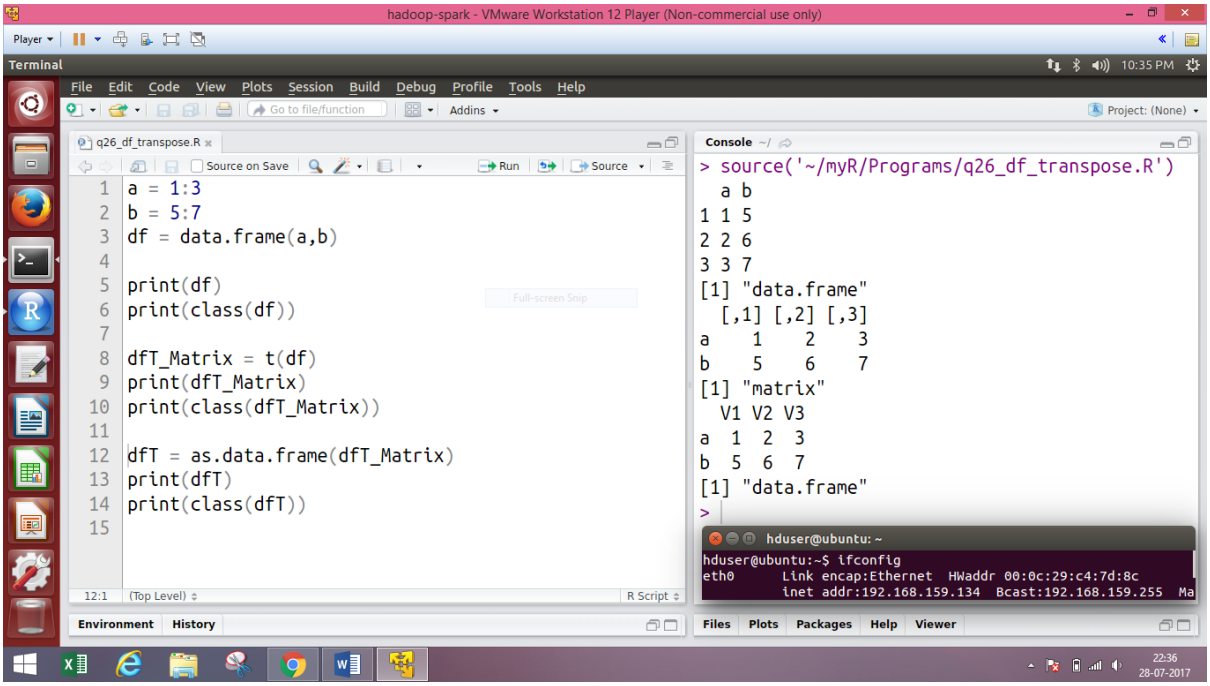


Figure 44: q26 - t() function to transpose the Data Frame

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)

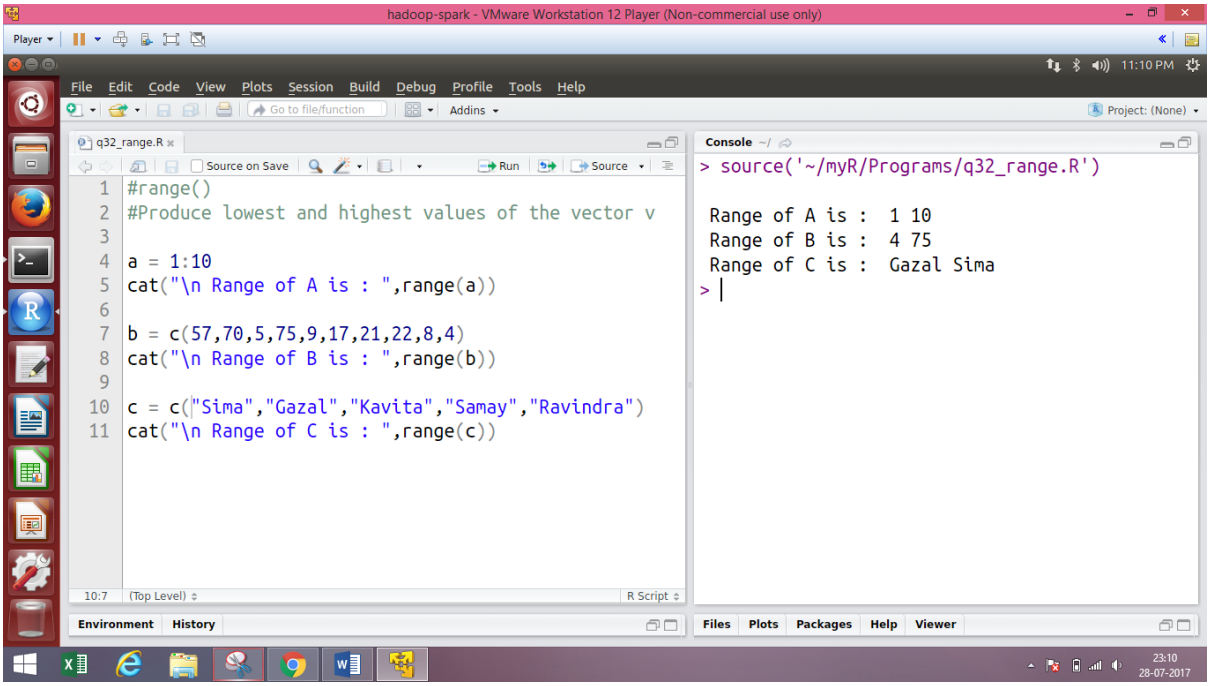


Figure 45: q32 - range() function

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, ...)

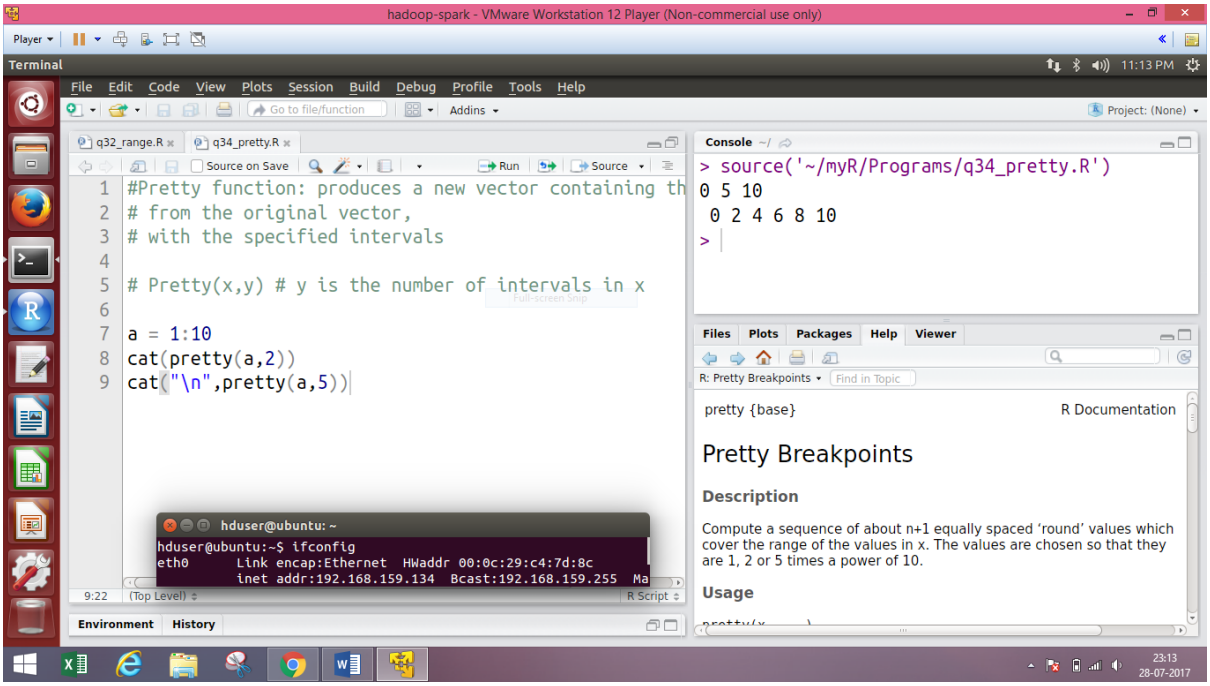


Figure 46: q33 - pretty() function

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.

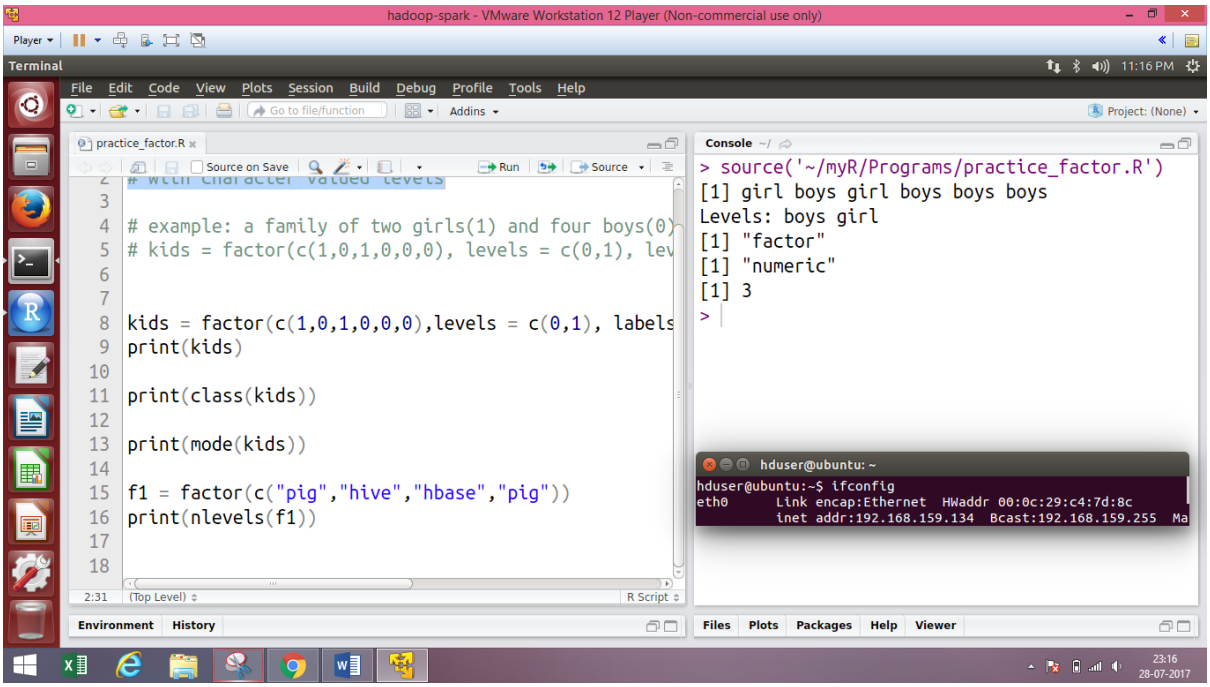


Figure 47: q34 - Factors

- **(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

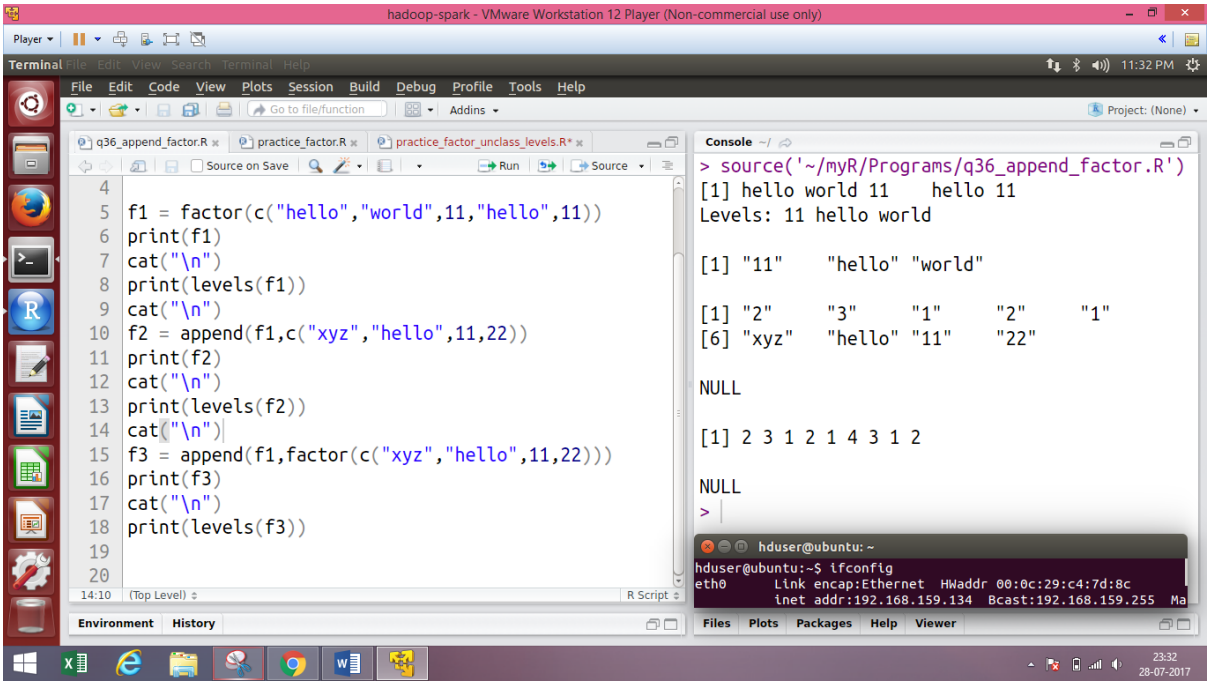


Figure 48: q36 - append() in Data Frame

- **unclass() function**

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

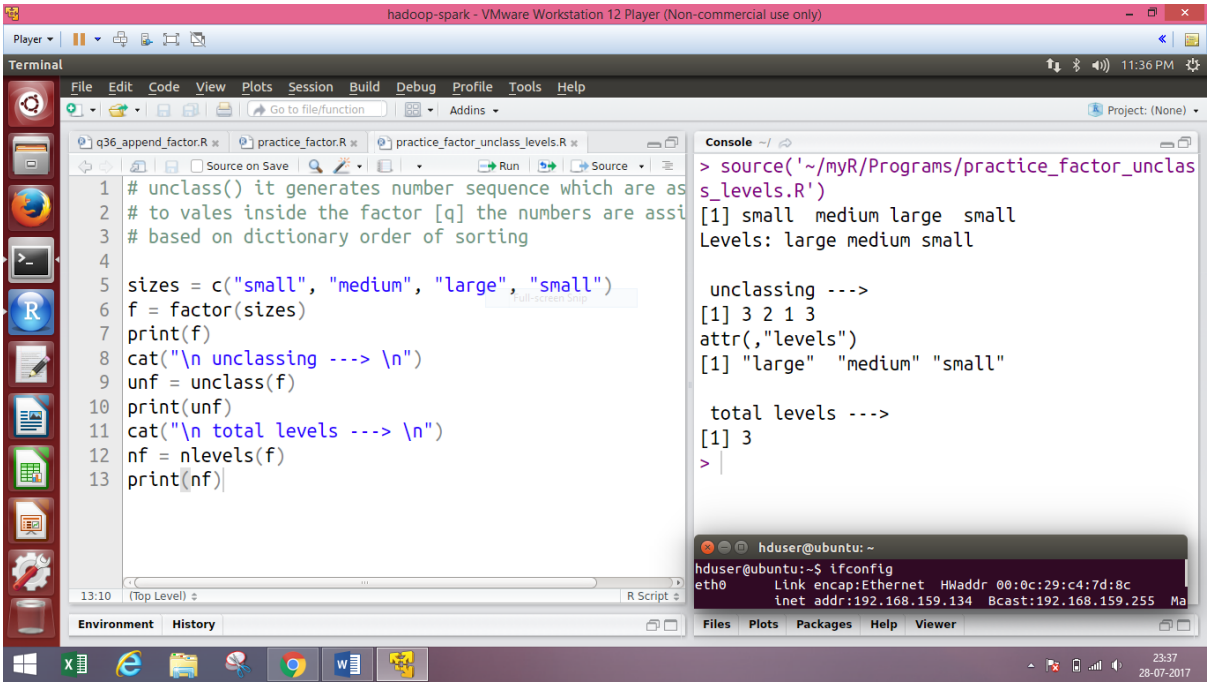


Figure 49: unclass() function for factors

- **table() function**

  Create table from factors.

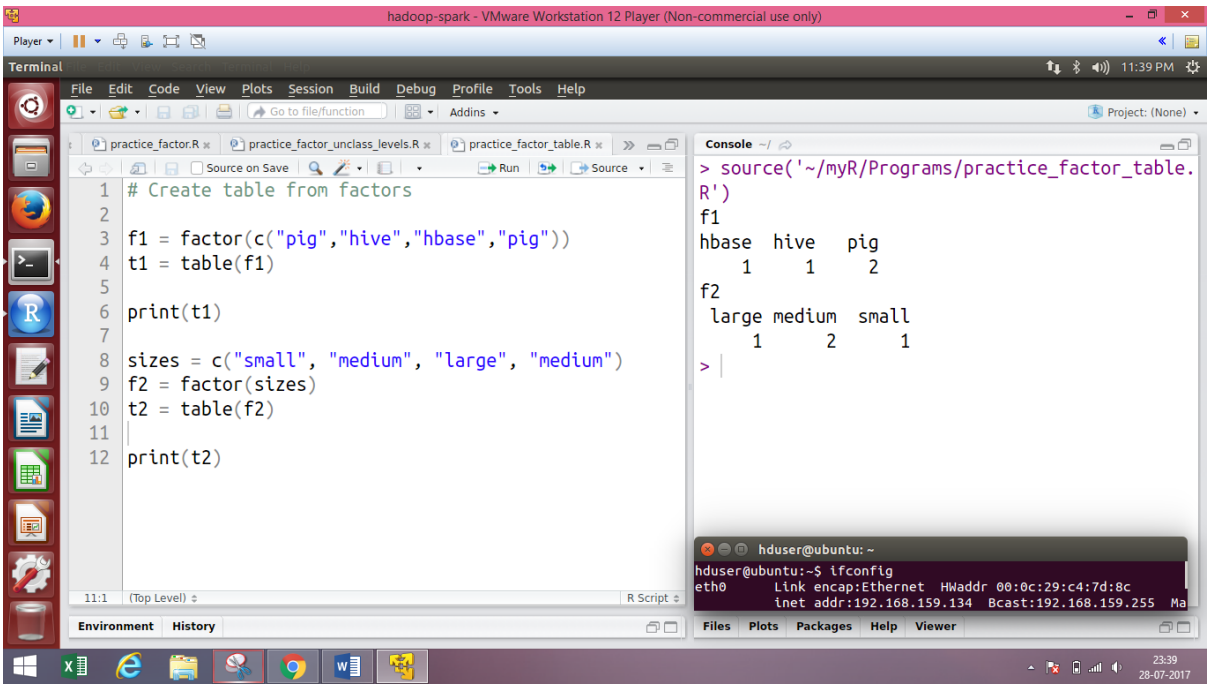


Figure 50: table() to make table from factors

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, ...)
```

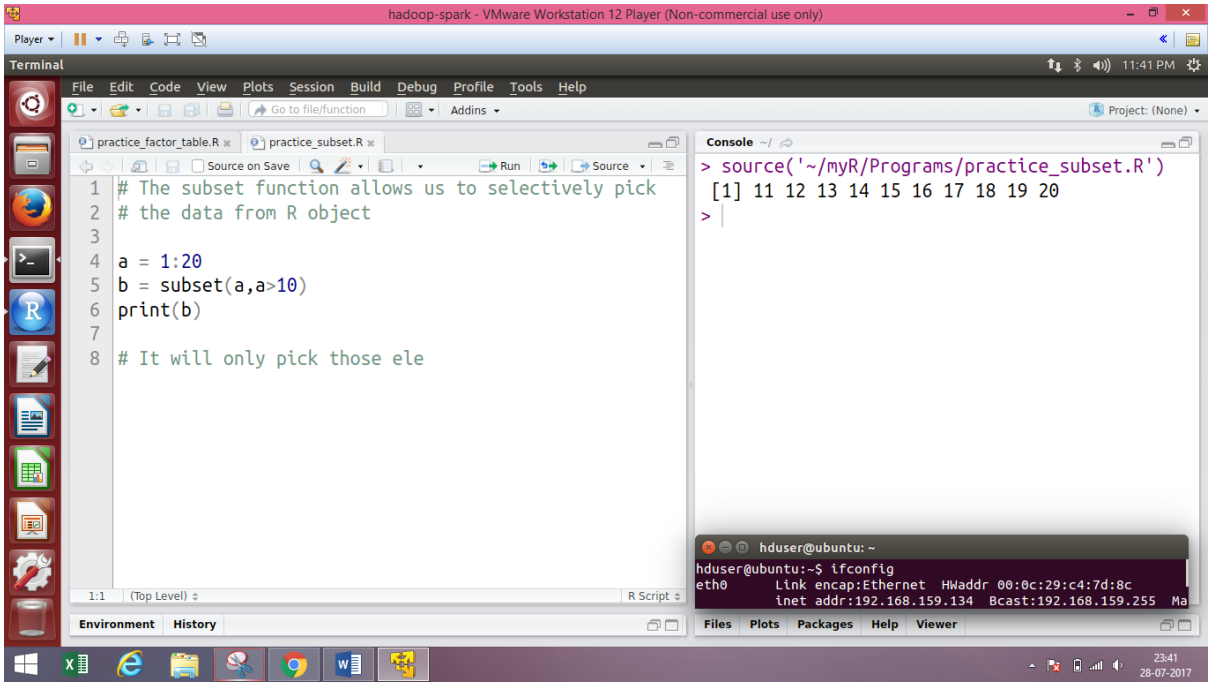


Figure 51: q - subset() function

a. Create Subset from Data Frame

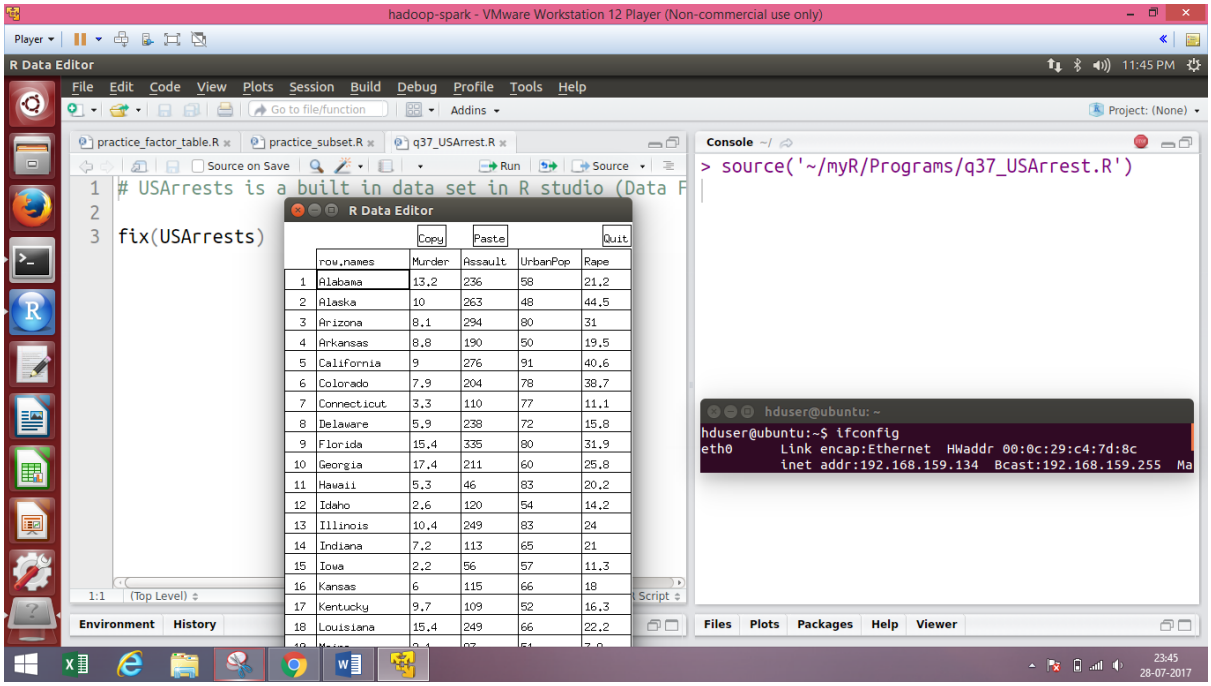


Figure 52: q37 - subset() function on Data Frame

b. Selection from Data Frame

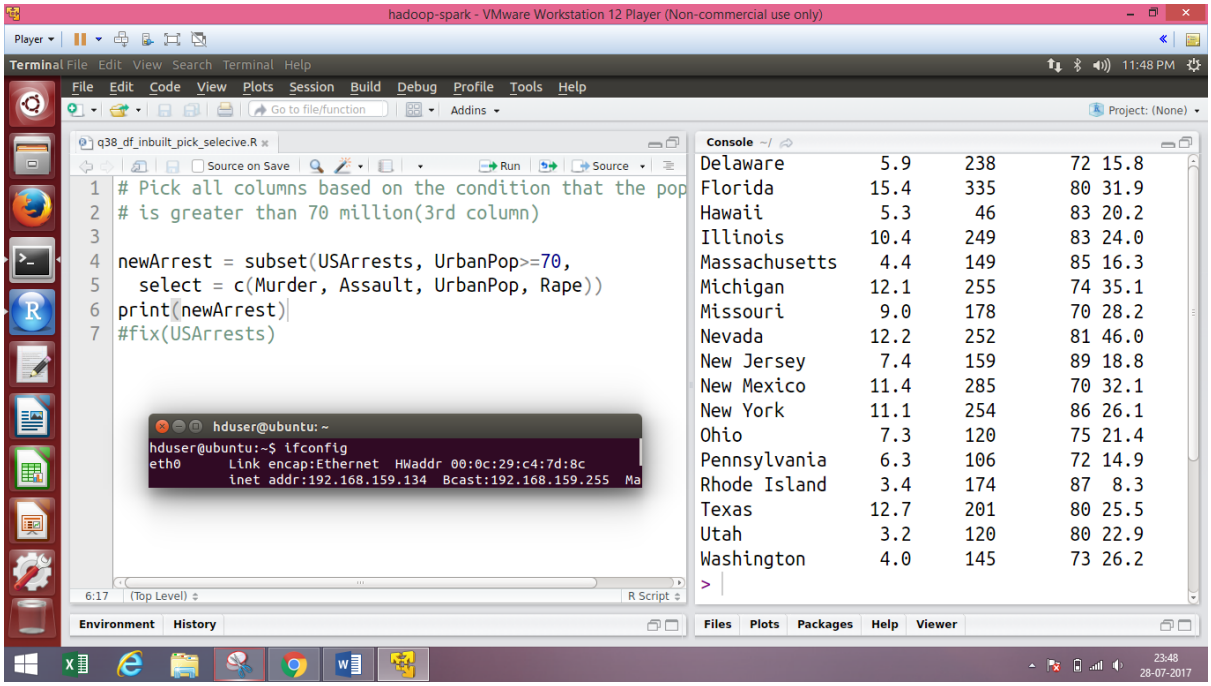


Figure 53: q38 - Pick up columns based on the condition using subset() function on Data Frame

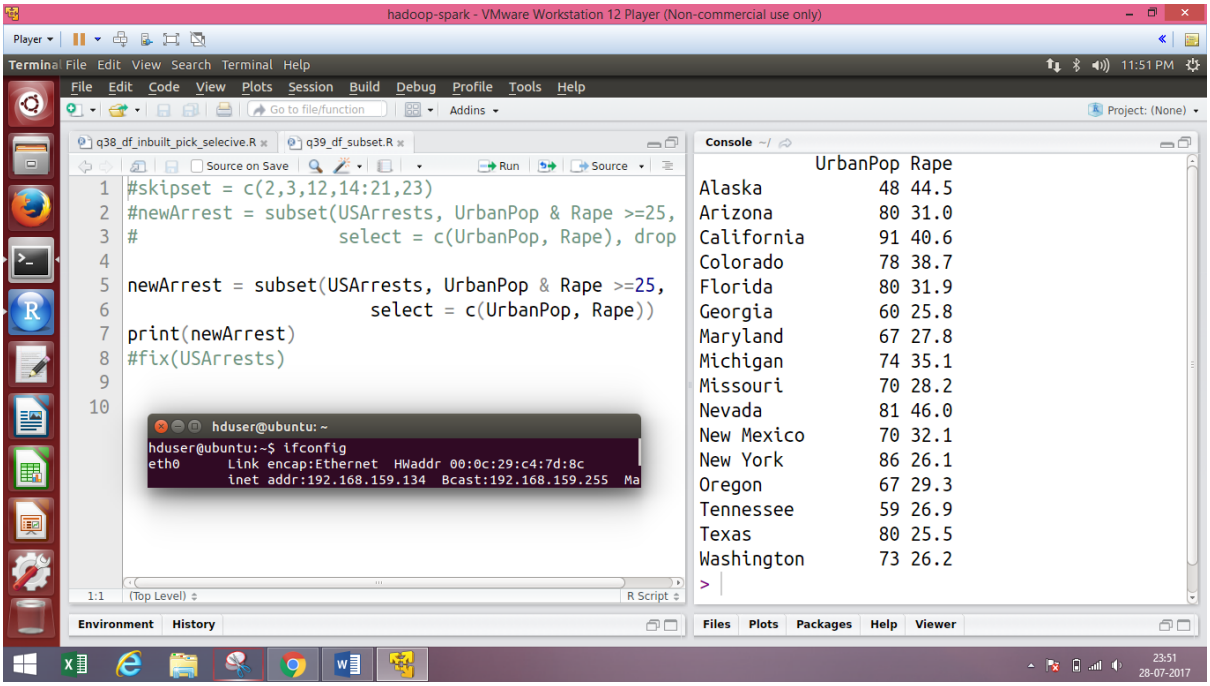


Figure 54: q39 - Subsetting Data Frame

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)

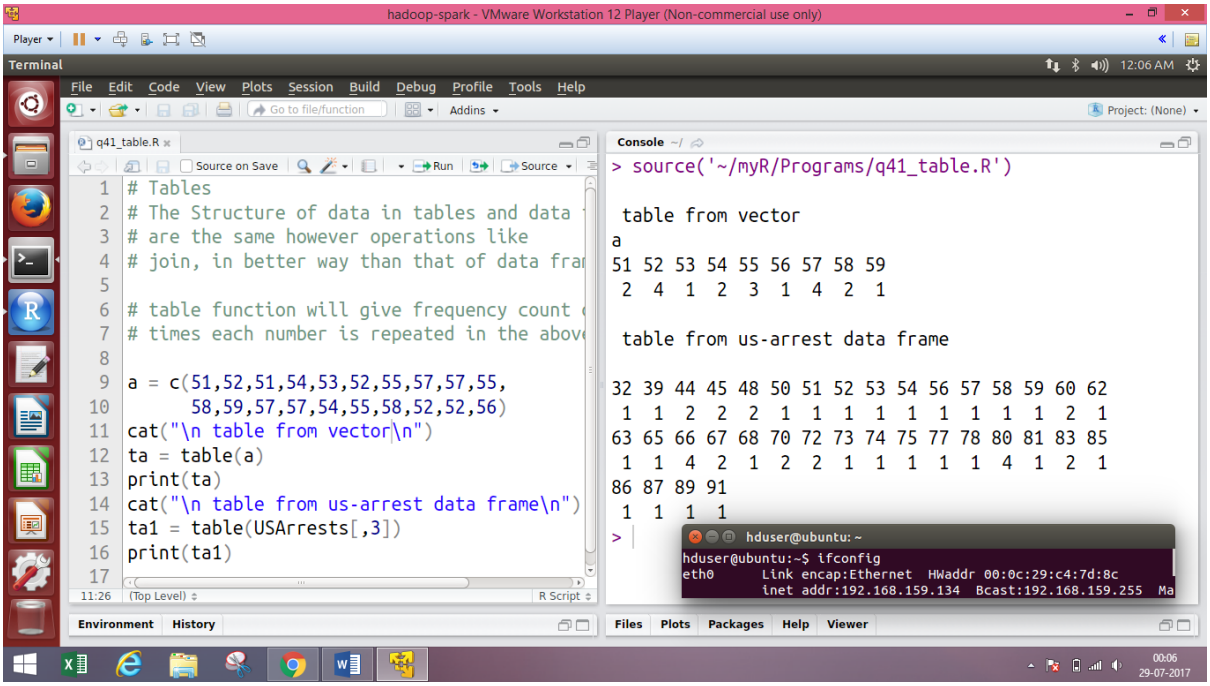


Figure 55: q41 - table() function, Create table from vector

Function table() with multiple arguments.
For example,

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

First argument becomes column names, and second row names.
The table elements will be the frequency count.

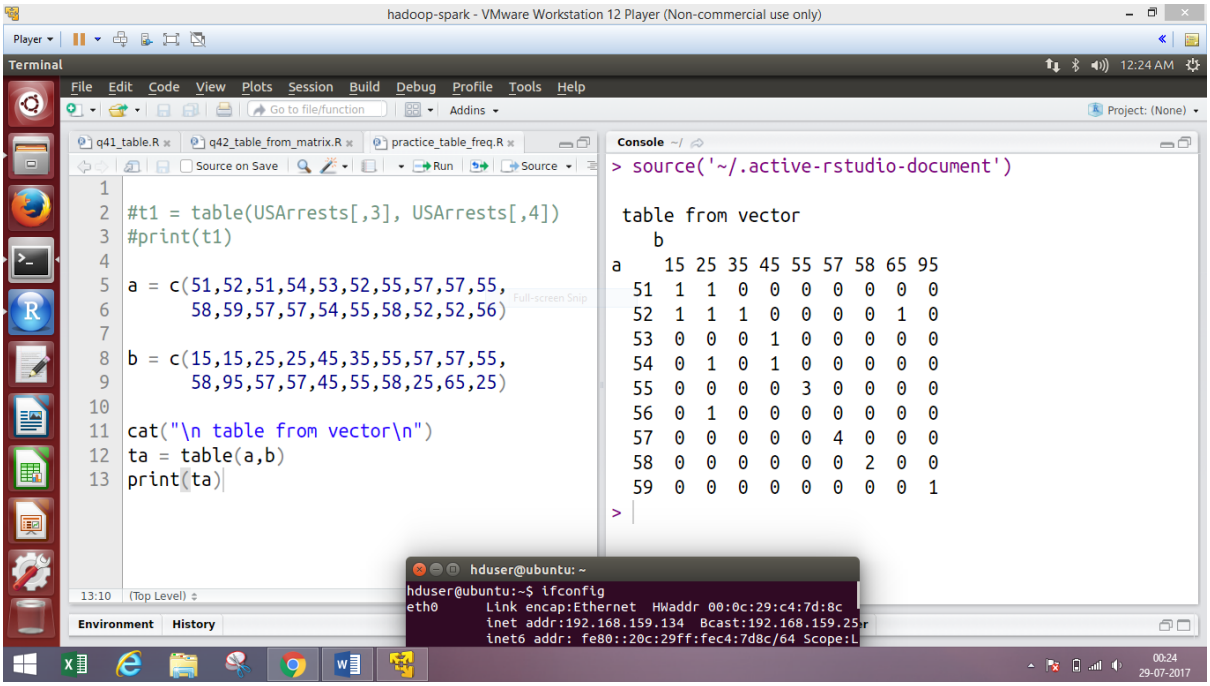


Figure 56: Table using table() function indicating frequency

(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.

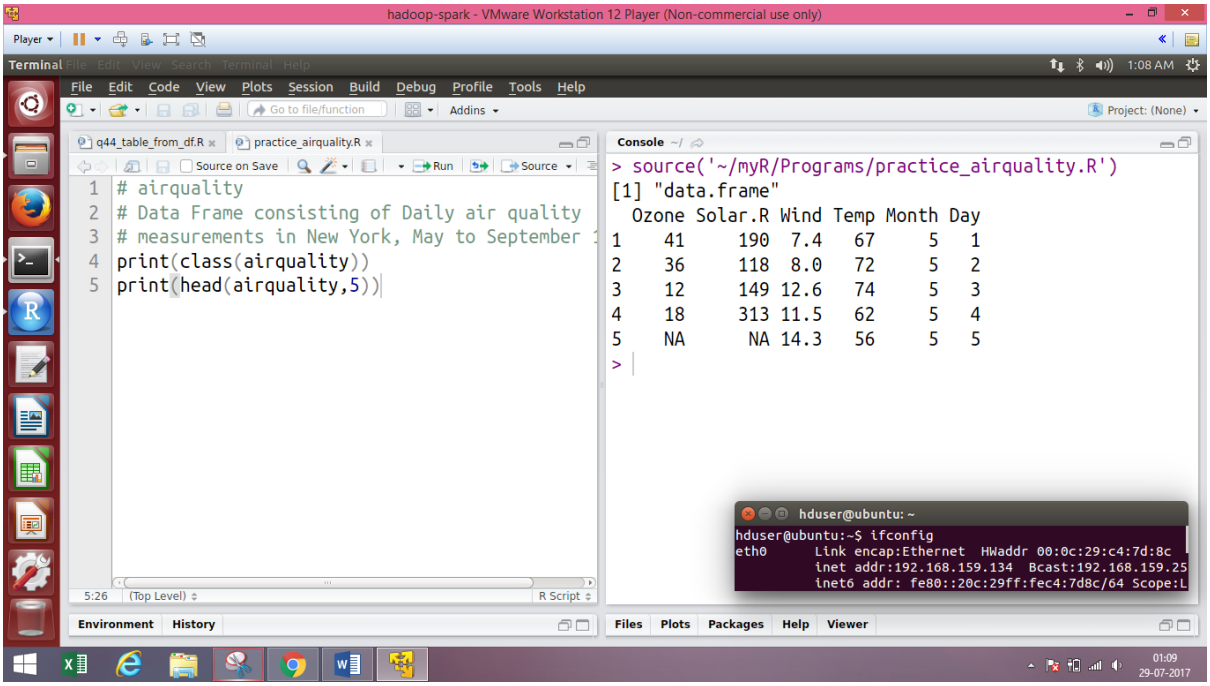


Figure 57: Table from Data Frame

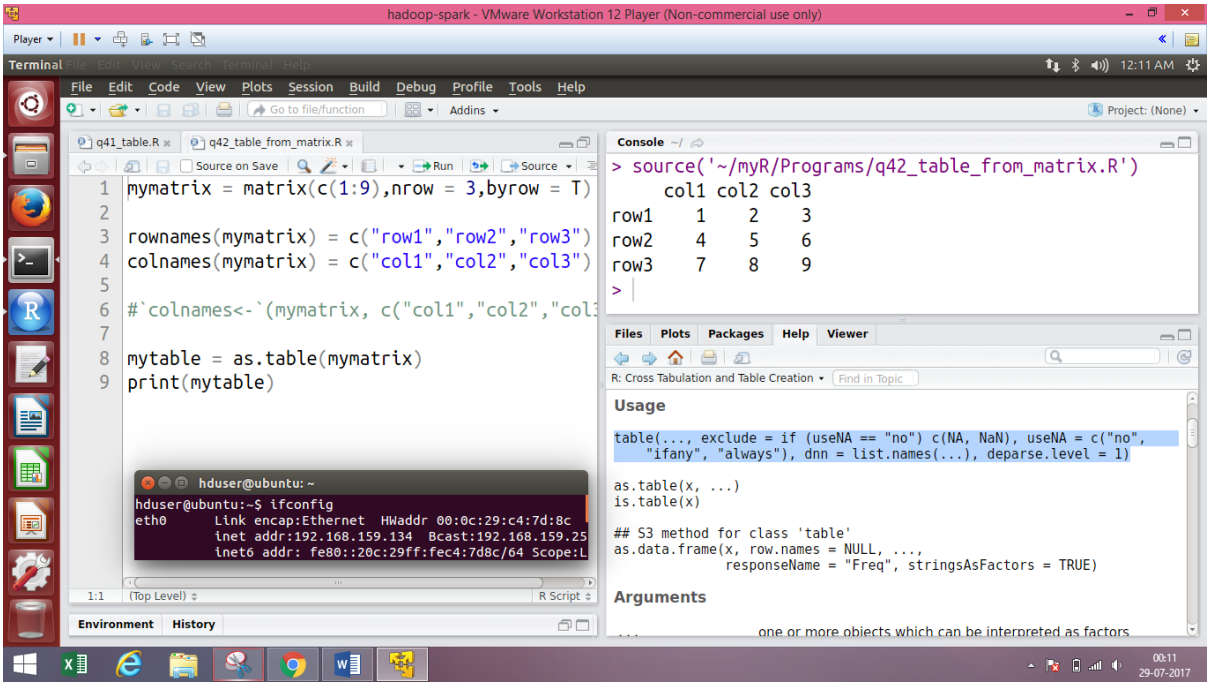


Figure 58: q42 - Table from Matrix

10.Difference

a. R vs. Non-R Programming

R Programming	Non-R Programming
• In R, vector and list and arrays are different and has different properties. • List can be named or un-named.	• 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
• 2 dimensional data structure • Homogeneous since a matrix could contain only identical data type.	• 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
○ **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
○ 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
○ 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
○ It's more like an array but with **only 2 dimensions.** ○ (non-R programmer's terms) it's still a **matrix** :)
Data Frame
○ 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 columnames)

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 = "", ...)
```

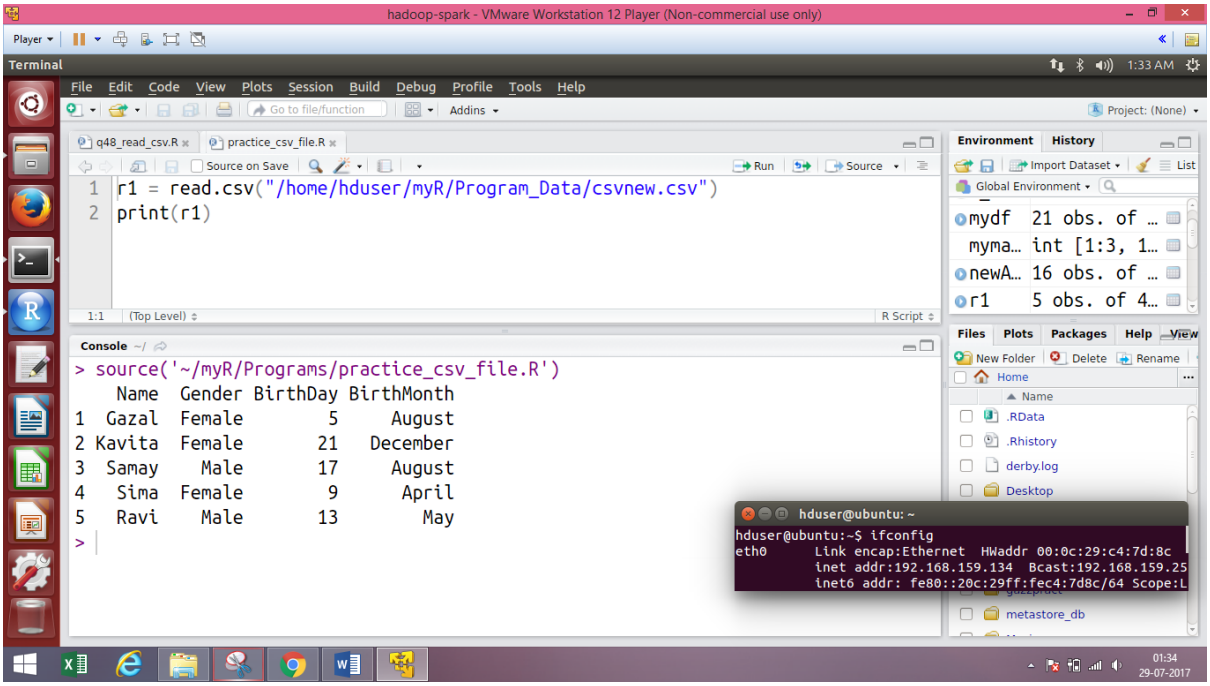


Figure 59: read.csv() function to read .CSV file

1. Default Header from file

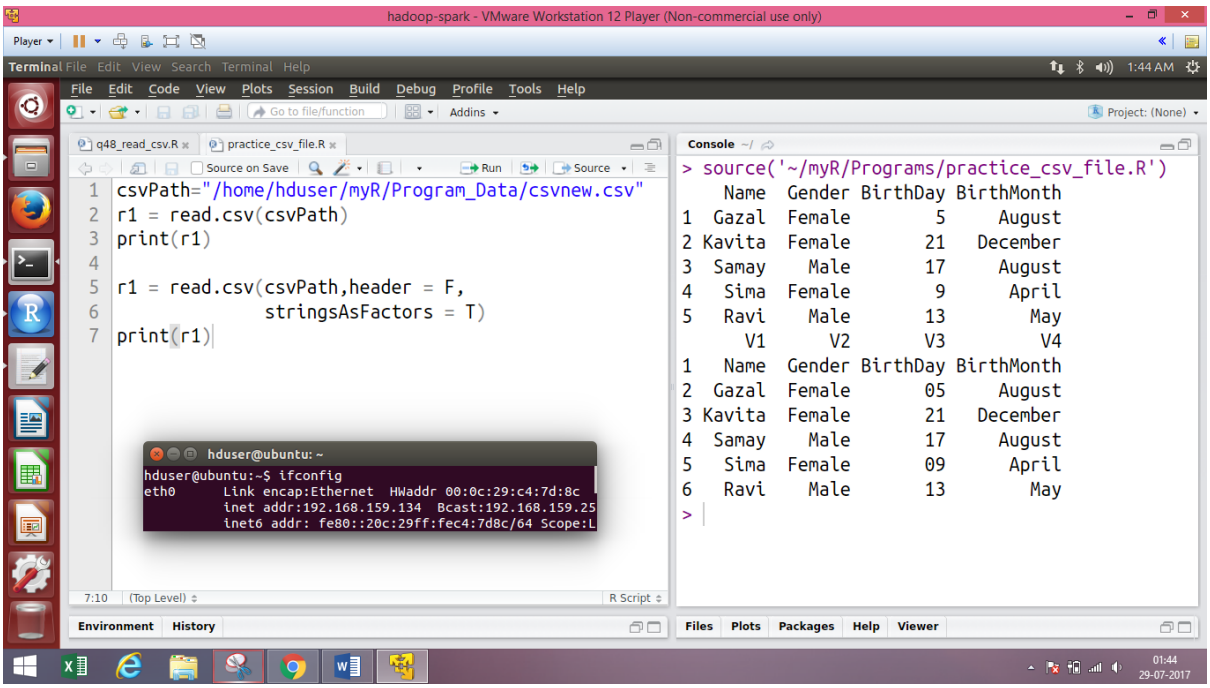


Figure 60: q48 - Read csv with title header as data

2. Skip few lines from header

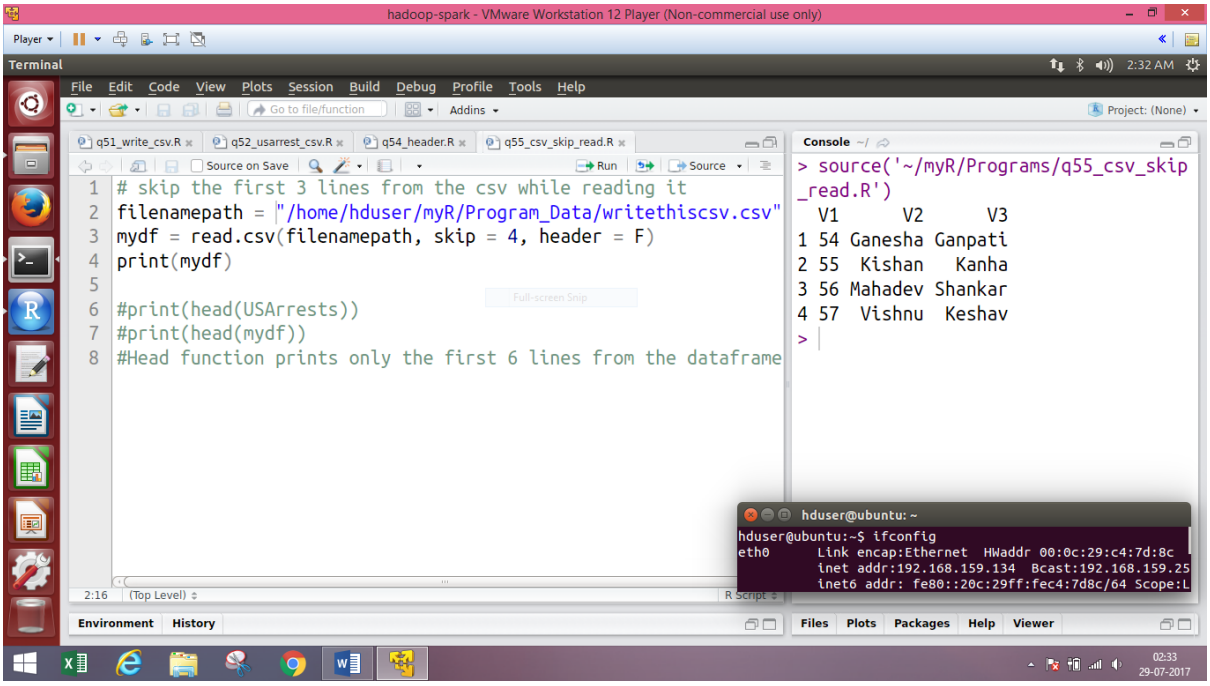


Figure 61: Skip Header lines

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

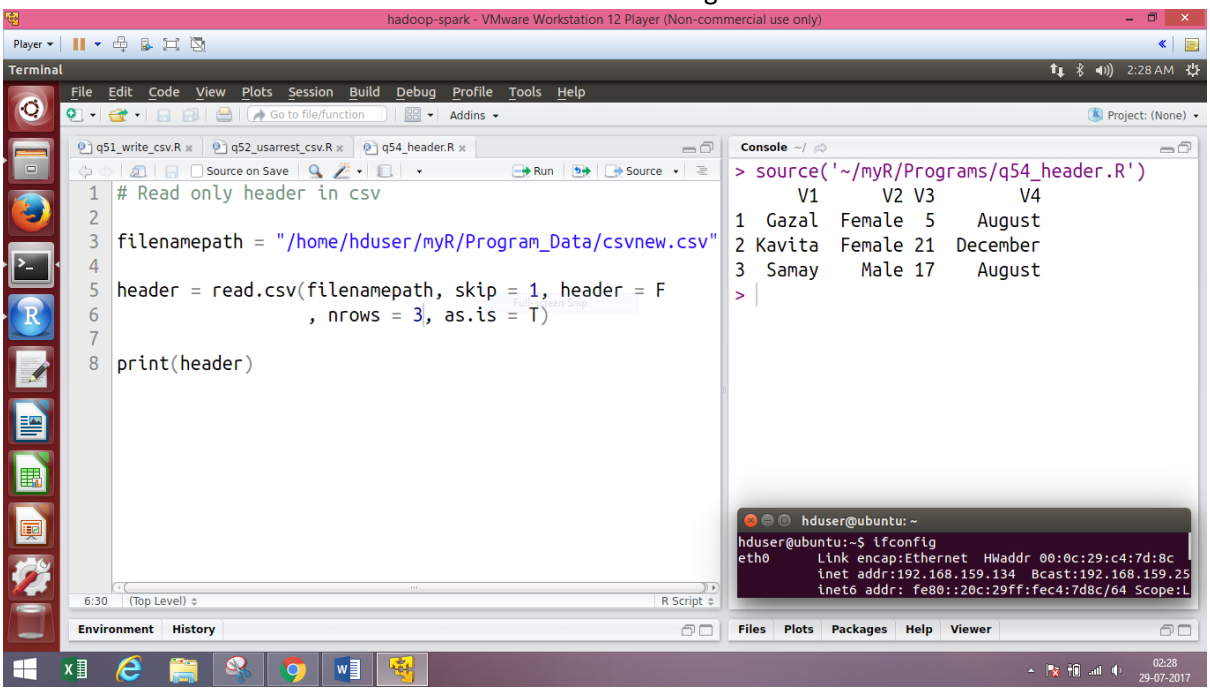


Figure 62: q54 - Read csv file header

ii. Data type

It returns data.frame class object as result.

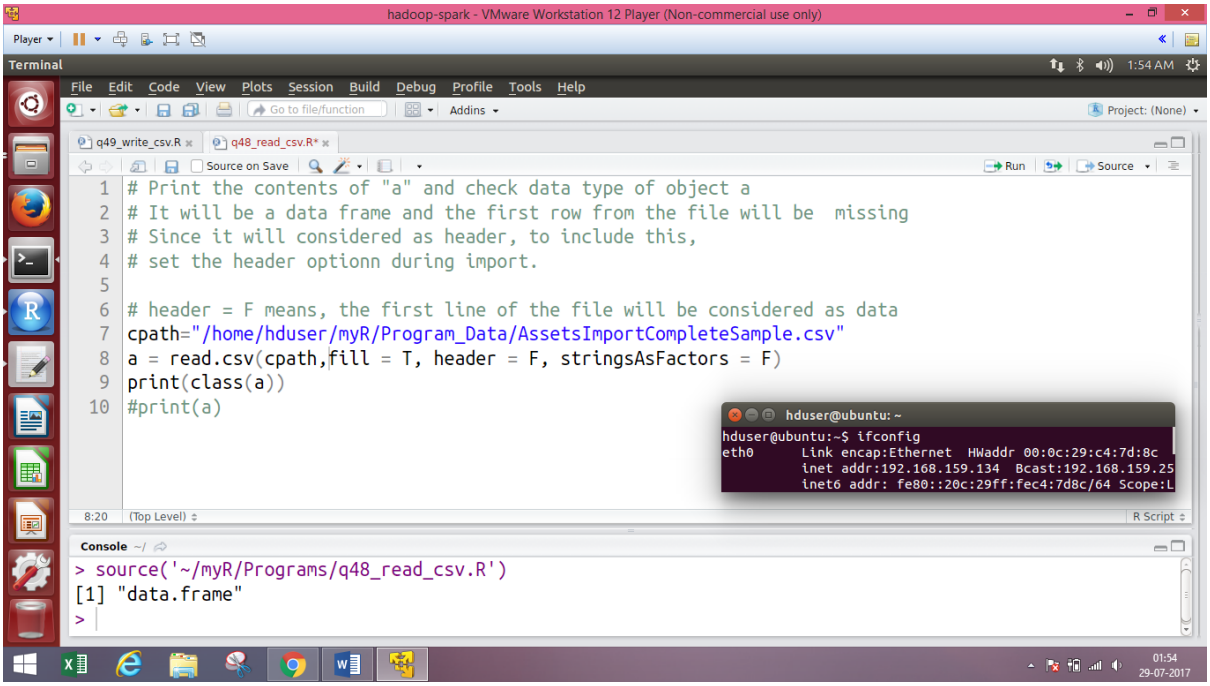


Figure 63: Class of reader variable of csv file

iii. write.csv()

1. From Object

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

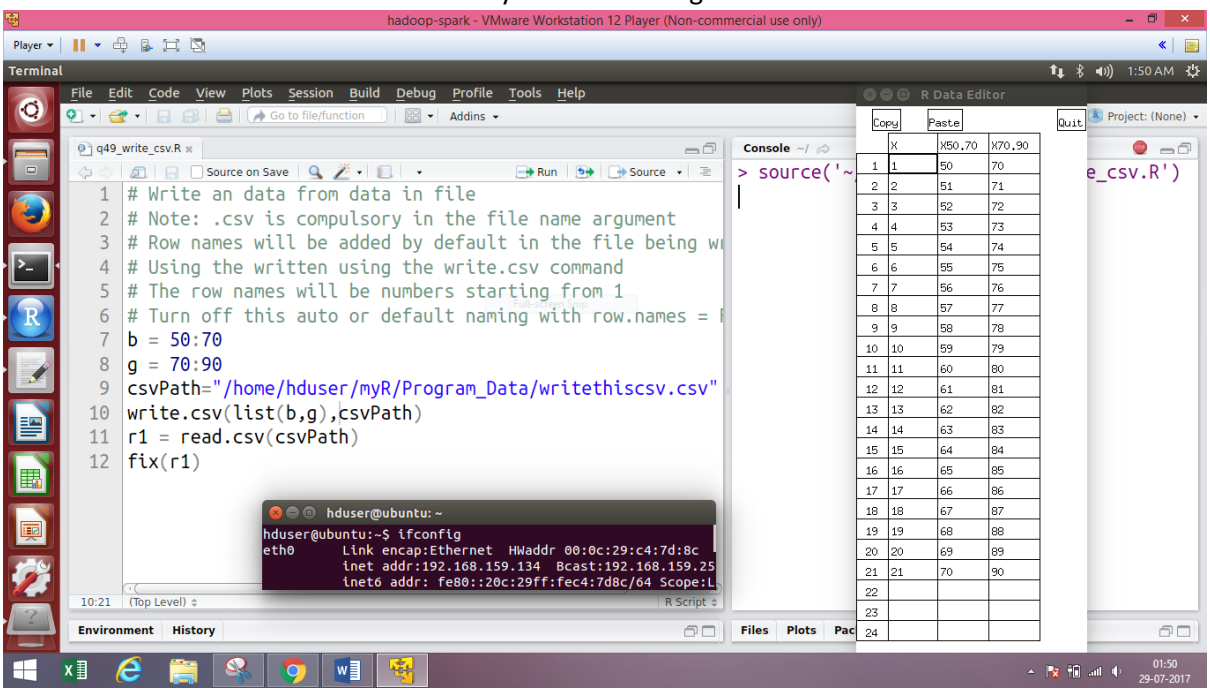


Figure 64: Write csv using object data

2. From Data Frame

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

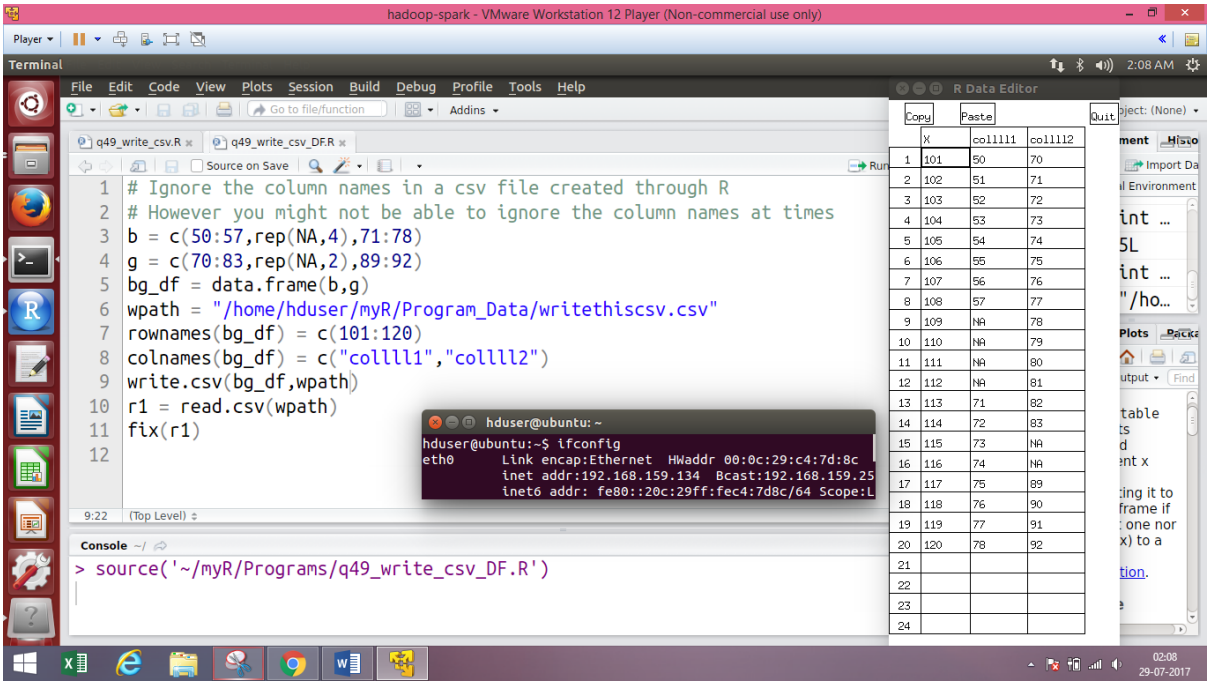
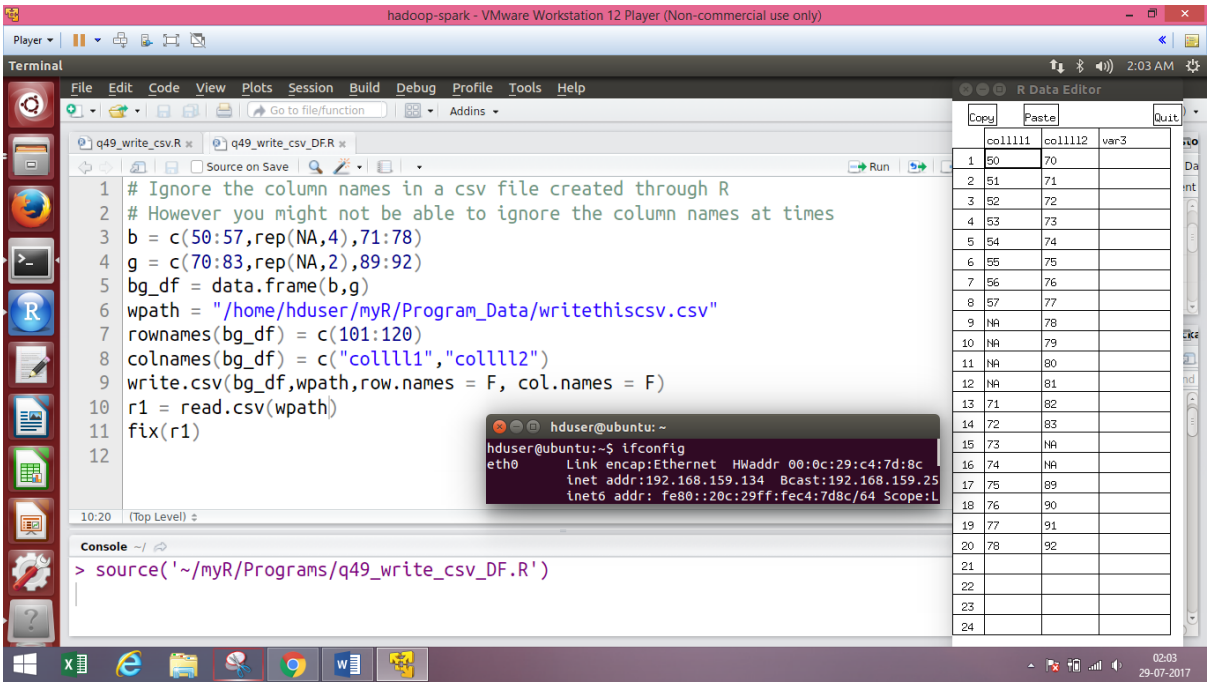


Figure 65: Write Data from Data Frame in .csv file

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.



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)

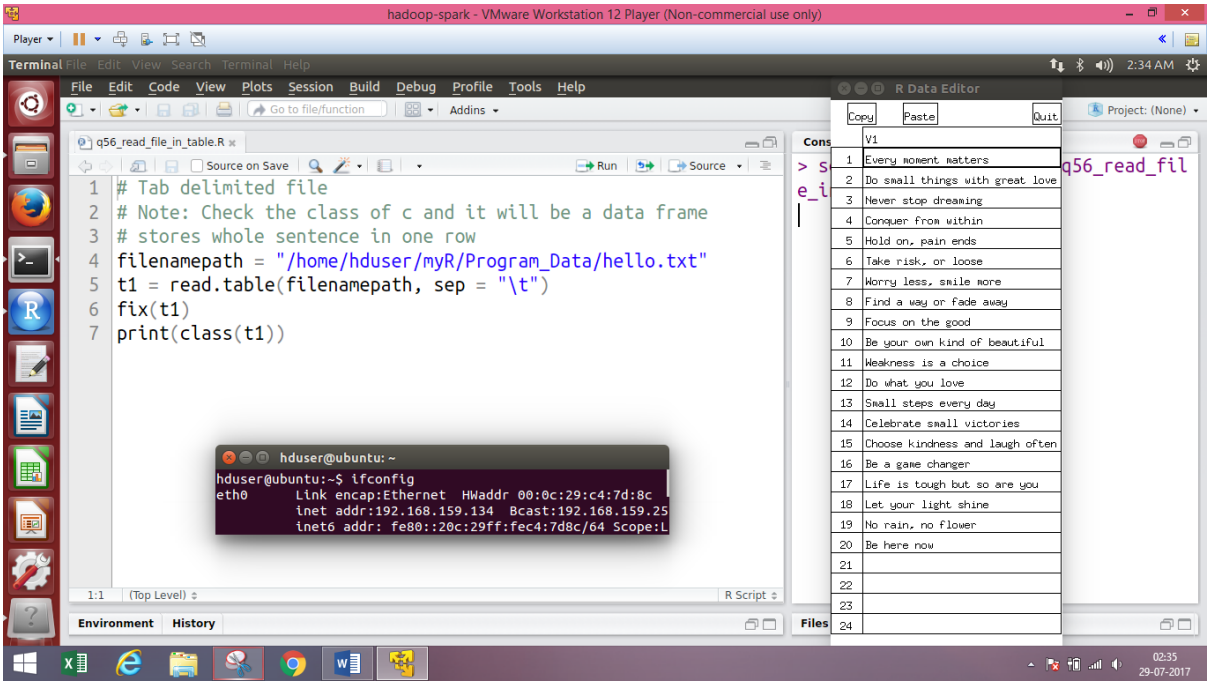


Figure 66: q56 - Read Text File

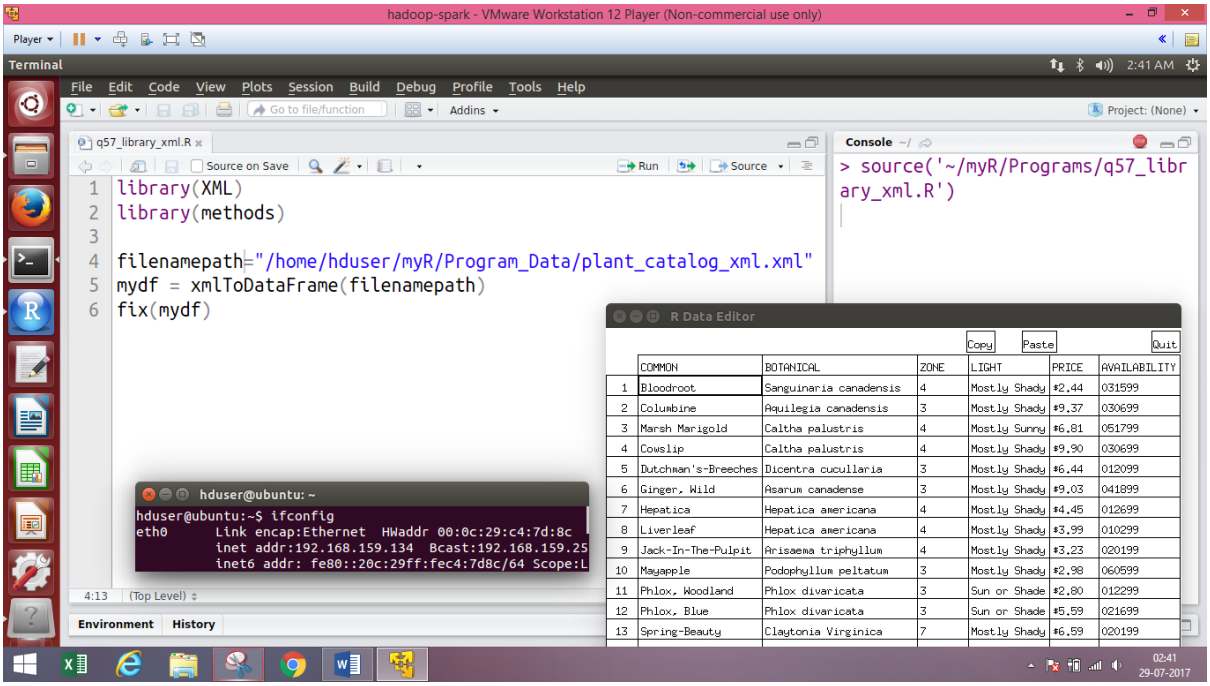
c. Import XML file

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())



	COMMON	BOTANICAL	ZONE	LIGHT	PRICE	AVAILABILITY
1	Bloodroot	Sanguinaria canadensis	4	Mostly Shady	$2.44	031599
2	Columbine	Aquilegia canadensis	3	Mostly Shady	$9.37	030699
3	Marsh Marigold	Caltha palustris	4	Mostly Sunny	$6.81	051799
4	Cowslip	Caltha palustris	4	Mostly Shady	$9.90	030699
5	Dutchman's-Breeches	Dicentra cucullaria	3	Mostly Shady	$6.44	012099
6	Ginger, Wild	Asarum canadense	3	Mostly Shady	$9.03	041899
7	Hepatica	Hepatica americana	4	Mostly Shady	$4.45	012699
8	Liverleaf	Hepatica americana	4	Mostly Shady	$3.99	010299
9	Jack-In-The-Pulpit	Arisaema triphyllum	4	Mostly Shady	$3.23	020199
10	Mayapple	Podophyllum peltatum	3	Mostly Shady	$2.98	060599
11	Phlox, Woodland	Phlox divaricata	3	Sun or Shade	$2.80	012299
12	Phlox, Blue	Phlox divaricata	3	Sun or Shade	$5.59	021699
13	Spring-Beauty	Claytonia Virginica	7	Mostly Shady	$6.59	020199

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)
```

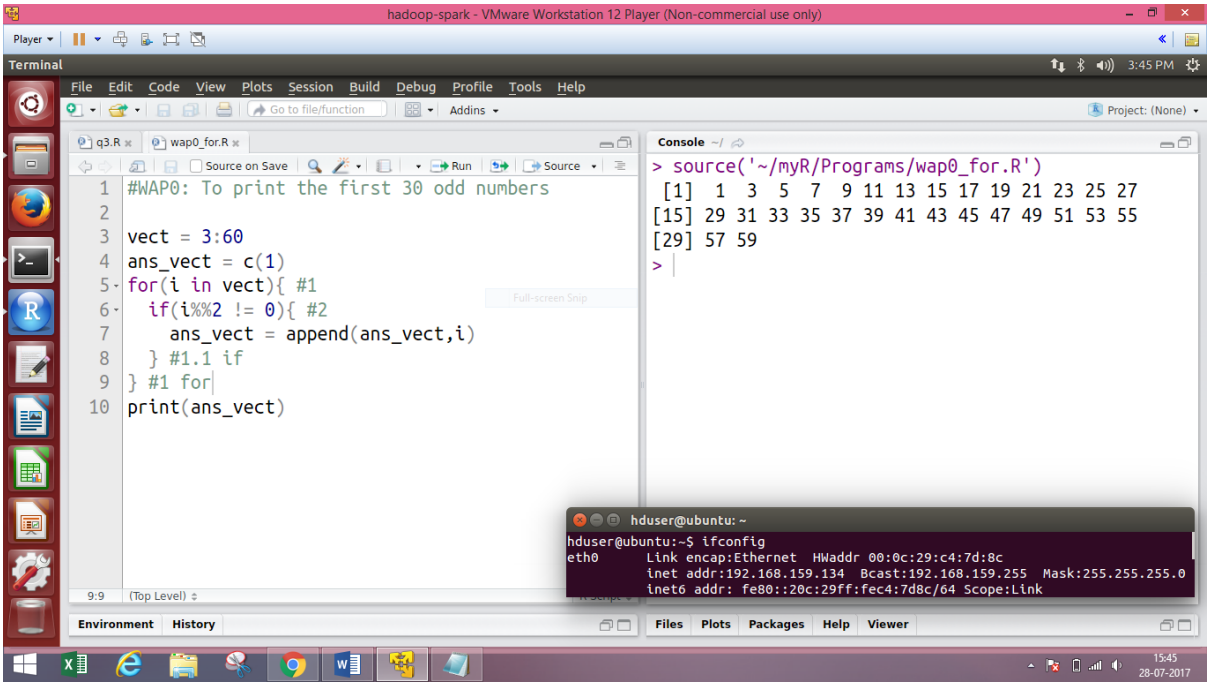


Figure 68: WAP 0 : For Loop

ii. Modify the repeat code to make infinite loop

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

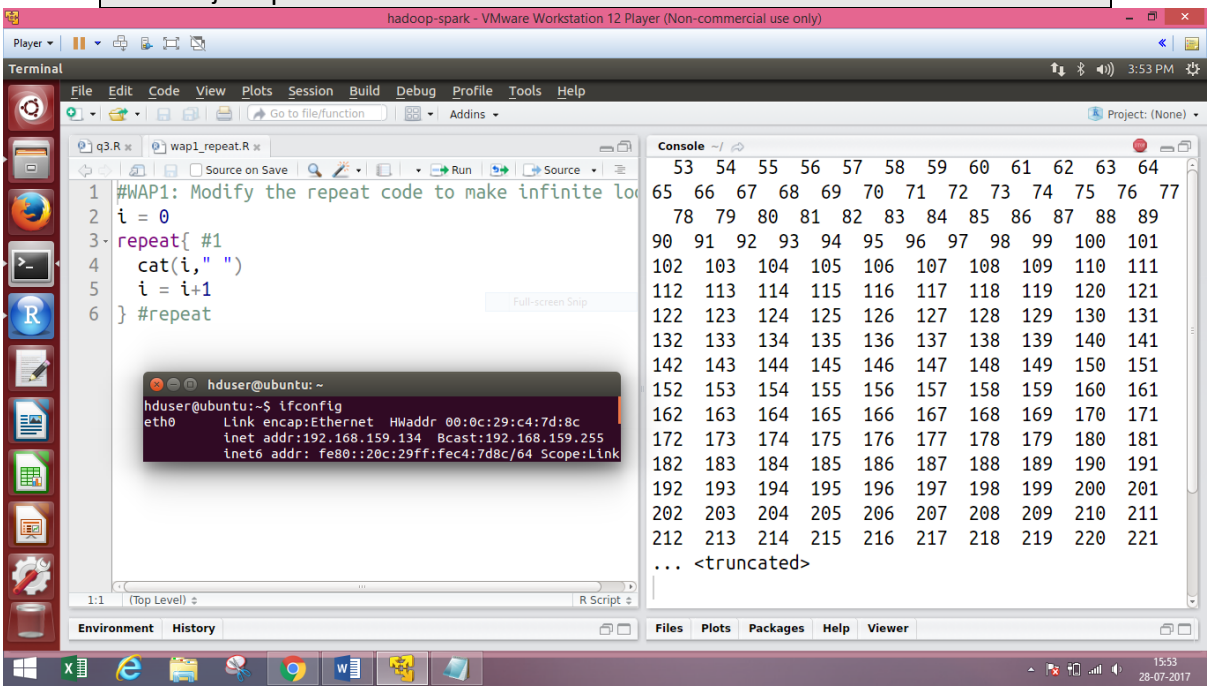


Figure 69: WAP 1 Repeat Loop

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)
```

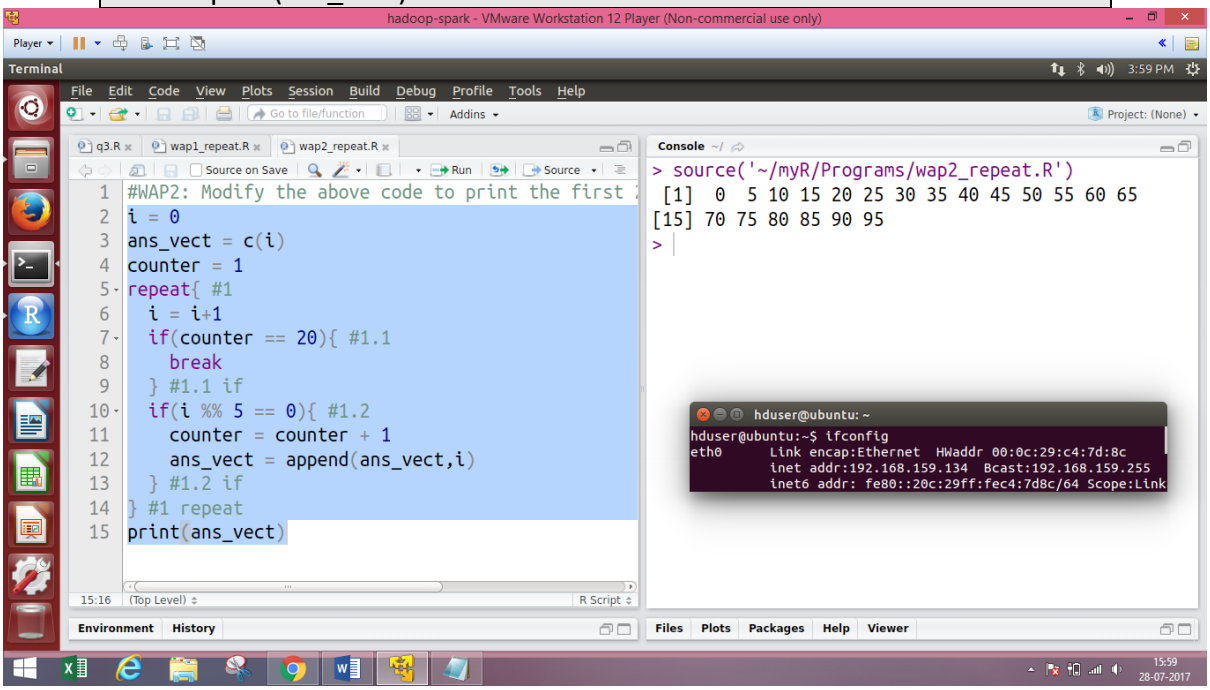


Figure 70: WAP 2 Repeat

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)
```

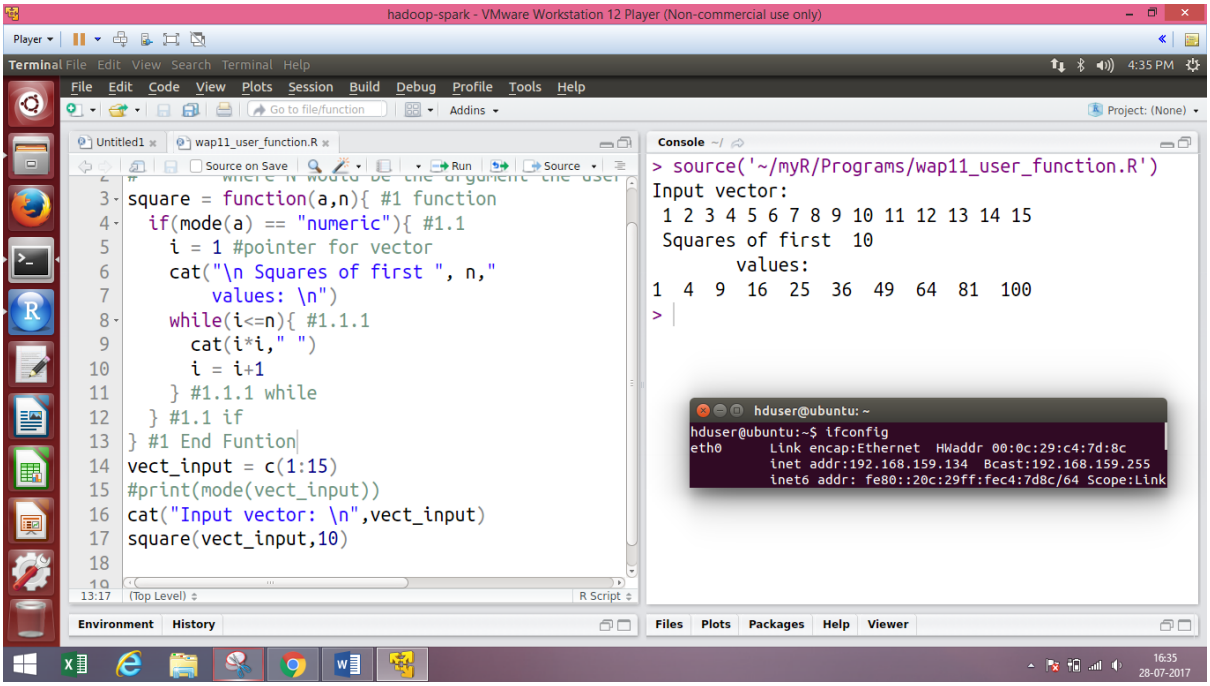


Figure 71: WAP 11 Function

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))
```

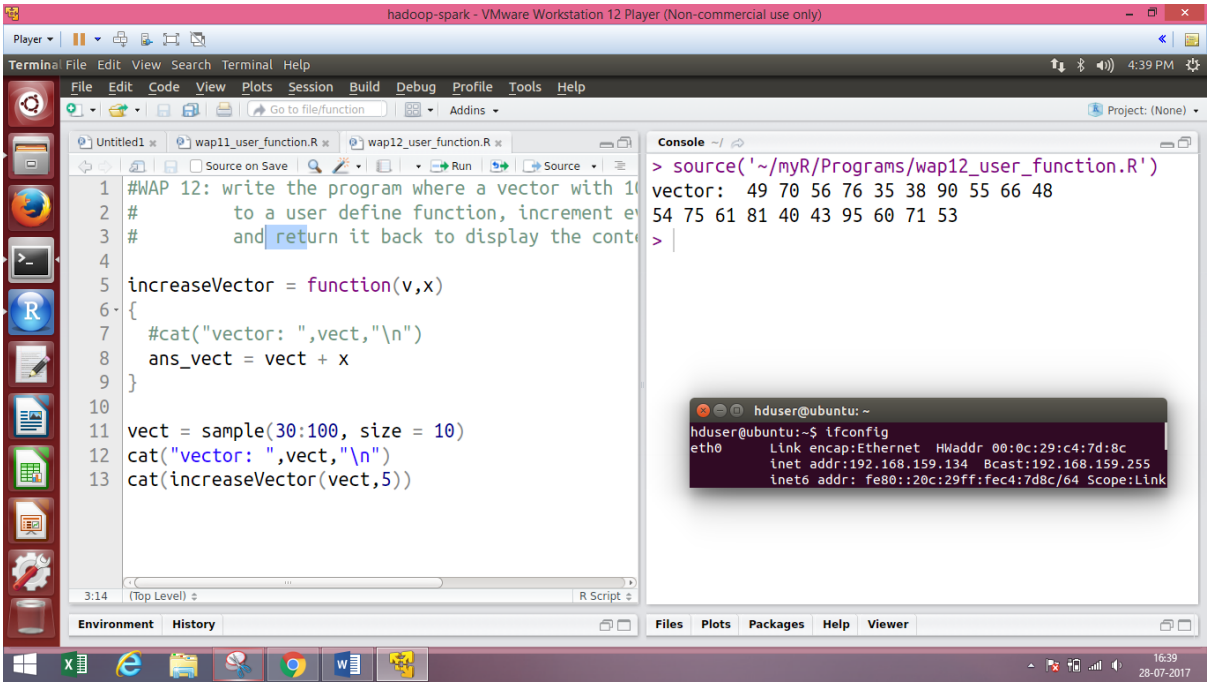


Figure 72: WAP 12 function

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))
```

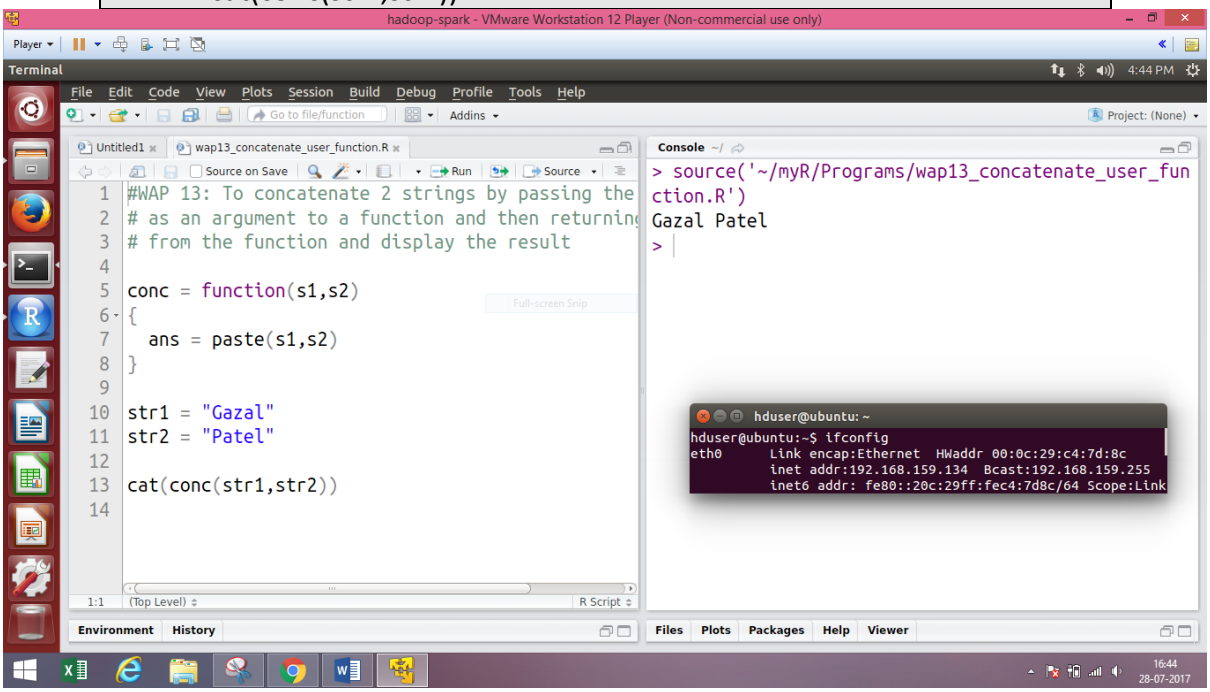


Figure 73: WAP 13 function

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))
```

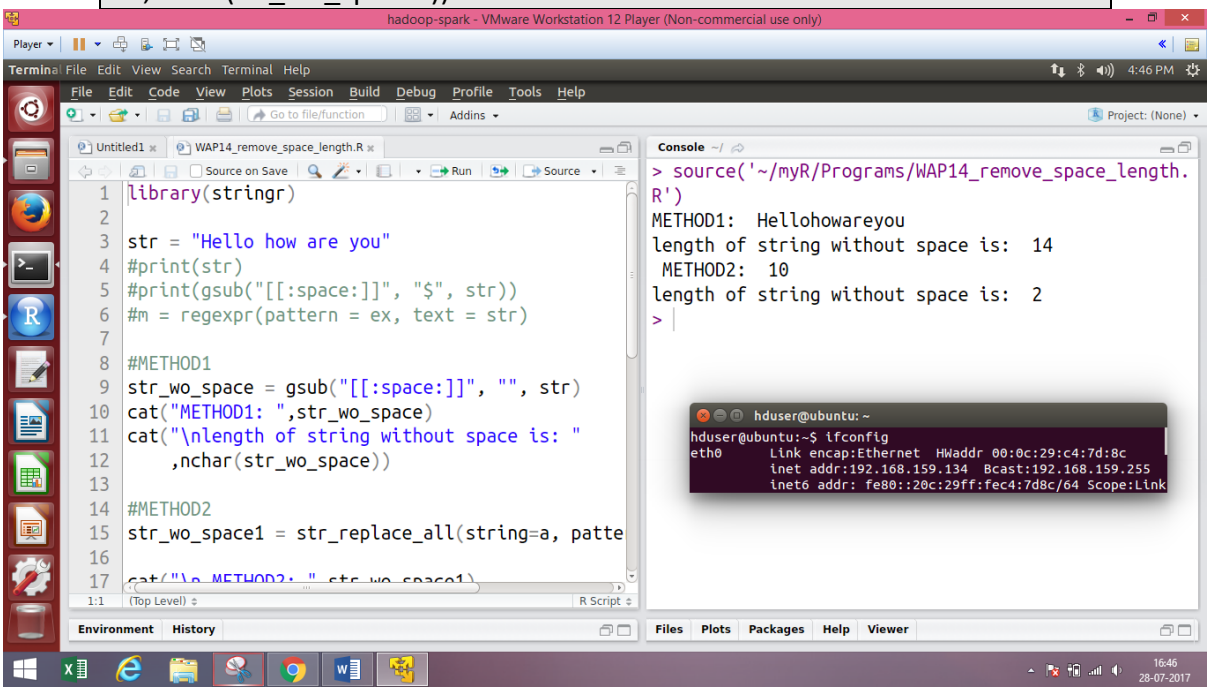


Figure 74: WAP 14 String

viii. WAP to count space in string.

```
CountOccurence = 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 =CountOccurence(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:
",CountOccurence(s2,p2))
```

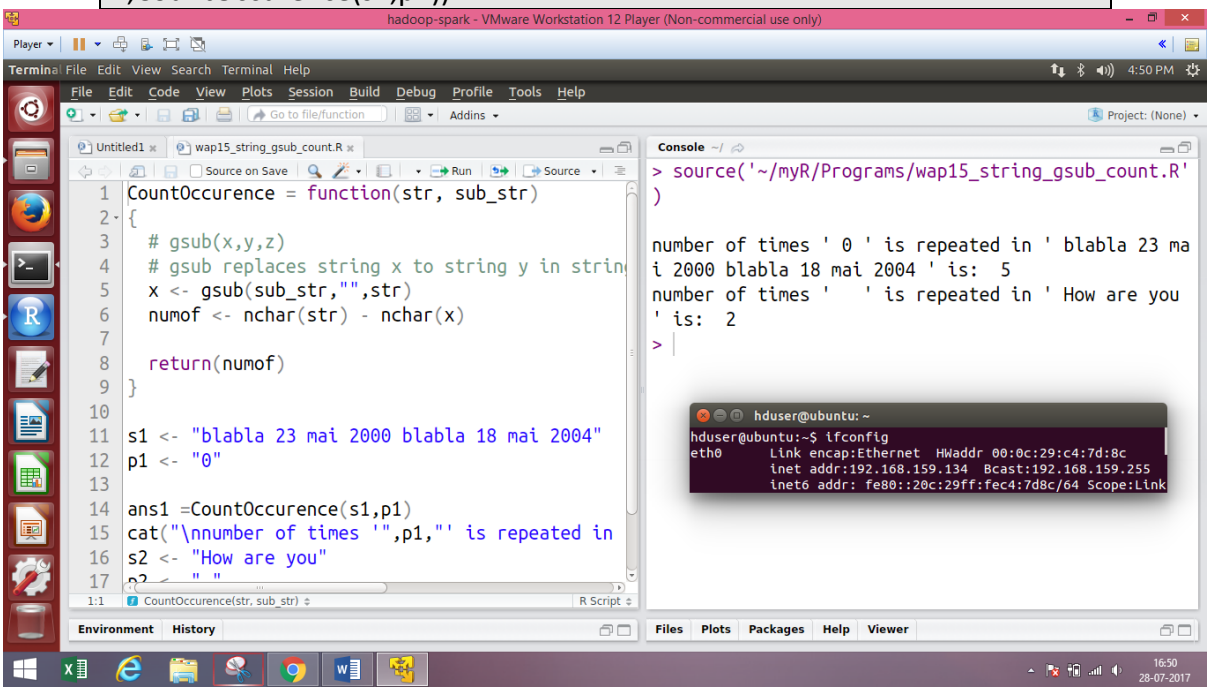


Figure 75: WAP 15 String count

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))
```

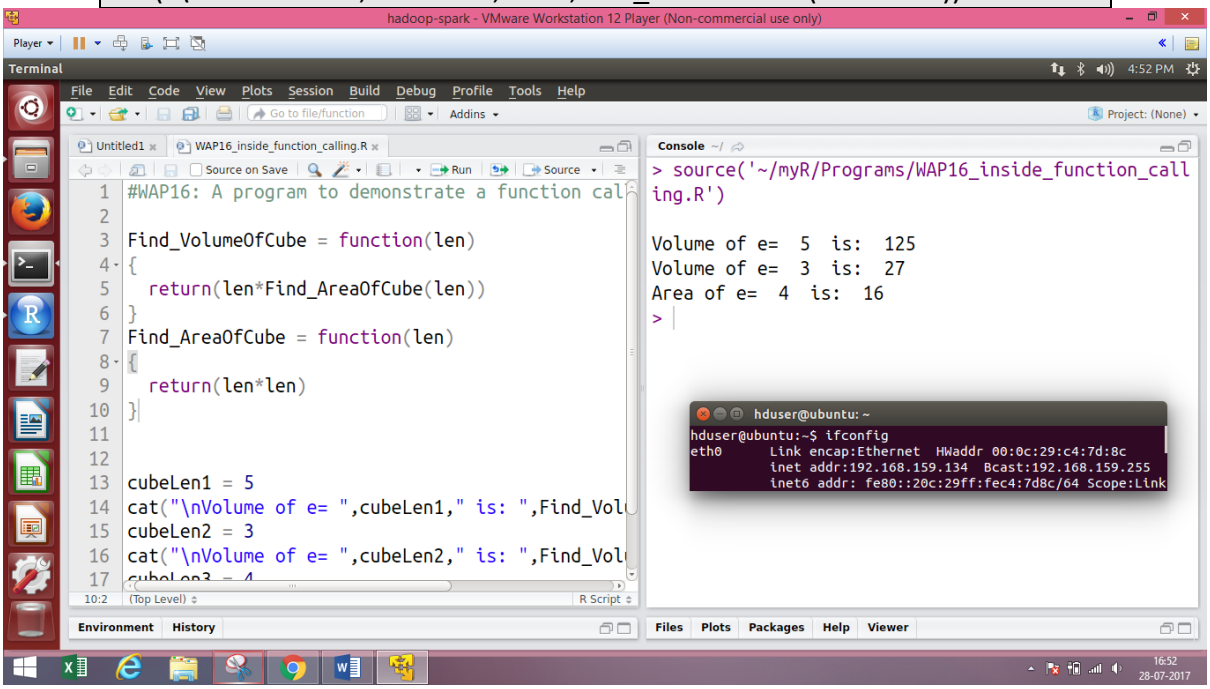


Figure 76: WAP 16 inside Function Calling

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)
```

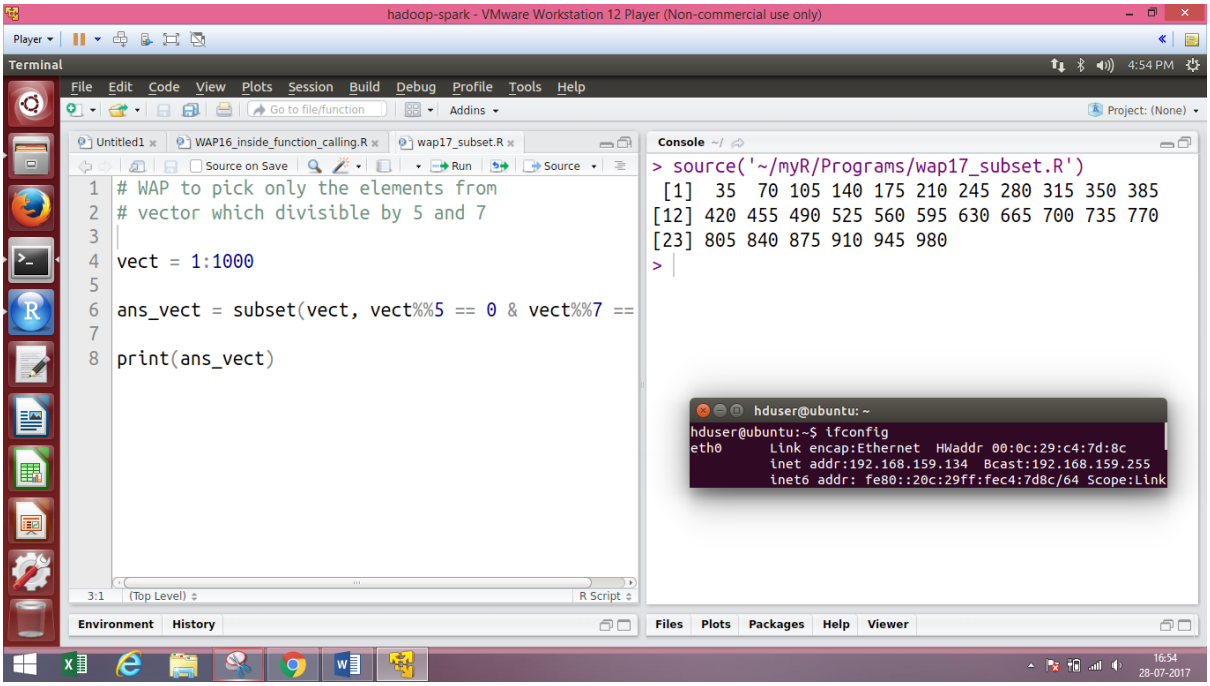


Figure 77: WAP 17 subset