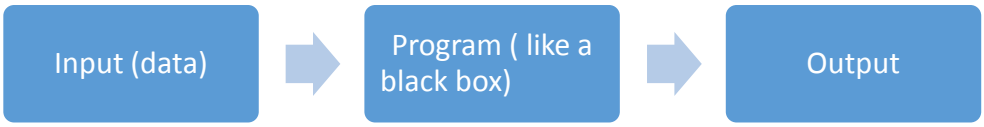
Basic R Programming

1. Program

- It is required have some data / input for the program.
- Input is set of instructions to work on the data.



2. Variable (Objects)

- A variable is storage for values inside R-Object that our programs can manipulate.
- A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number. For Example: var1, var_1, .var1, vAr1.
- The variables can be assigned values using leftward, rightward and equal to operator.

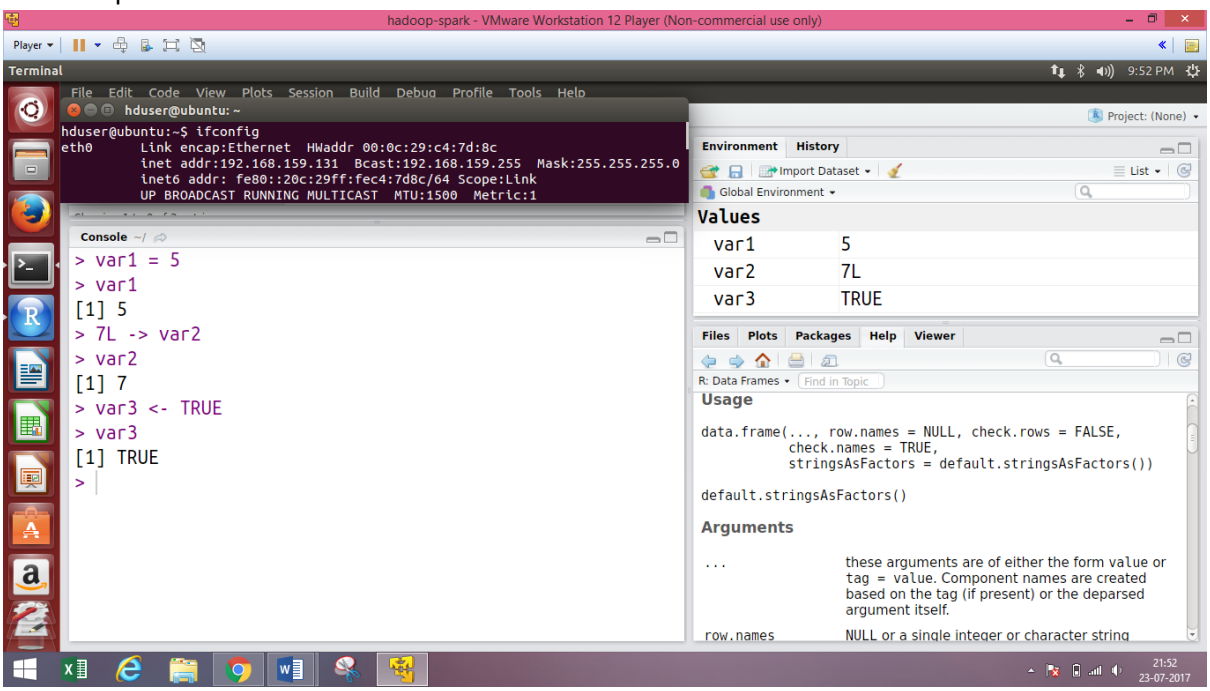


Figure 1: Variable Assignment

- These values can be Character, Numeric, Integer, Complex, Logical and Raw **Data Type**. These are called as **Class** or **Mode**. These data can also be stored in complex – atomic vector variables such as Vectors, Lists, Matrices, Arrays, Factors and Data Frames.
    i. **Logical**: It can be either TRUE or FALSE

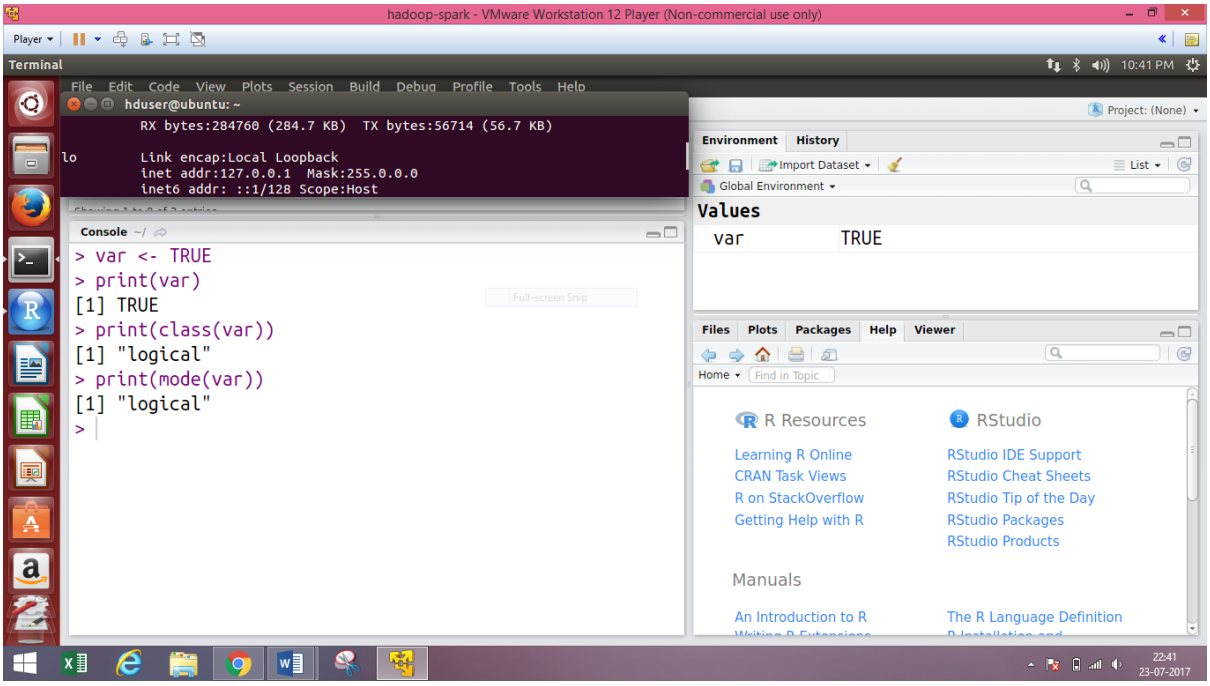


Figure 2: Logical Object

ii. **Numeric**: It is any numeric number. It is root class for Integer and Decimal numbers.

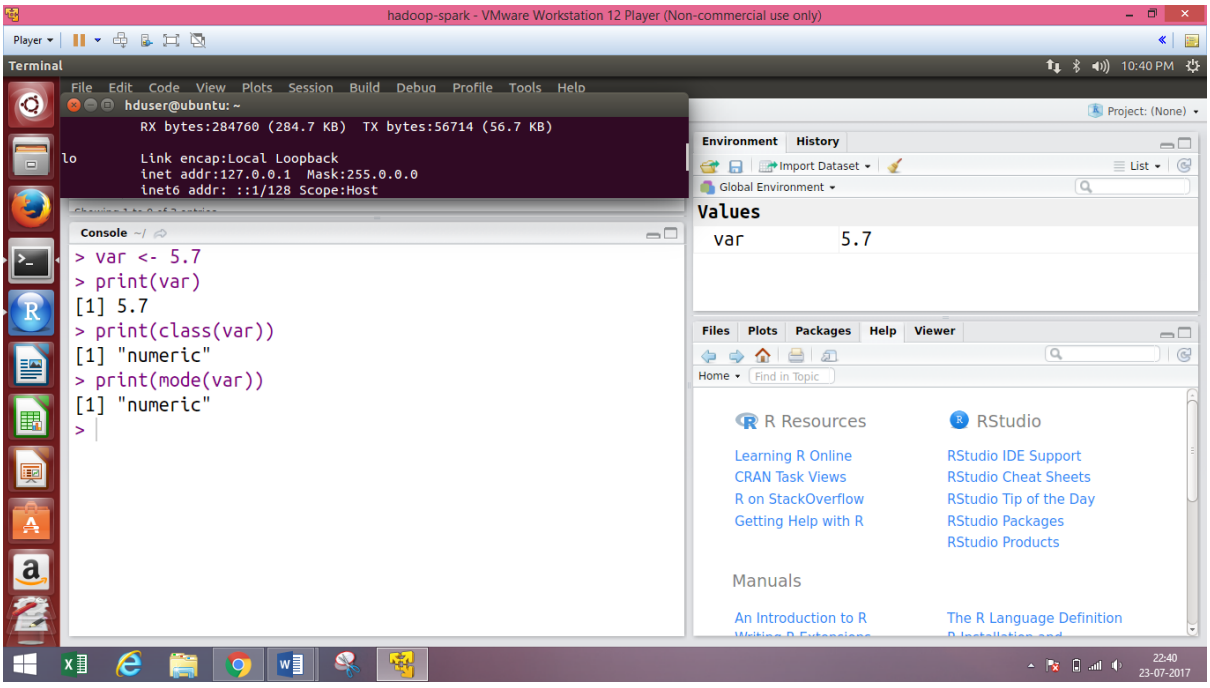


Figure 3: Numeric Object

iii. **Integer**: It is strictly whole number. This can be positive, negative or zero. It is subclass of Numeric class. It is initiated with number preceding to L. For example: 5L

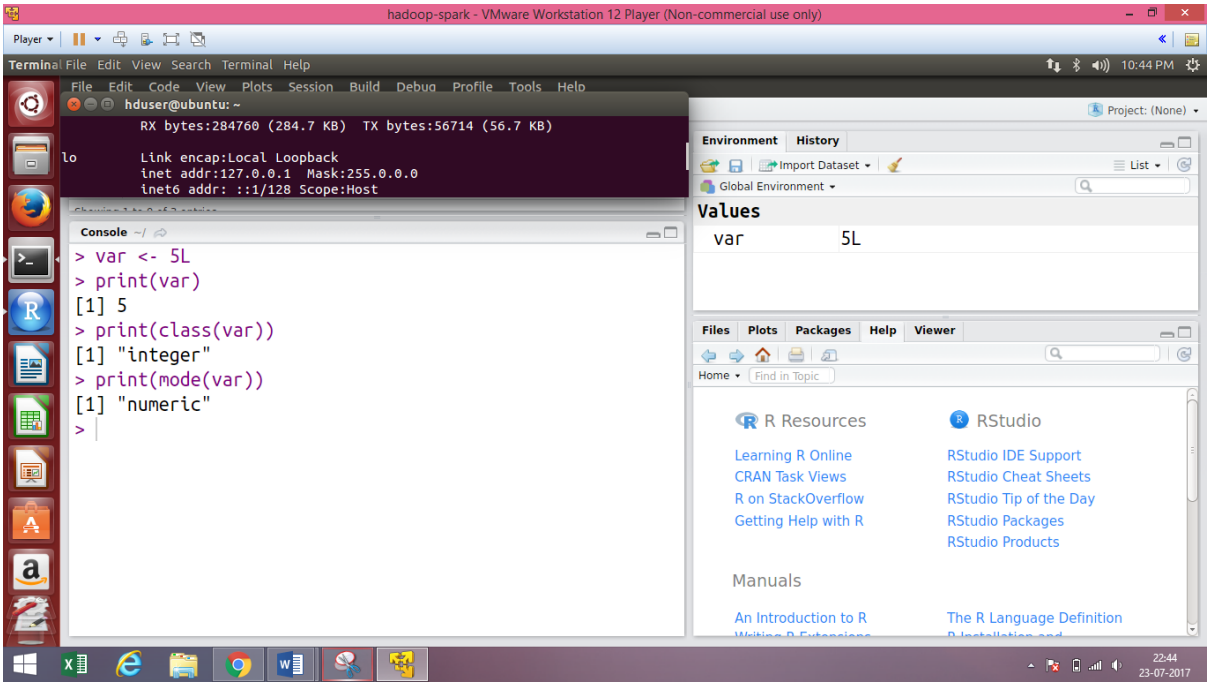


Figure 4: Integer Object

iv. **Complex**: It is complex numeric number. For example: 5 + 7i

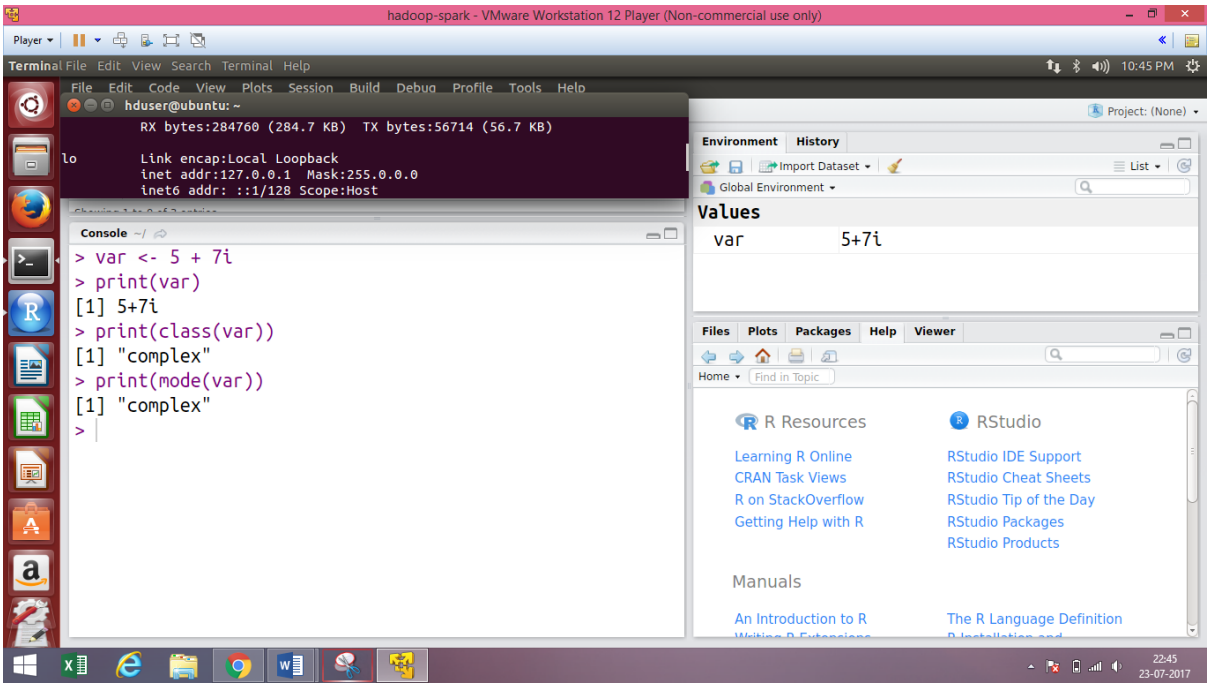


Figure 5: Complex Object

v. **Character**: It is any character or string variable.

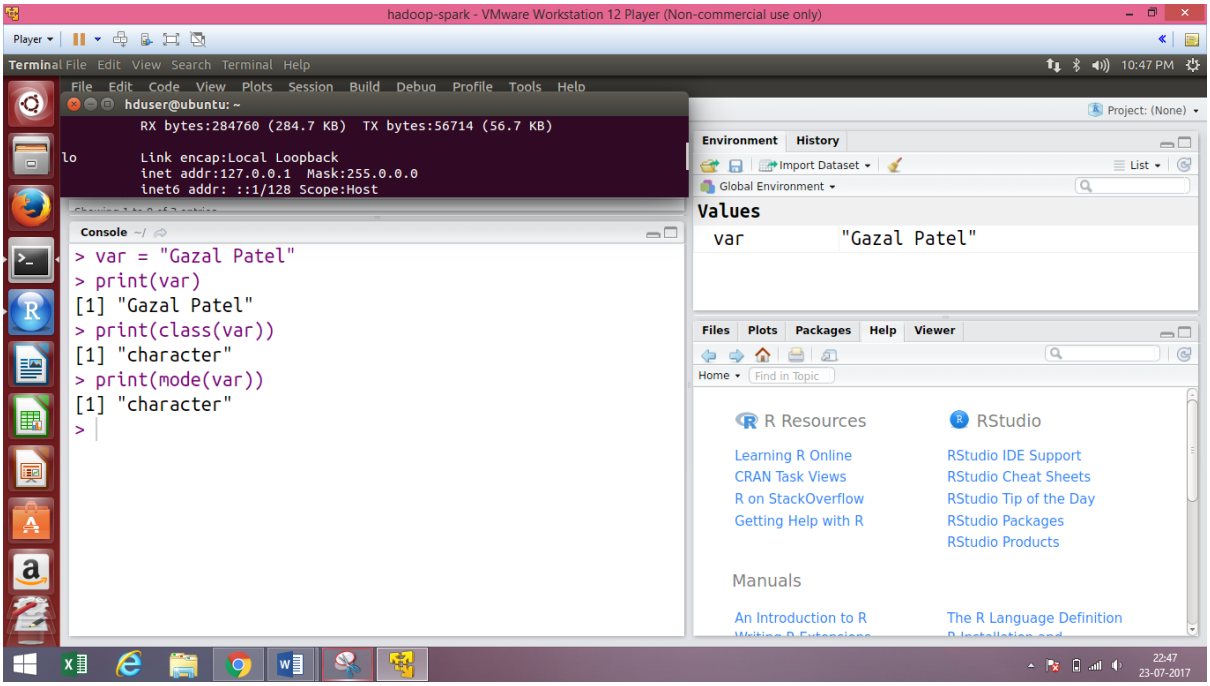


Figure 6: Character Object

vi. **Raw**: The raw type is intended to hold raw bytes. It is possible to extract sub-sequences of bytes, and to replace elements (but only by elements of a raw vector). A raw vector is printed with each byte separately represented as a pair of hex digits. Coercion to raw treats the input values as representing small (decimal) integers, so the input is first coerced to integer, and then values which are outside the range [0 ... 255] or are NA are set to 0 (the **nul** byte). For Example, "Hello" is stored as 48 65 6c 6c 6f

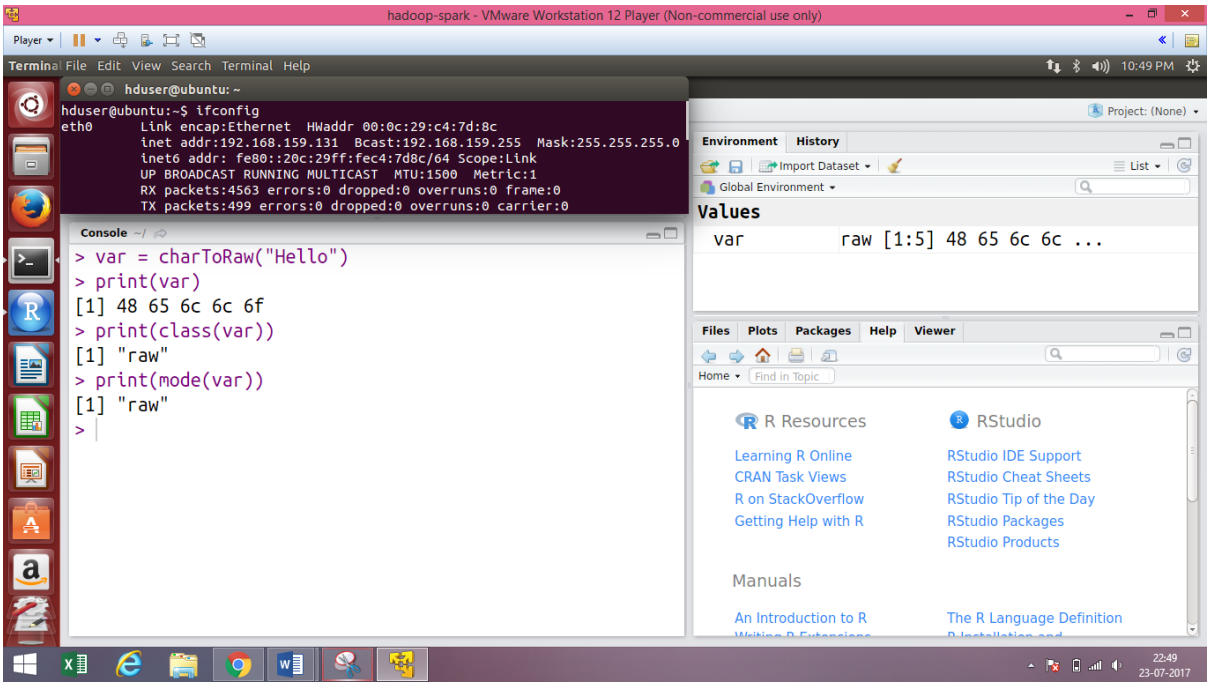


Figure 7: Raw Object

vii. **Vectors**: It is collection of more than one data. Data can be of any datatype. Use **c()** function to create vector. Also See SEQ

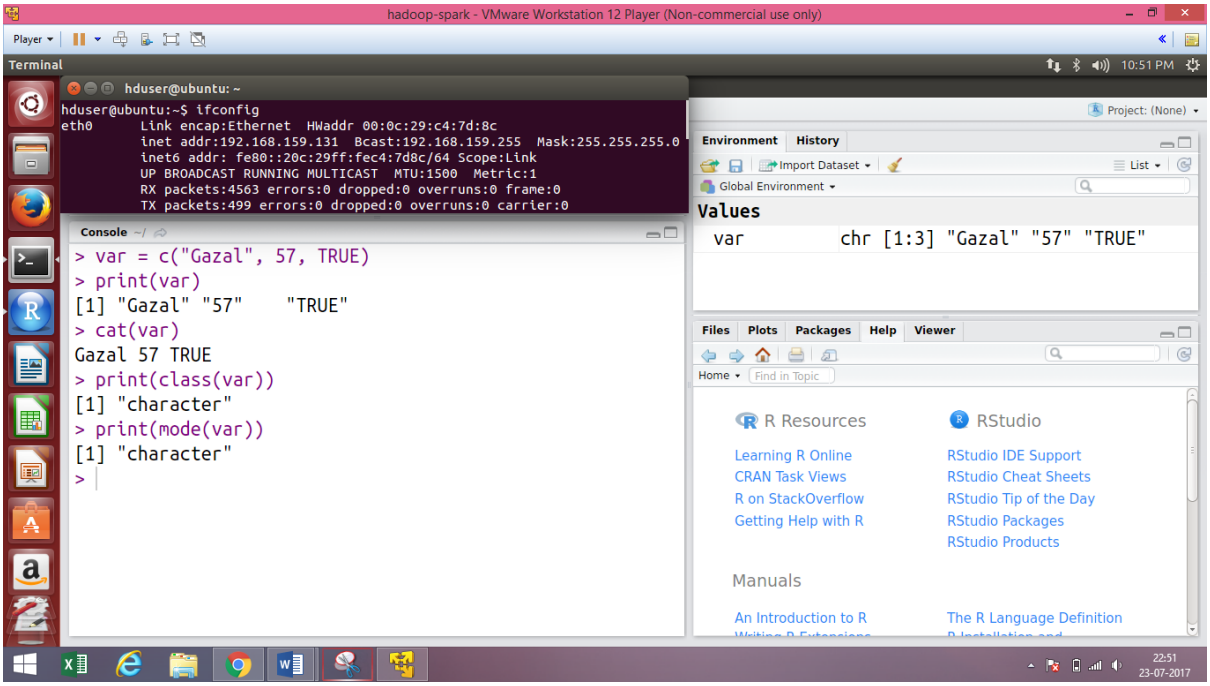


Figure 8: Vector

viii. **Lists**: A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

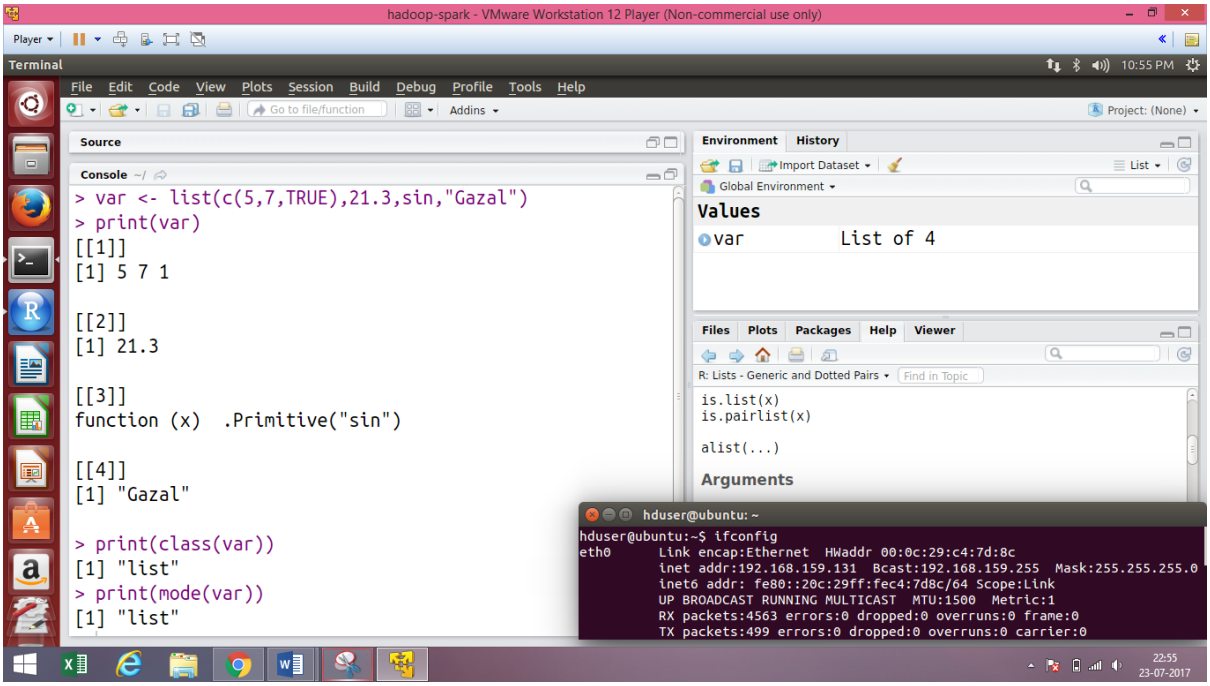


Figure 9: List

ix. **Matrix**: A matrix is a two-dimensional rectangular data set. It is confined to two dimensions. It can be created using a vector input to the **matrix()** function.
Also see Difference with Array.

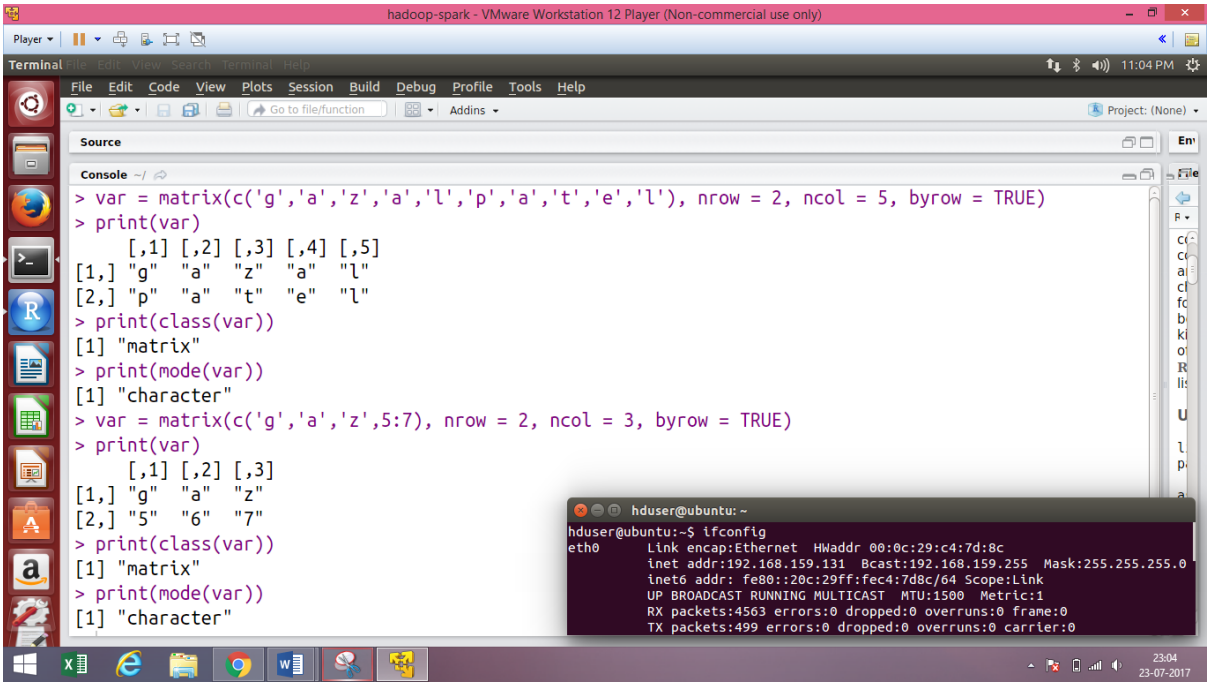


Figure 10: Matrix

x. **Array**: Arrays can be seen as matrix but consisting of any number of dimensions. The **array()** function takes a 'dim' attribute which creates the required number of dimension. Attribute 'dim' holds vector of three: total rows, columns and repetition of data-vector to fill array.
Difference with Matrix here is that in matrix number of data elements to be filled should be multiple of rows and columns number. On the other hand array does repetition of data until for required data space.

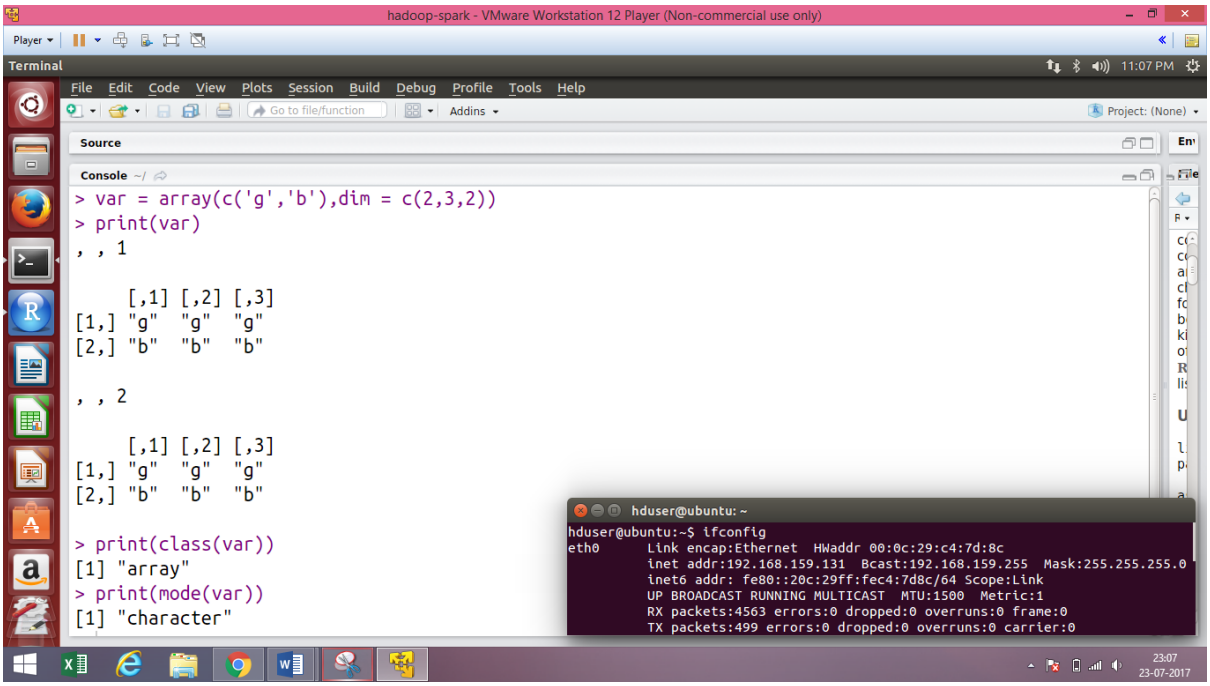


Figure 11: Arrays

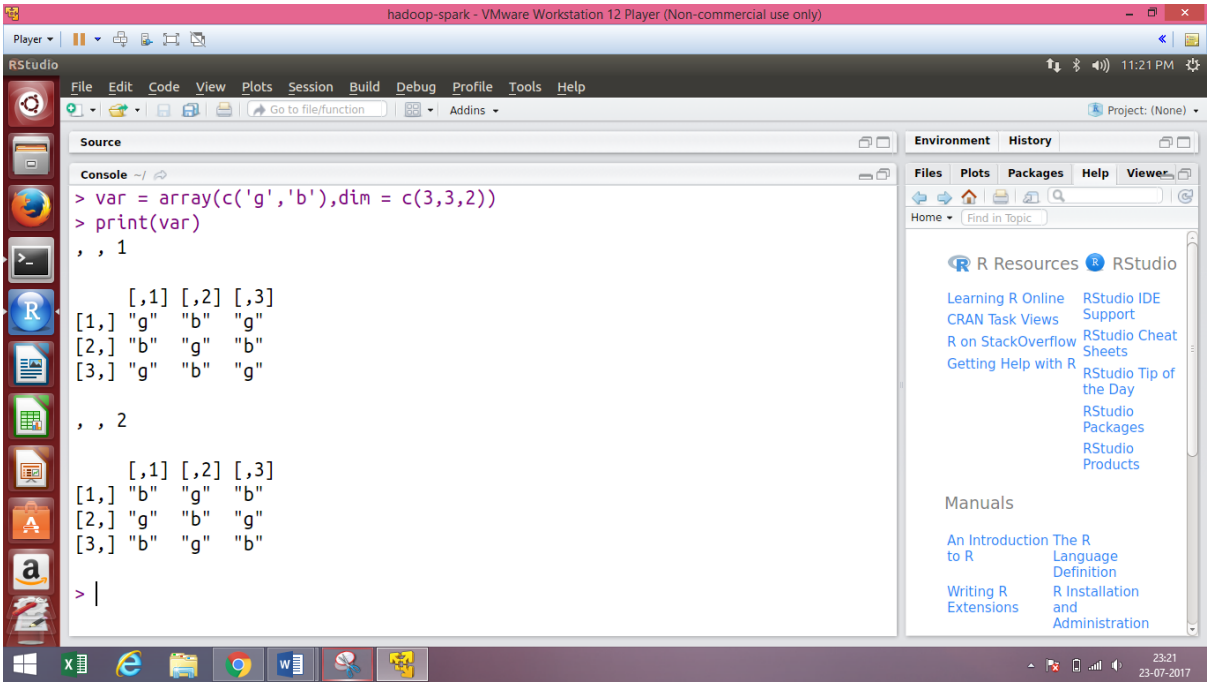


Figure 12: Array Explained

xi. **Factor**: It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. It is created using the **factor()** function. The **nlevels()** functions gives the count of levels.

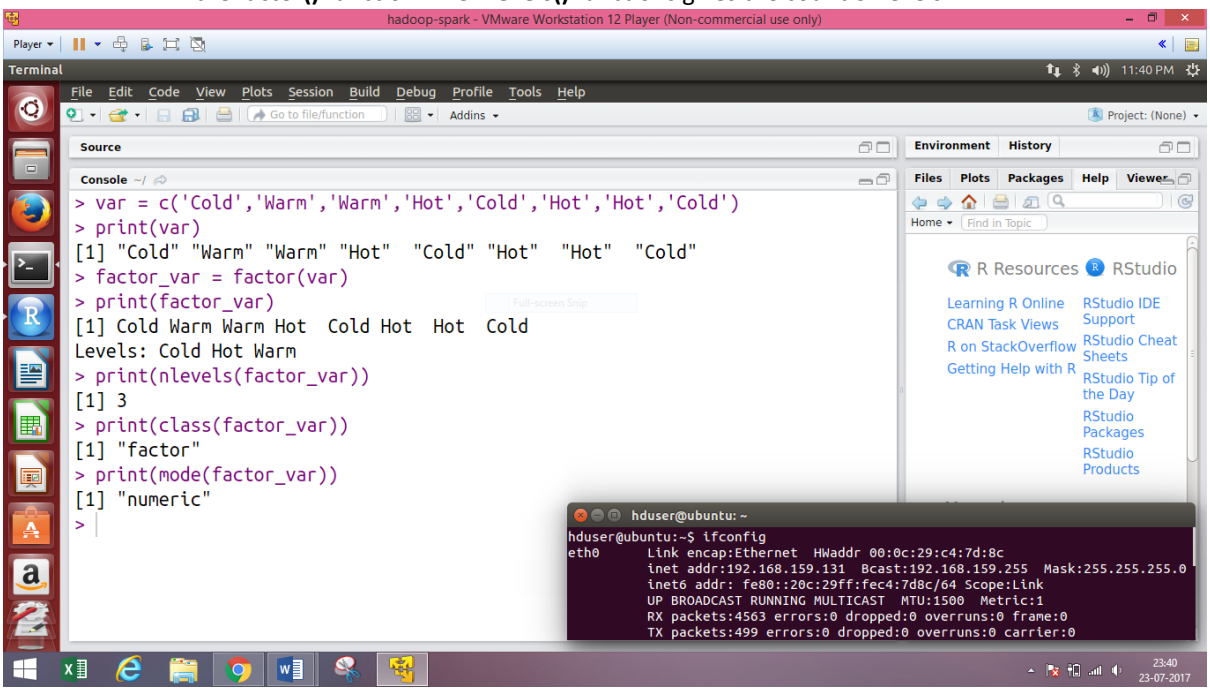


Figure 13: Factor

xii. **Data Frame:** Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length. Data Frames are created using he **data.frame()** function.

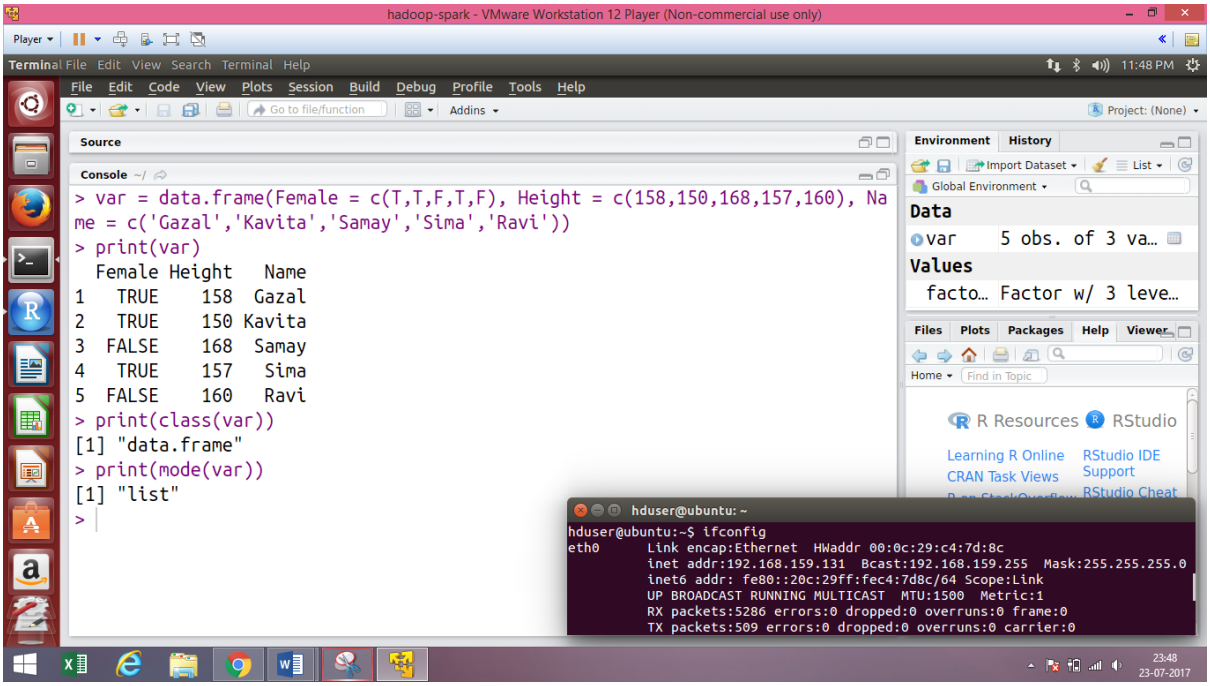


Figure 14: Data Frame

- Printing Variable can be done by **print()** or **cat()** functions.

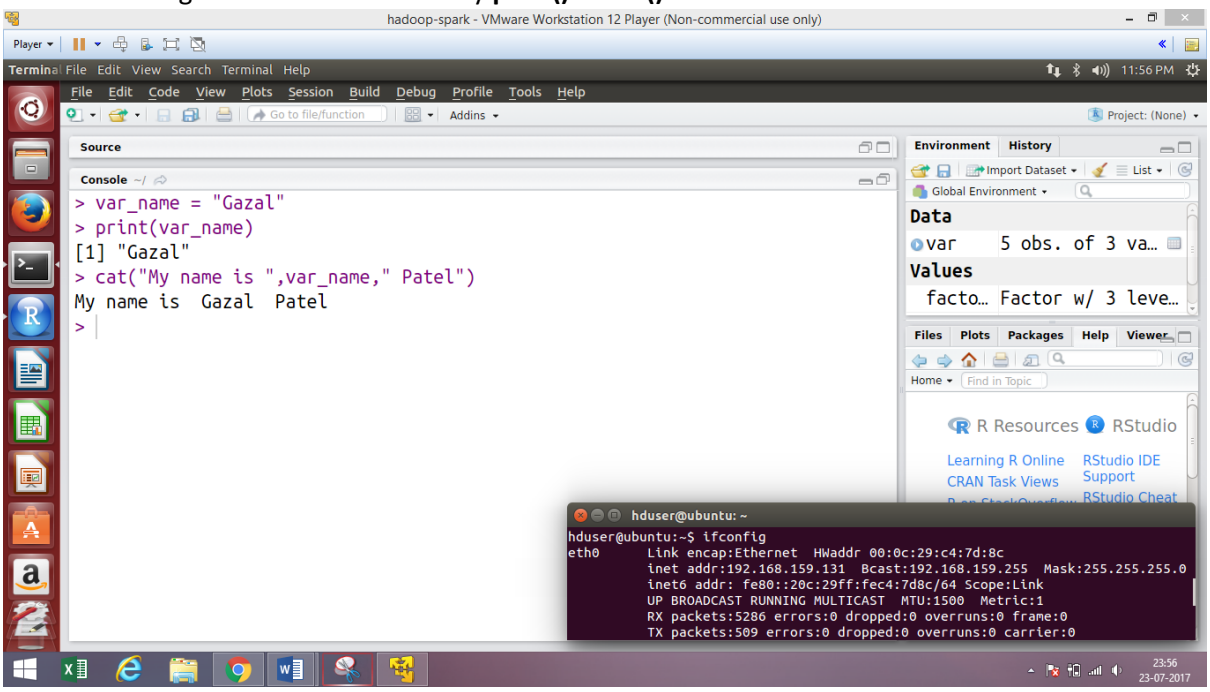


Figure 15: print() and cat()

- Search Variables
    i. **ls()** : Print all currently available variables of the current workspace

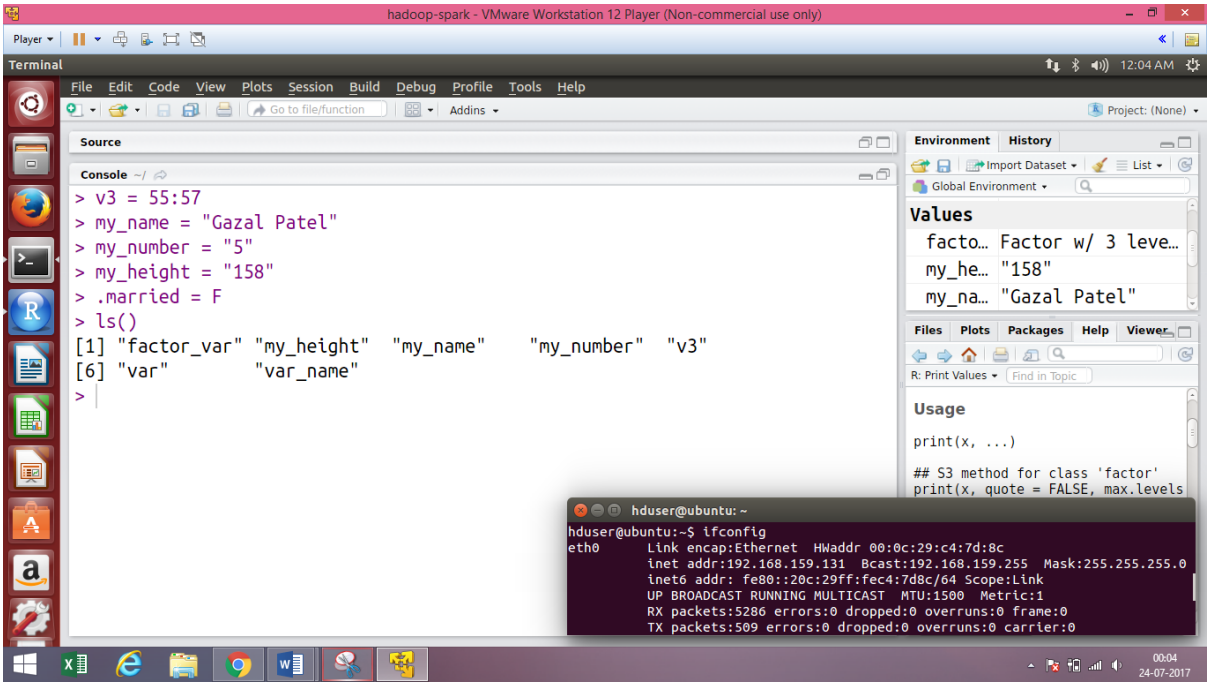


Figure 16: ls()

ii. **ls() for pattern** : ls() function can use patterns to match the variable names.

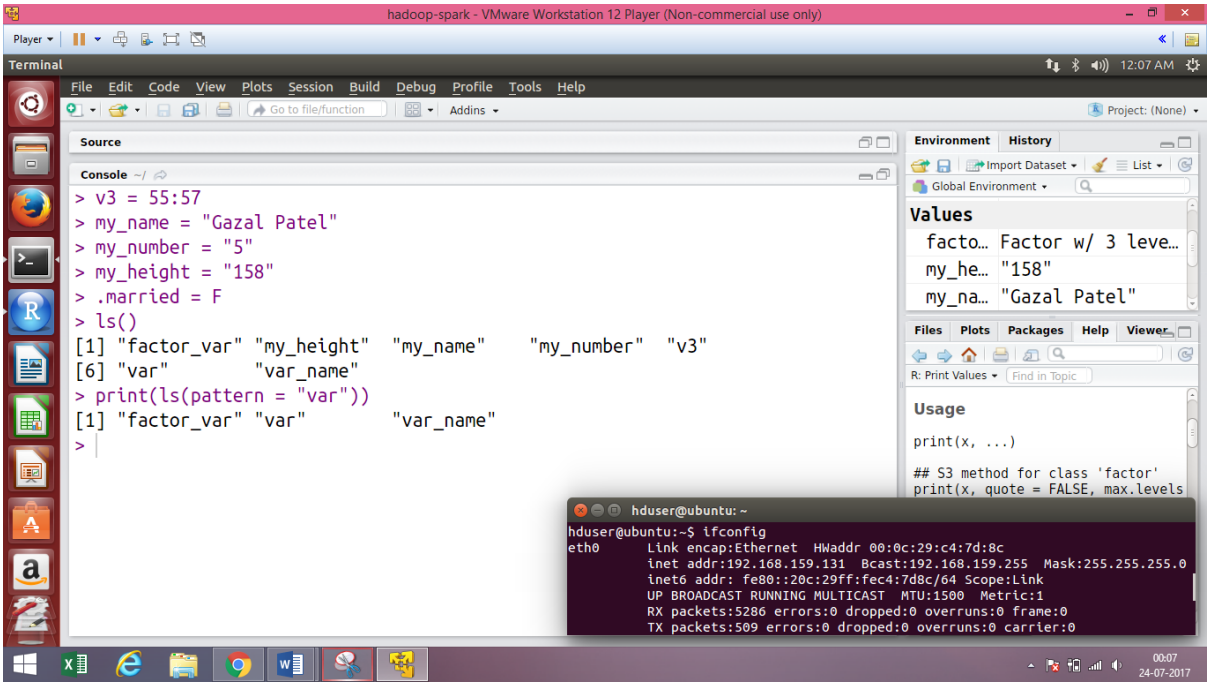


Figure 17: ls() for Pattern

iii. **ls() for Hidden Variable** : The variables starting with dot(.) are hidden, they can be listed using "all.names = TRUE" argument to ls() function.

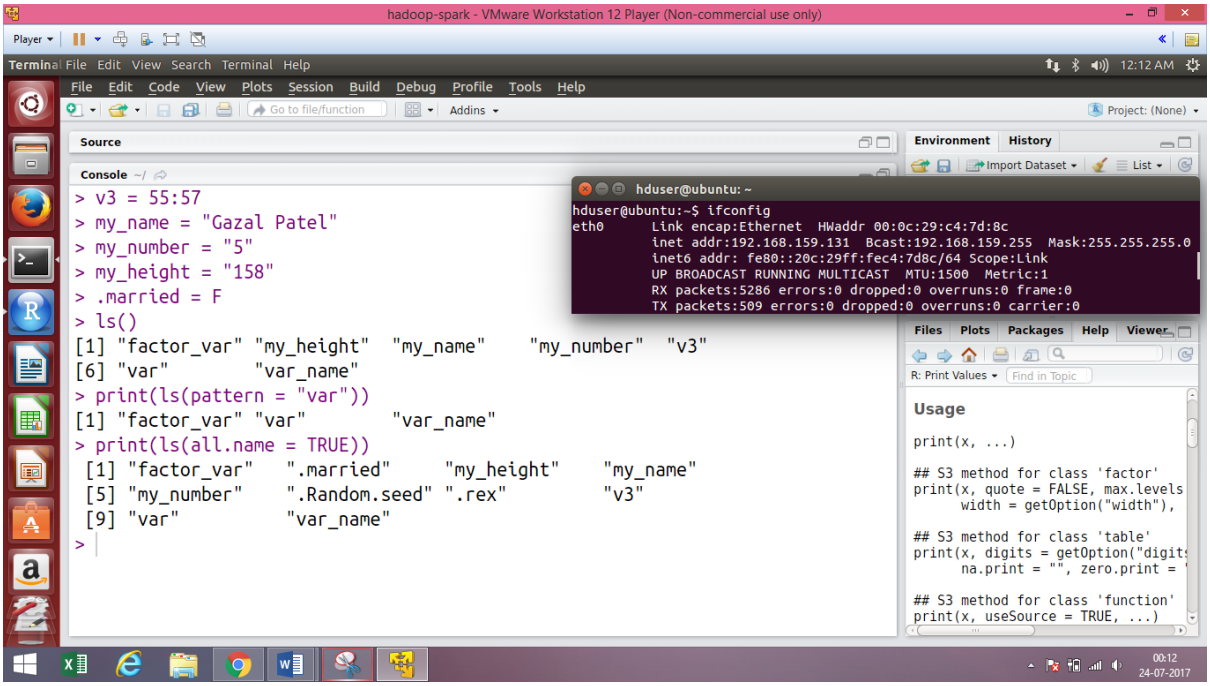


Figure 18: ls() for hidden variable

- Delete Variable with **rm()** :
  i. Single Variable: Use single variable as parameter to the **rm()** function to delete single variable.

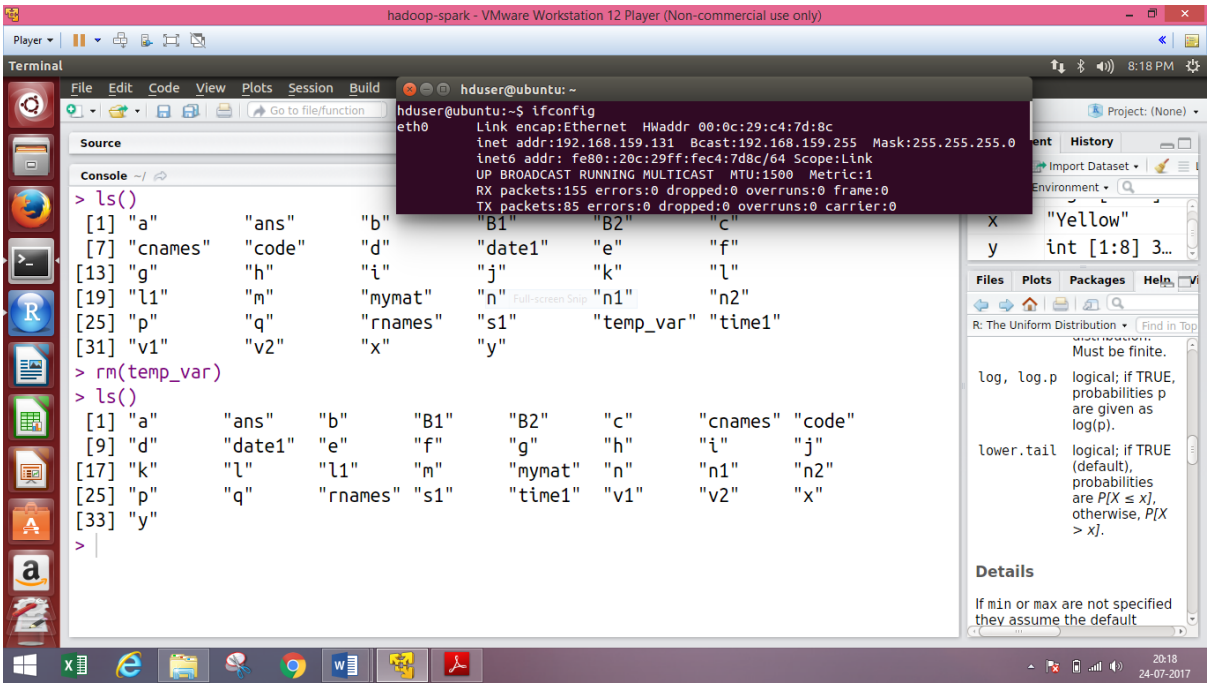


Figure 19: rm() function

ii. Multiple Variable: Use vector having variable as a parameter to **rm()** function to delete them.

iii. All Variables: Use **list** created by **ls()** function as a parameter to **rm()** function to delete all the variables from workspace except for hidden variables.

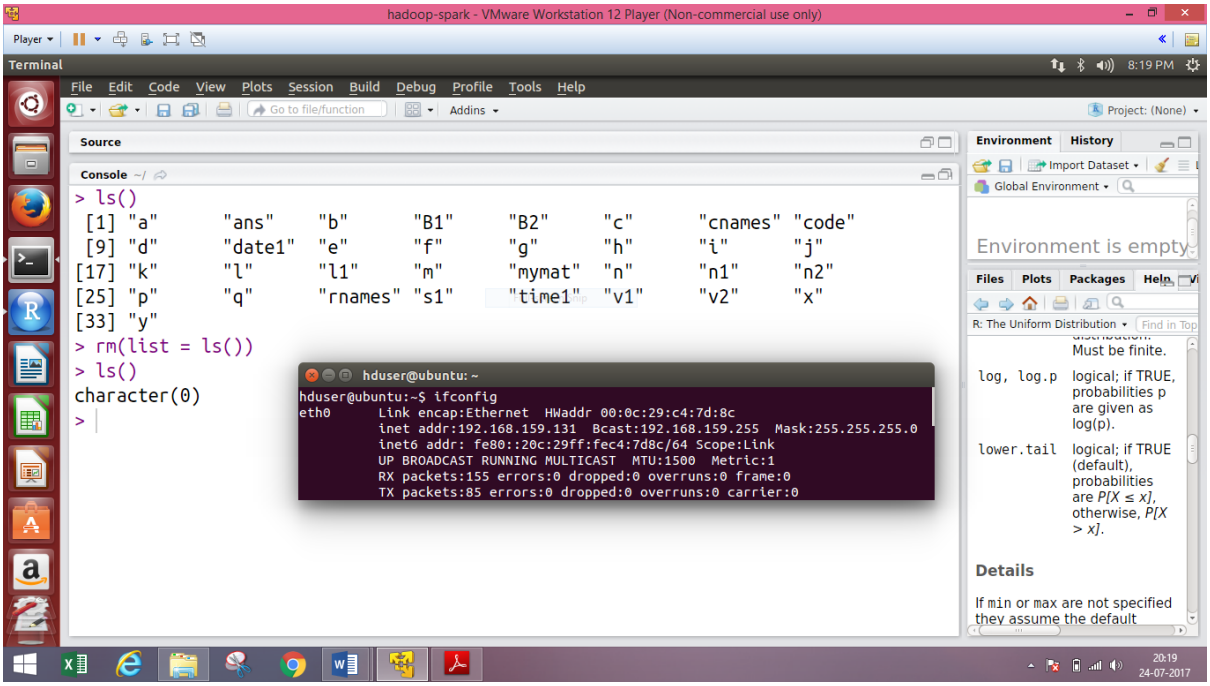


Figure 20: rm all variables in workspace

3. Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

- Arithmetic Operators

+	Adds two vectors
-	Subtracts second vector from the first
*****	Multiplies both vectors
/	Divide the first vector with the second
%%	Give the remainder of the first vector with the second
%/%	The result of division of first vector with second (quotient)
^	The first vector raised to the exponent of second vector

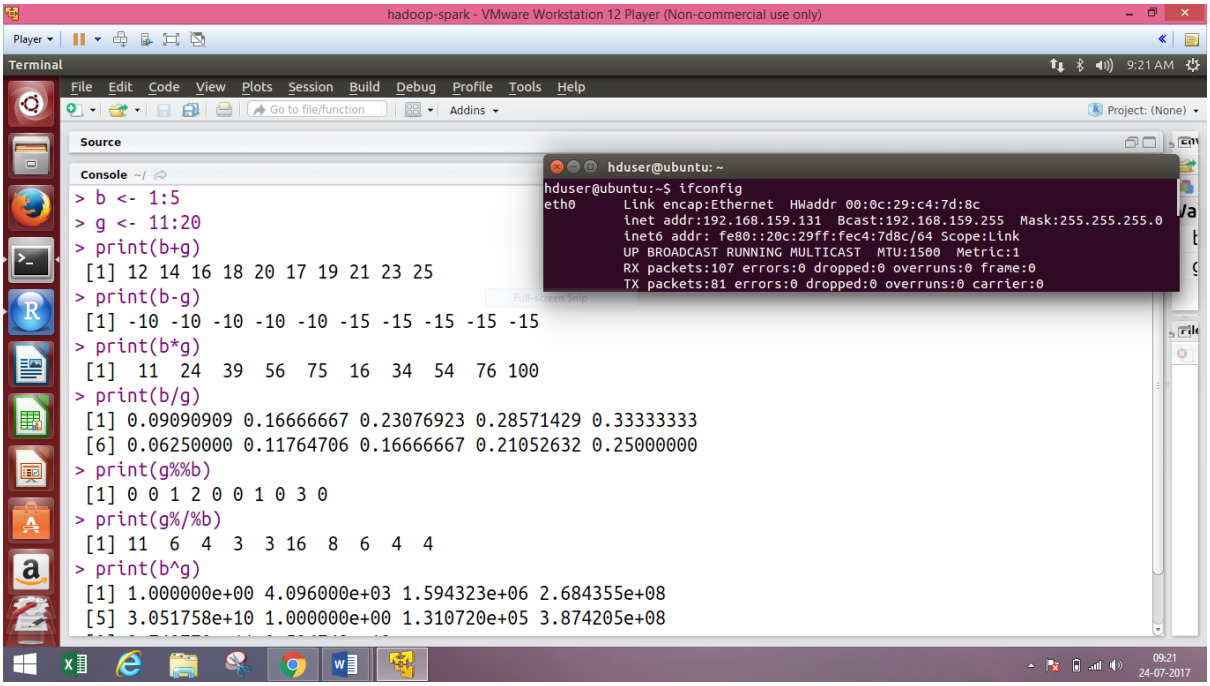


Figure 21: Arithmetic Operators

- Relational Operators

<	Checks if each element of the first vector is greater than the corresponding element of the second vector.
>	Checks if each element of the first vector is less than the corresponding element of the second vector.
==	Checks if each element of the first vector is equal to the corresponding element of the second vector.
<=	Checks if each element of the first vector is less than or equal to the corresponding element of the second vector.
>=	Checks if each element of the first vector is greater than or equal to the corresponding element of the second vector.
!=	Checks if each element of the first vector is unequal to the corresponding element of the second vector.

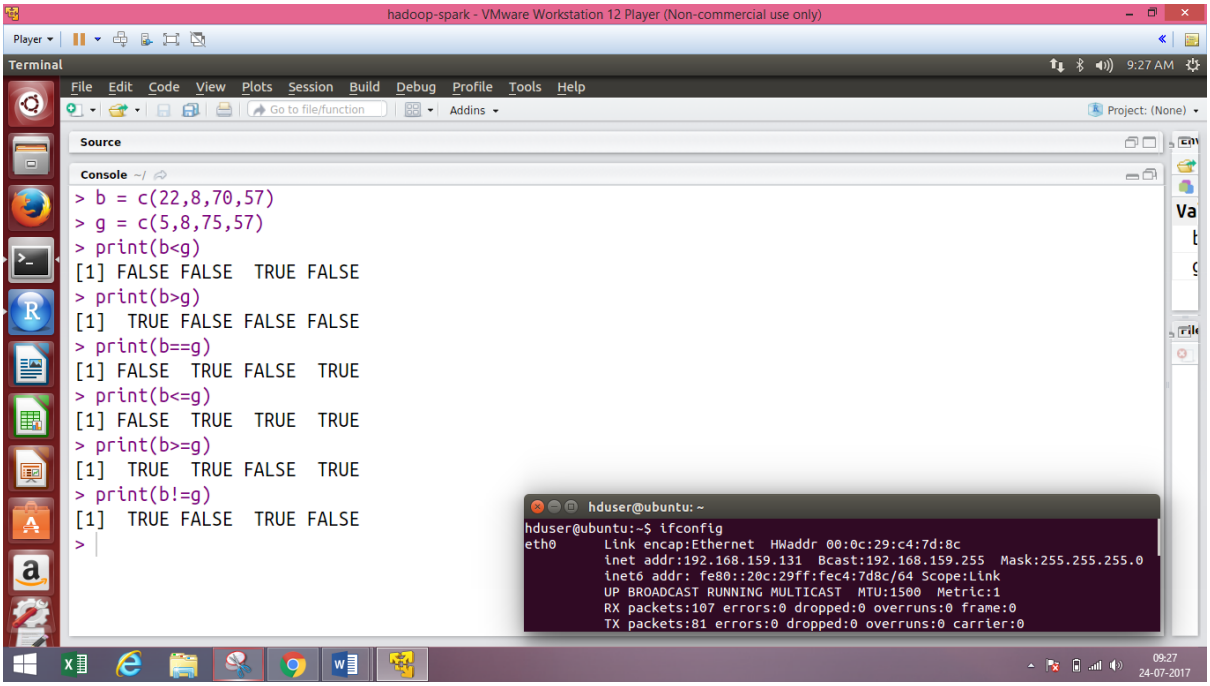


Figure 22: Relational Operators

- Logical Operators

&	It is called Element-wise Logical AND operator. It combines each element of the first vector with the corresponding element of the second vector and gives the output TRUE if both the elements are TRUE.
|	It is called Element-wise Logical OR operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if one the elements is TRUE.
!	It is called Logical NOT operator. Takes each element of the vector and gives the opposite logical value.
&&	It is called as Logical AND operator. Takes first element of both the vectors and gives the TRUE only if both are TRUE. It considers only the first element of the vectors and give a vector of single element as output.
||	Called Logical OR operator. Takes first element of both the vectors and gives the TRUE if one of them is TRUE. It considers only the first element of the vectors and give a vector of single element as output.

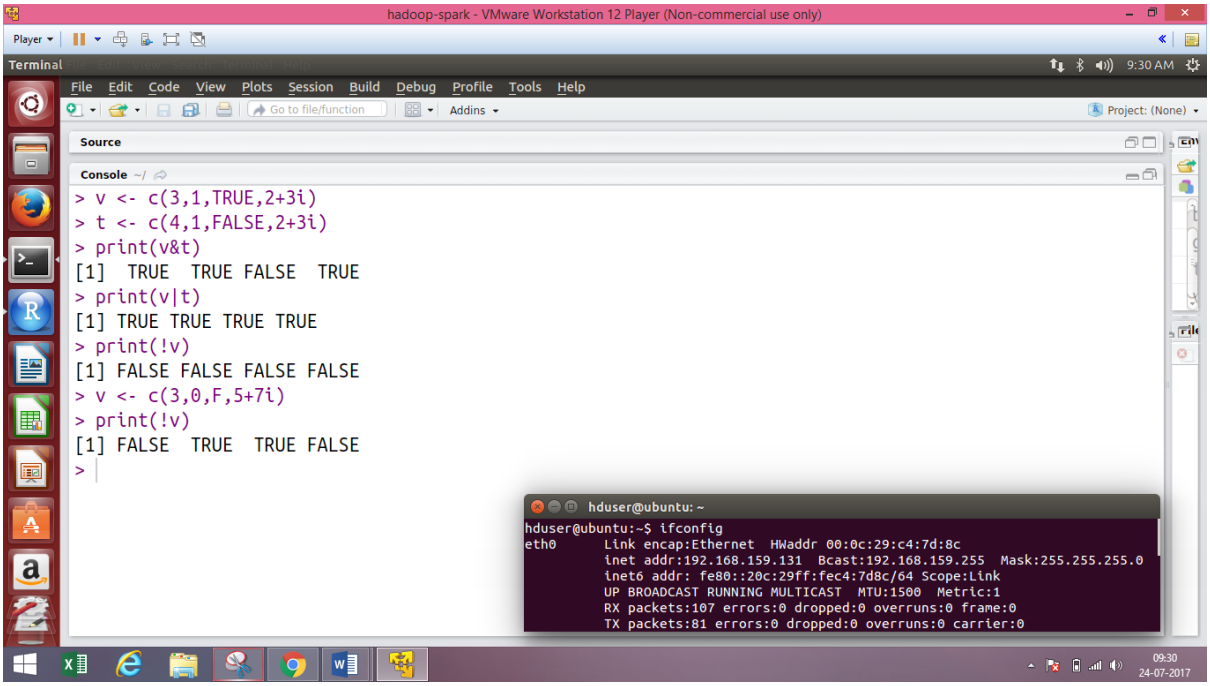


Figure 23: Logical Operators

- Assignment Operators

<- **==** **<--**	Left Assignment Operators
-> **->**	Right Assignment Operators

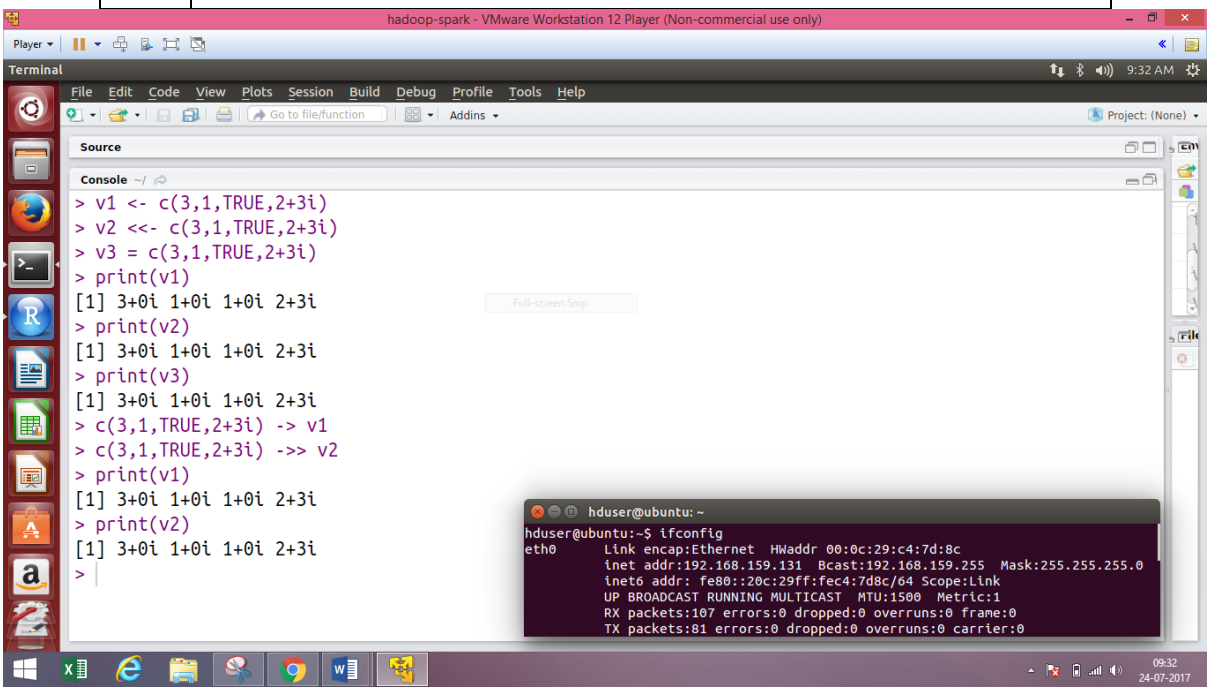


Figure 24: Assignment Operators

- Miscellaneous Operators

:	It is Colon operator. It creates the series of numbers in sequence for a vector.

%in%	This operator is used to identify if an element belongs to a vector.
%*%	This operator is used to multiply a matrix with its transpose.

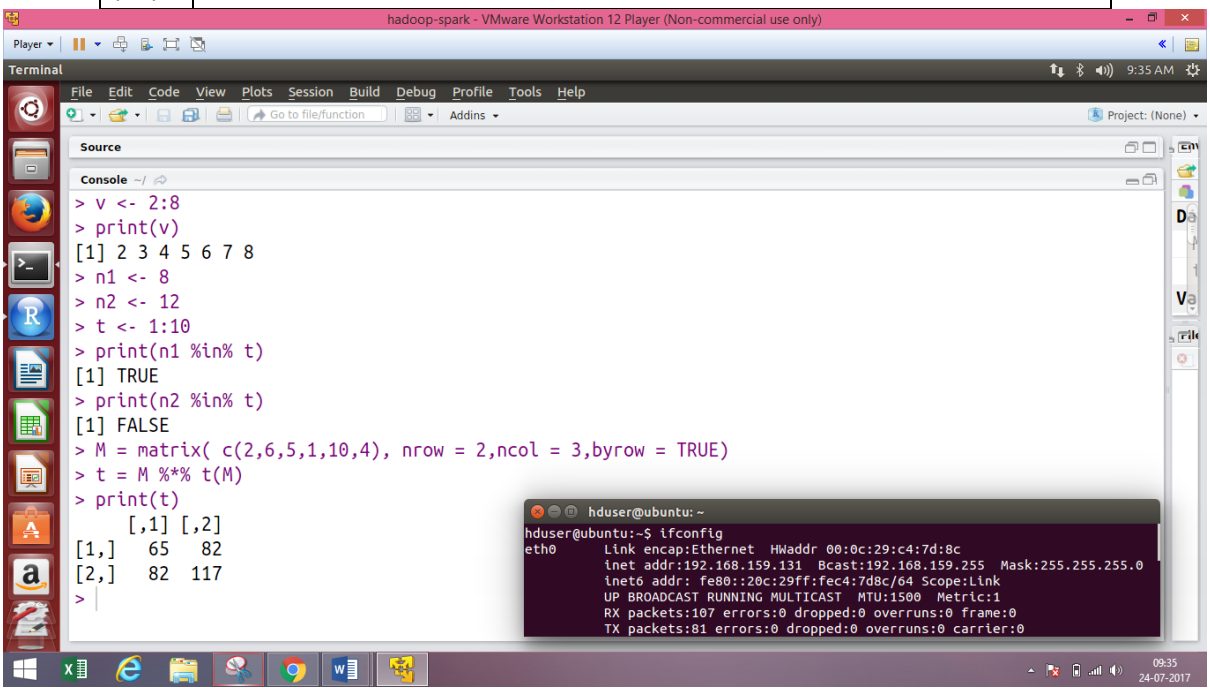


Figure 25: Miscellaneous Operators

4. Functions

- **class** and **mode** of variable/object:
  R possesses a simple generic function mechanism which can be used for an object-oriented style of programming. Class can be seen as abstract data type here. Mode is the type or storage mode of an object.
  Use:

  class(x)
  class(x) <- value
  mode(x)
  mode(x) <- value

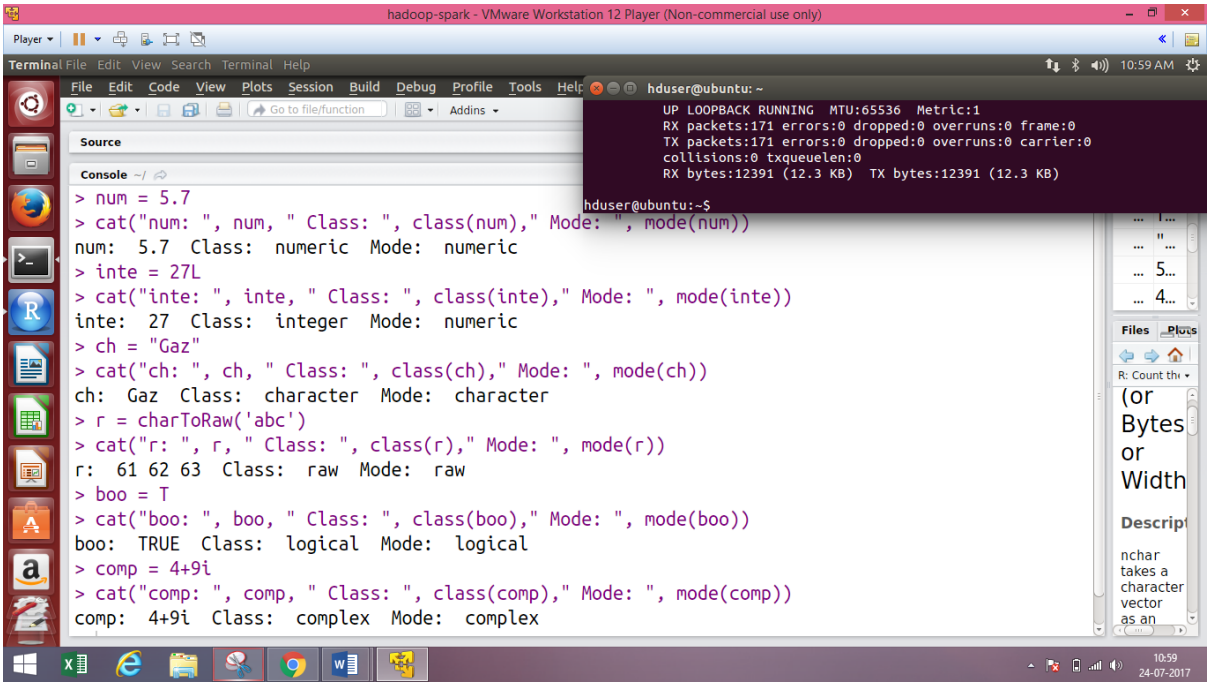


Figure 26: class() and mode() functions

- Type Casting:
  i. **as.integer()** function: This function is used to convert numeric, complex, logical, raw data in integer.
  P.S. Converting Character to Integer generate NA which is introduced by coercion.
  Use: as.integer(x, ...)

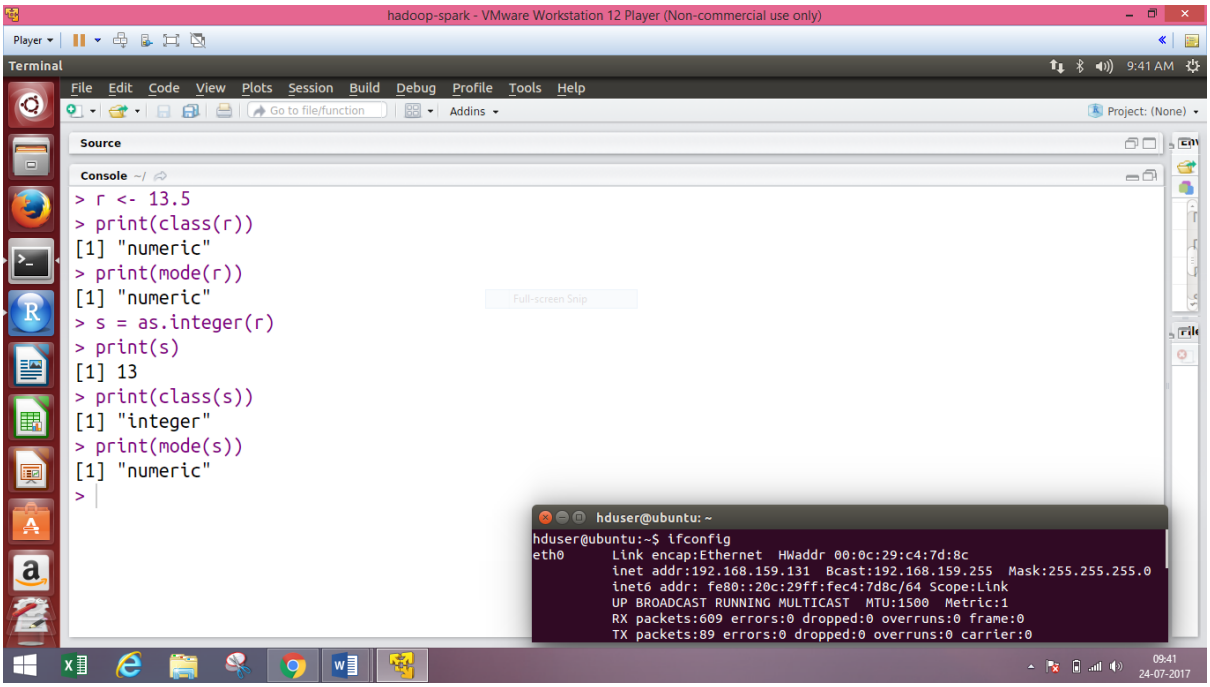


Figure 27: as.integer() function

ii. **as.double()** and **as.numeric()** functions: Create, coerce to or test for a double-precision vector.
Use:

as.double(x, ...)
as.single(x, ...)
as.numeric(x, ...)

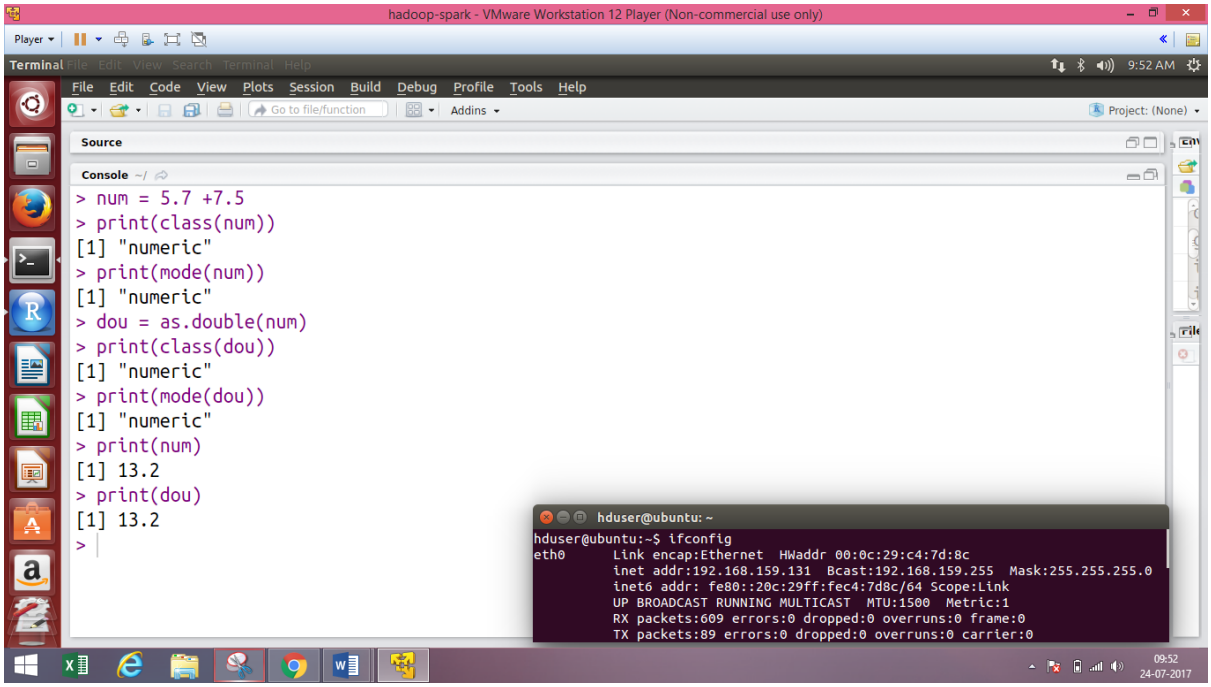


Figure 28: as.double() function

iii. **as.character()** function: Create or test for objects of type "character".
Use: as.character(x, ...)

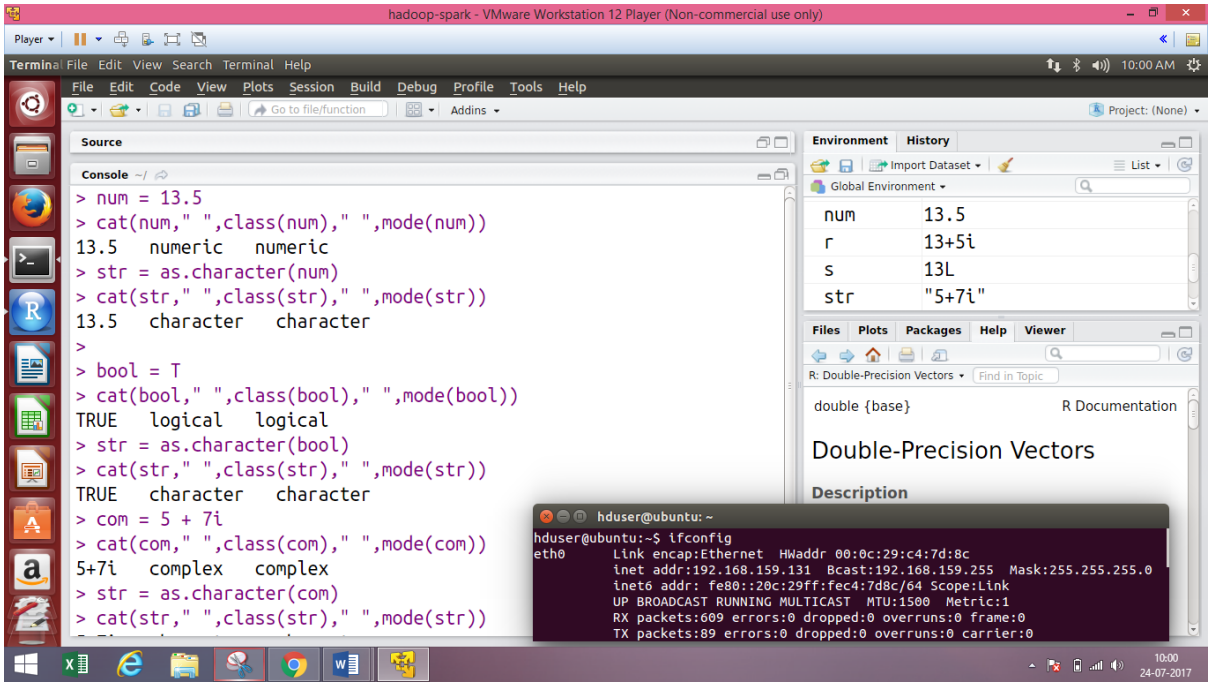


Figure 29: as.character() function

- **length()** function:
  It gives the number of the objects in the vector. Also see LIST-LENGTH
  Use:

length(x)
length(x) <- value

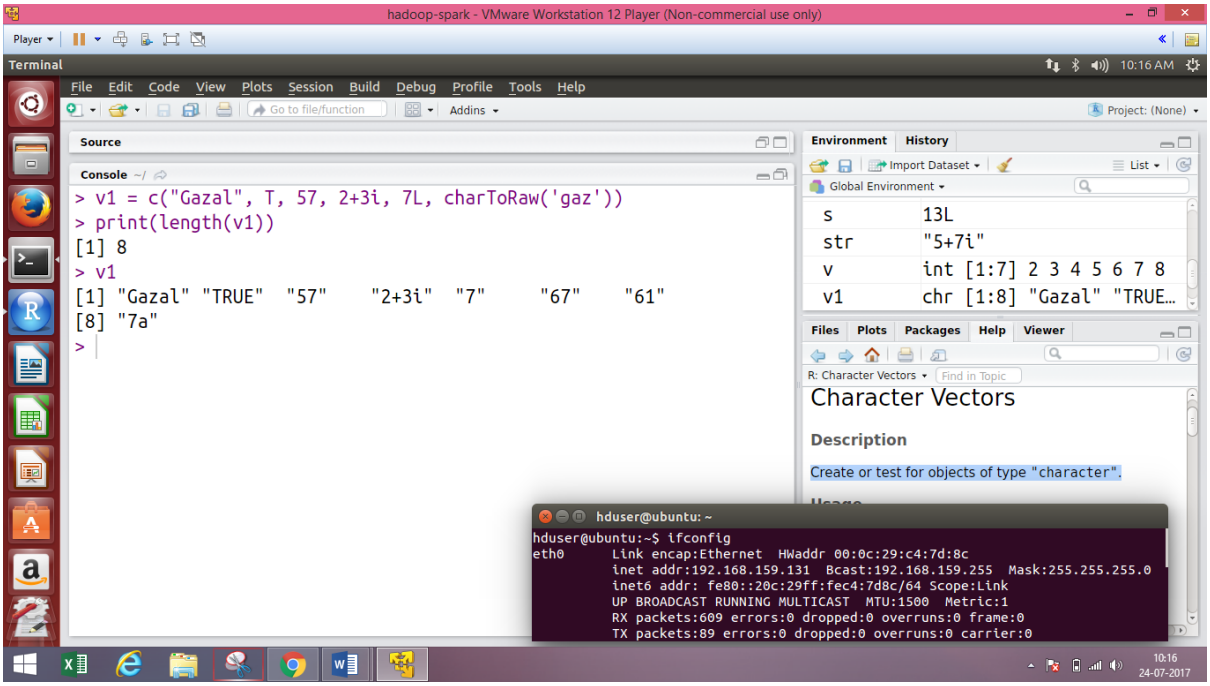


Figure 30: length() function

- **nchar()** function:
  Gives vector f sizes of the corresponding elements of provided vector.
  Use:

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

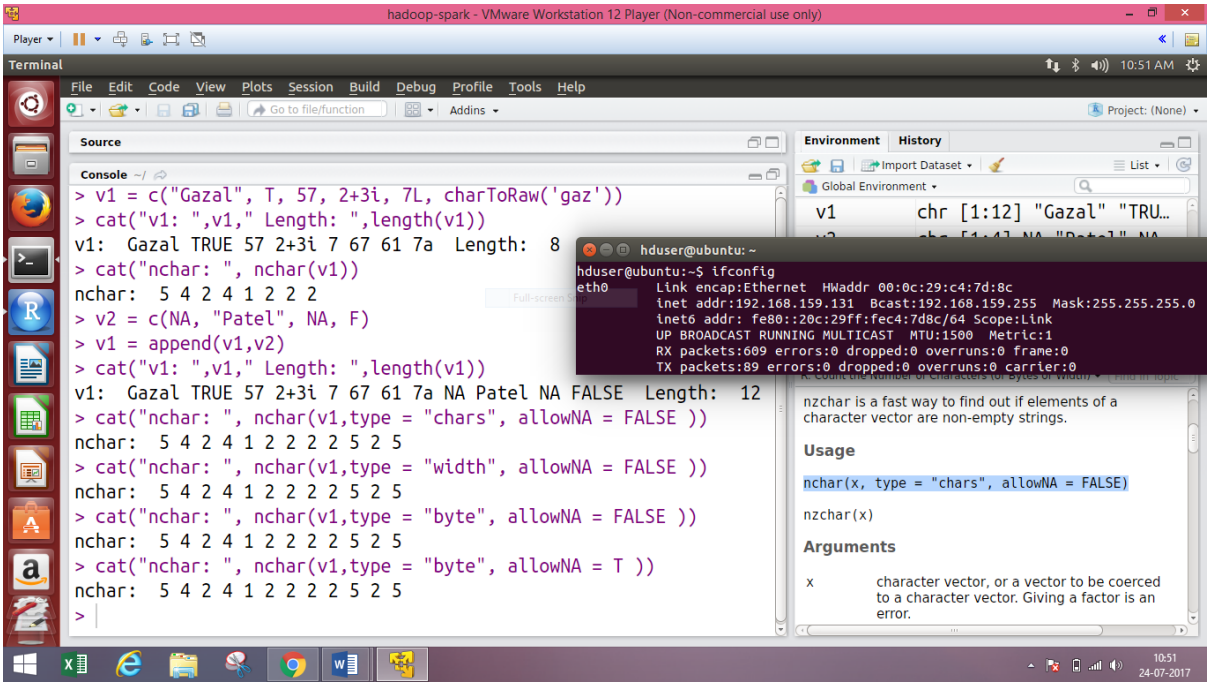


Figure 31: nchar() function

- **c()** function: creates vector of any object.

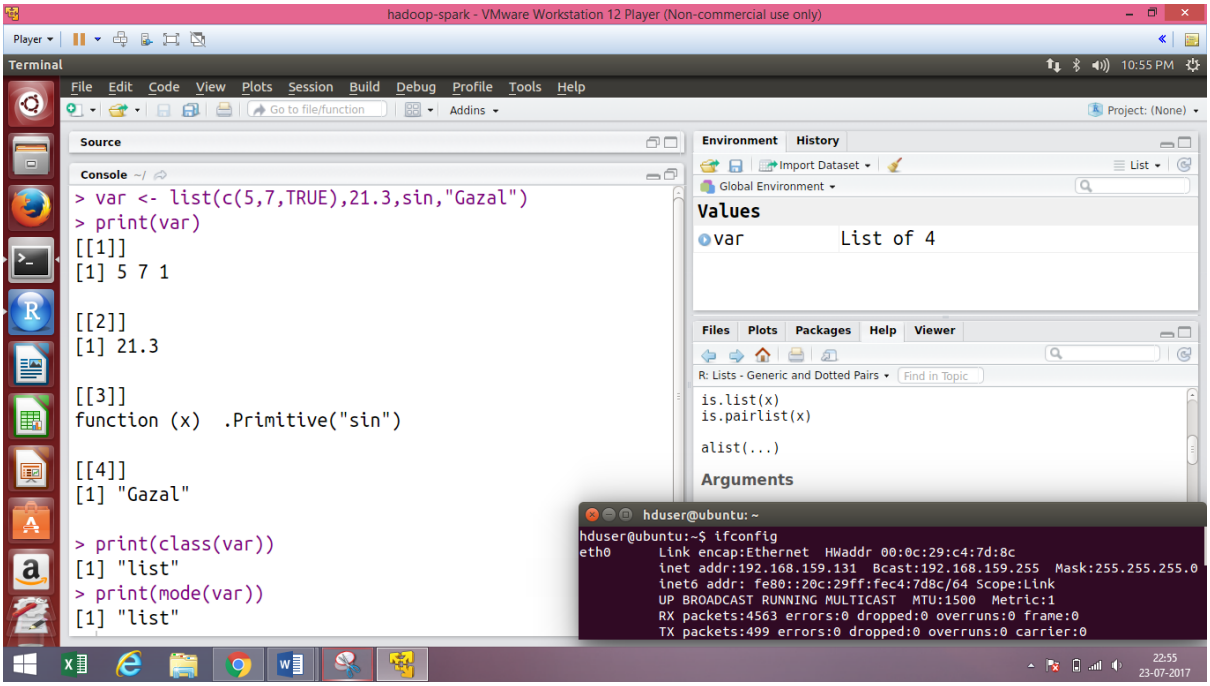


Figure 32: c() function

- **append()** function:
  It appends any element (Singular or Atomic variable or function) at the end of the given vector. Value for 'after' can be given optionally which is a subscript, after which the values are to be appended.
  Use: append(x, values, after = length(x))

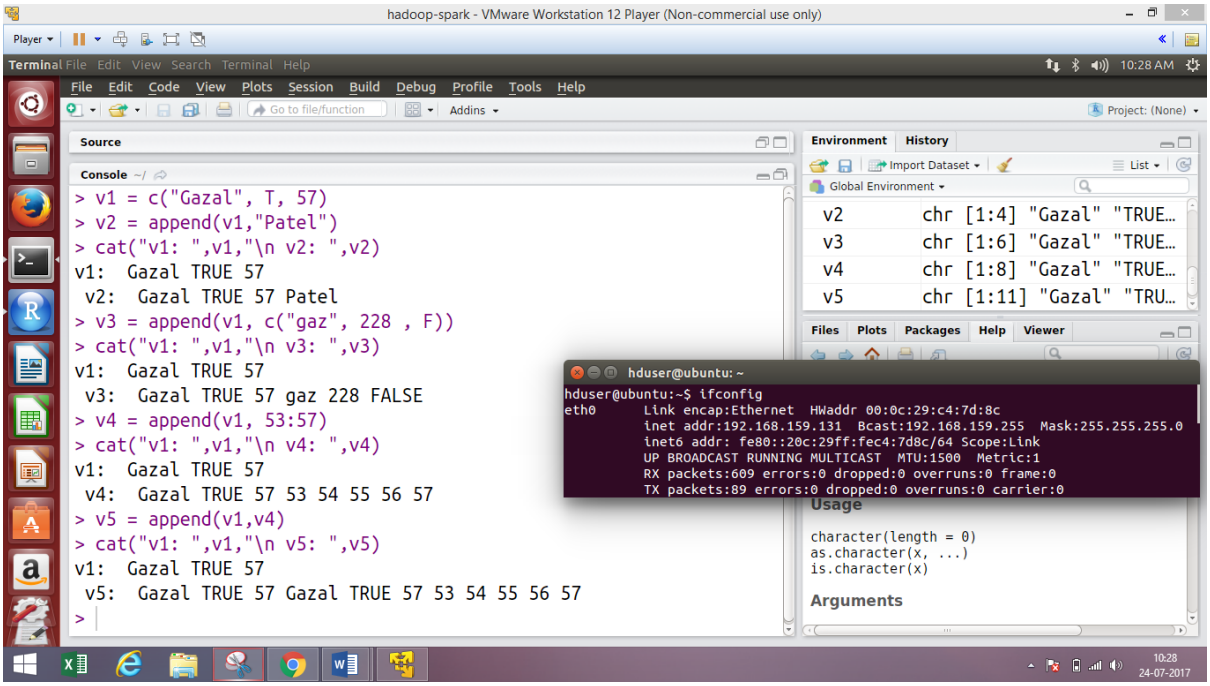


Figure 33: append() function

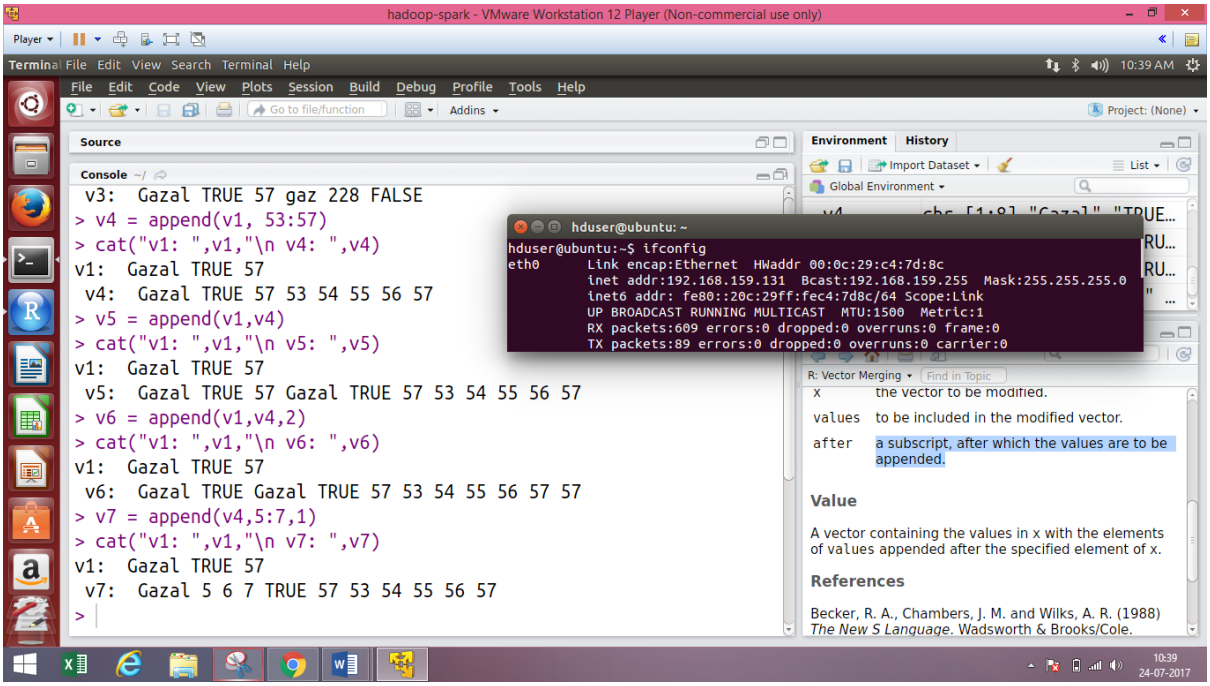


Figure 34: append() function with 'after' parameter

- **seq**
  Generate regular sequences.
  i. **seq()** or **seq.default()** functions: seq is a standard generic with a default method. All its input are finite numeric. (That is, not infinite, NaN or NA). It takes from, to and stepping factor as default parameters.
  Use:

```
seq(...)
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)), length.out =
NULL, along.with = NULL, ...)
seq(from, to)
seq(from, to, by= )
seq(from, to, length.out= )
seq(along.with= )
seq(from)
seq(length.out=)
```

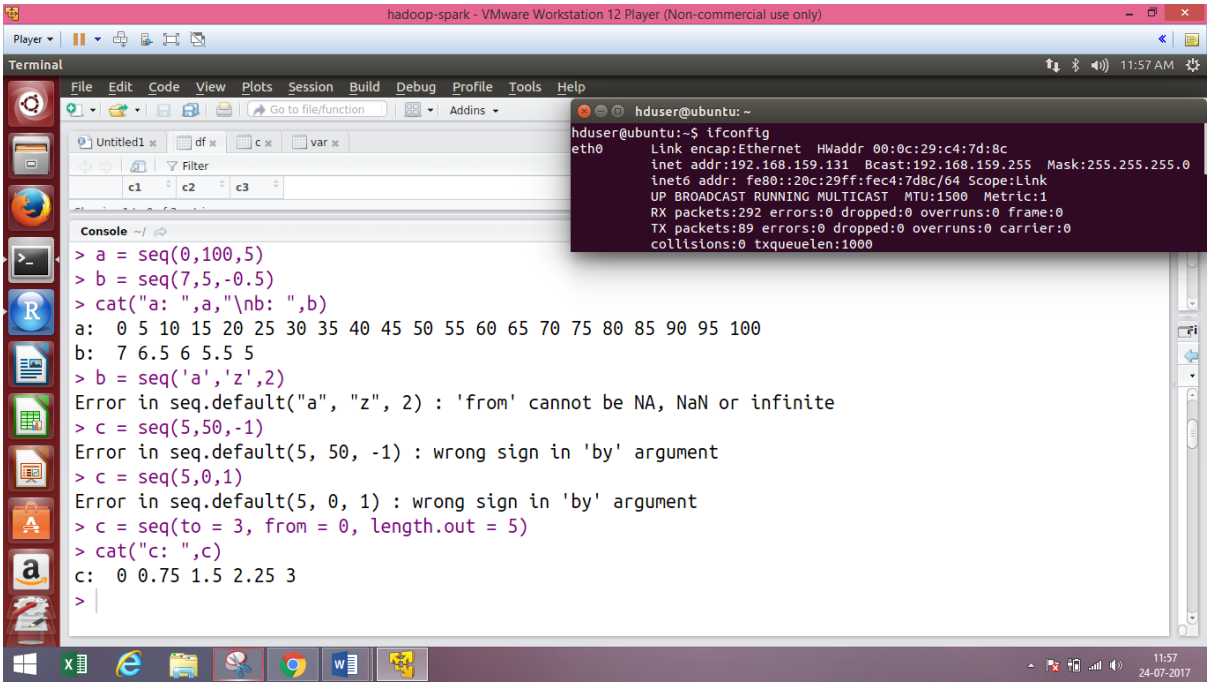


Figure 35: seq() function

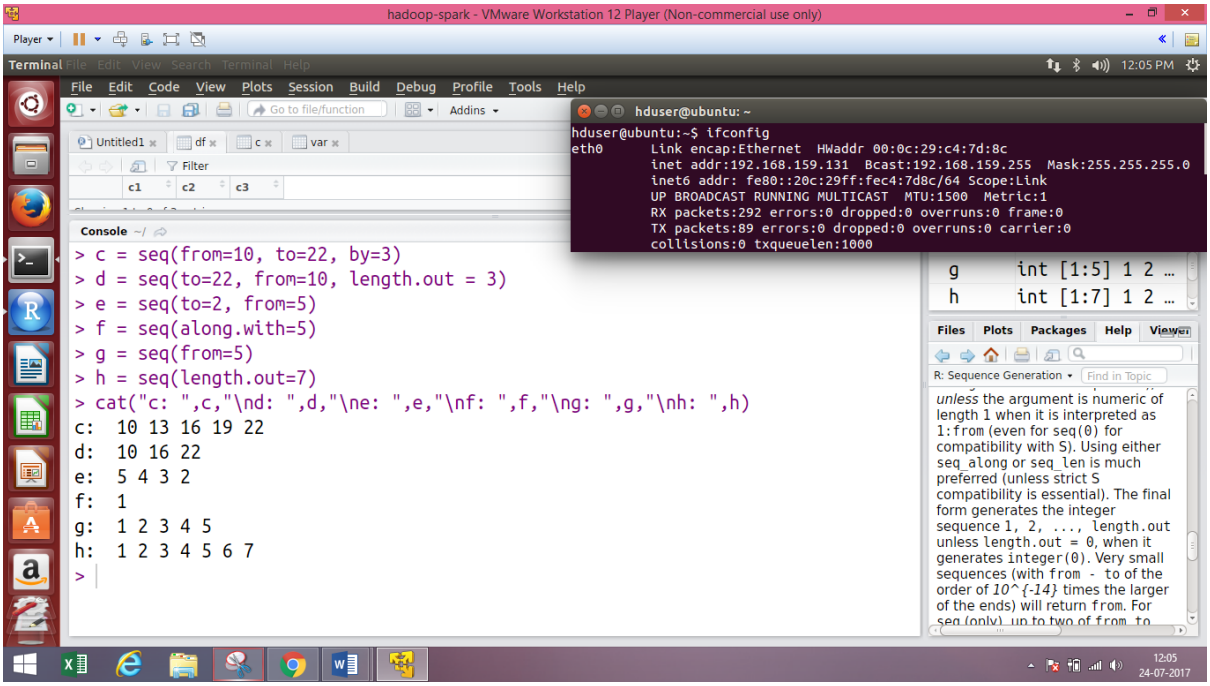


Figure 36: seq() parameter variance

ii. **seq.int()** function: It generates regular sequences. It is standard generic with default method which is faster but has few restrictions.
Use: seq.int(from, to, by, length.out, along.with, ...)

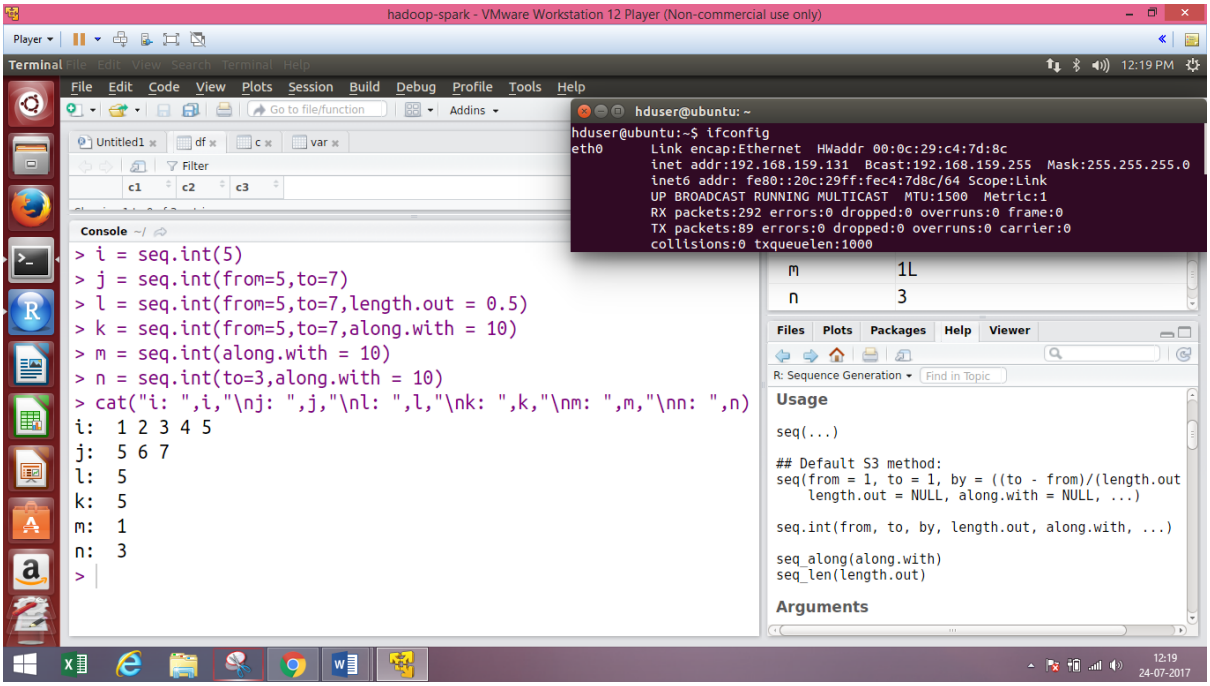


Figure 37: seq.int() function

iii. **seq.along()** function: seq is a standard generic with a default method. All its input are finite numeric. (That is, not infinite, NaN or NA). It takes from, to and stepping factor as default parameters.
Use: seq_along(along.with)

iv. **seq_len()** function: seq is a standard generic with a default method. All its input are finite numeric. (That is, not infinite, NaN or NA). It takes from, to and stepping factor as default parameters.
Use: seq_len(length.out)

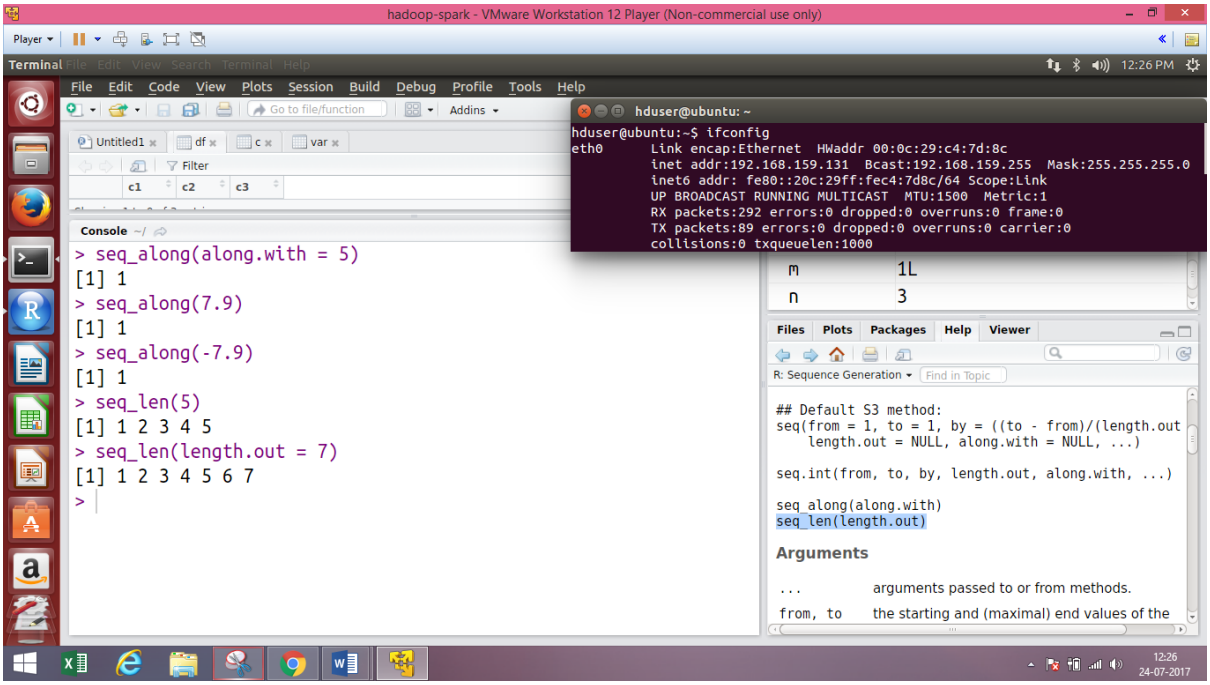


Figure 38: seq_along() and seq_len() functions

- **min(), max(), median(), which.min()** and **which.max()** functions:
  These min(), max() and median() functions determines the minimun and maximum and median of the vector resepectively. Which.min() and which.max() functions determines the location of minimum and maximum value in vector respectively, i.e., index of the (first) minimum or maximum of a numeric (or logical) vector.
  For a logical vector x, which.min(x) and which.max(x) return the index of the first FALSE or TRUE, respectively.
  Use:

  max(..., na.rm = FALSE)
  min(..., na.rm = FALSE)
  which.min(x)
  which.max(x)

  P.S. x in above function is either integer or double.

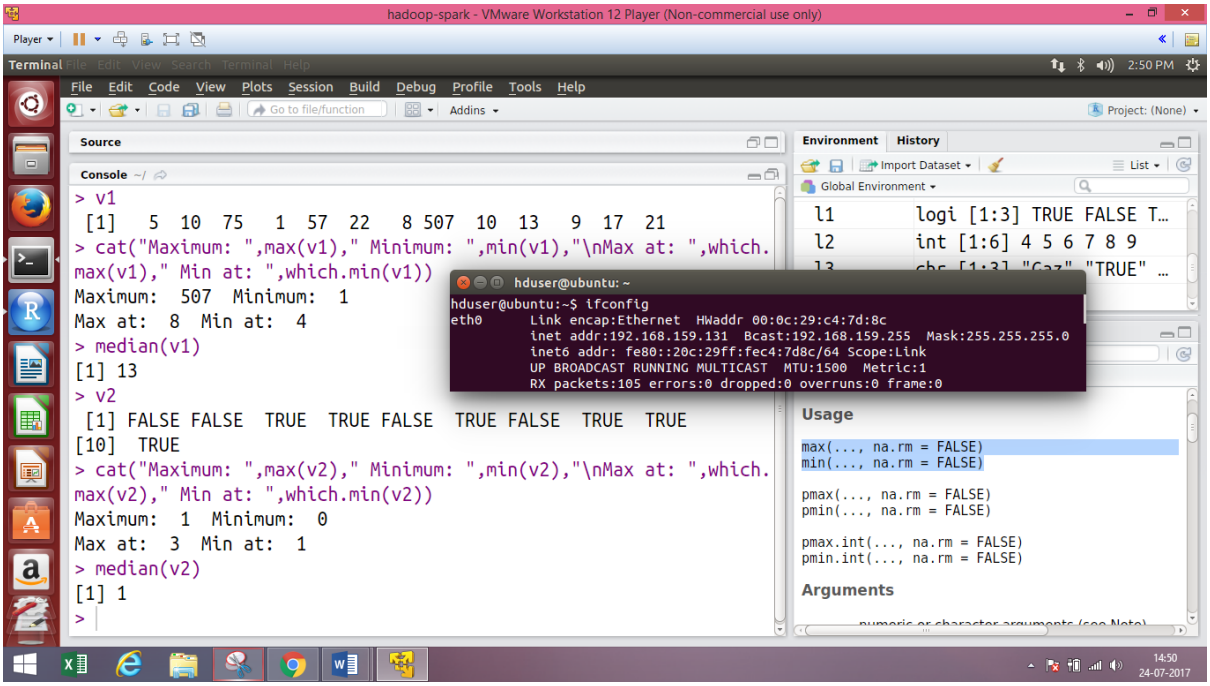


Figure 39: min() and max() functions

- **rep()** function: rep replicates the values in x.
  i. **rep()** function: It is a generic repetition function.
  Use: rep(x, ...)

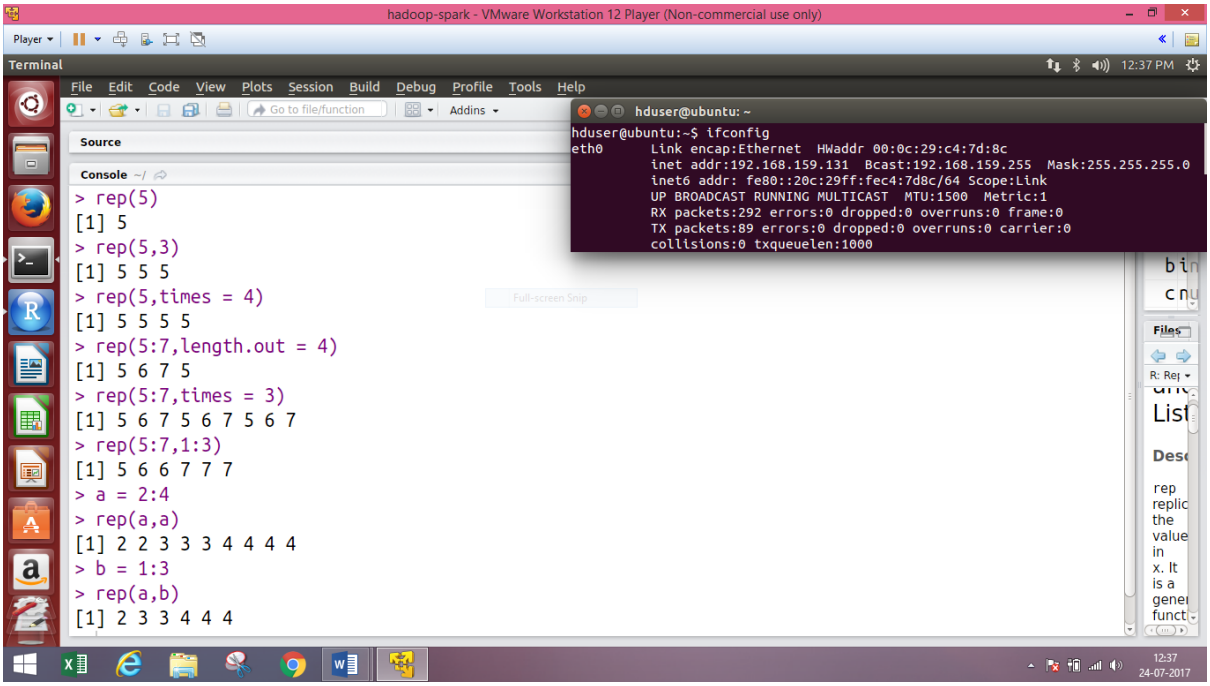


Figure 40: rep() function

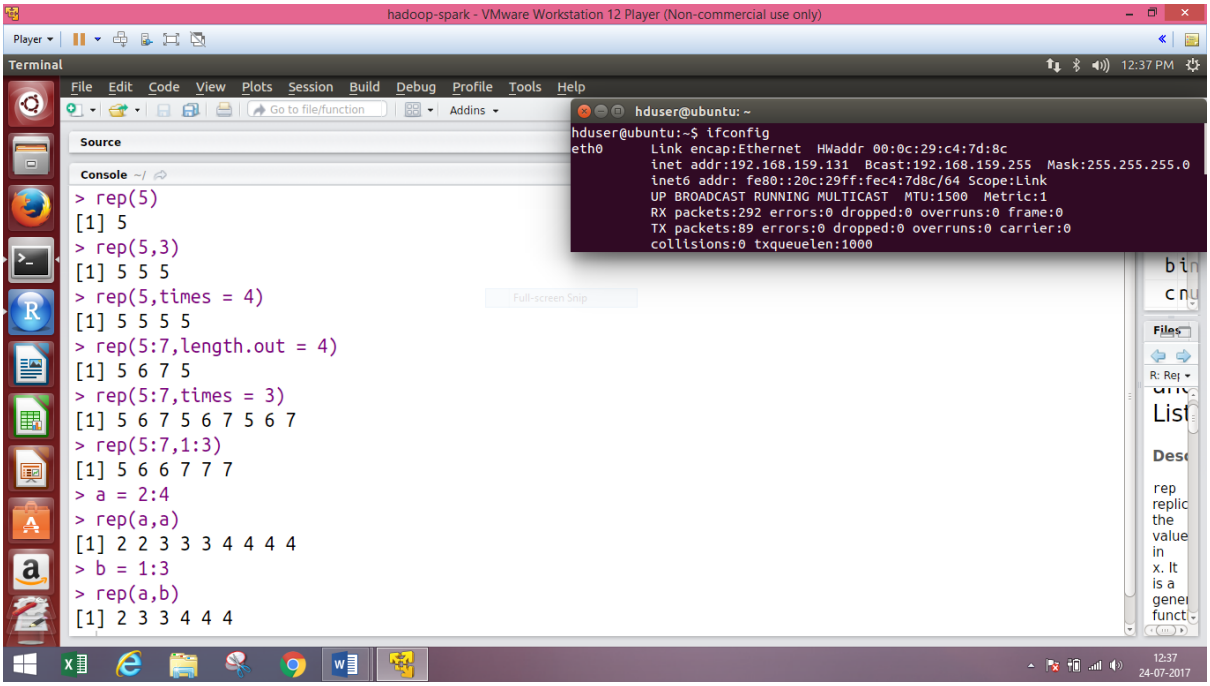


Figure 41: rep() function

ii. **rep.int()** and **rep_len()** functions:
rep.int and rep_len are faster simplified versions for two common cases. They are not generic.
Use:

rep.int(x, times)
rep_len(x, length.out)

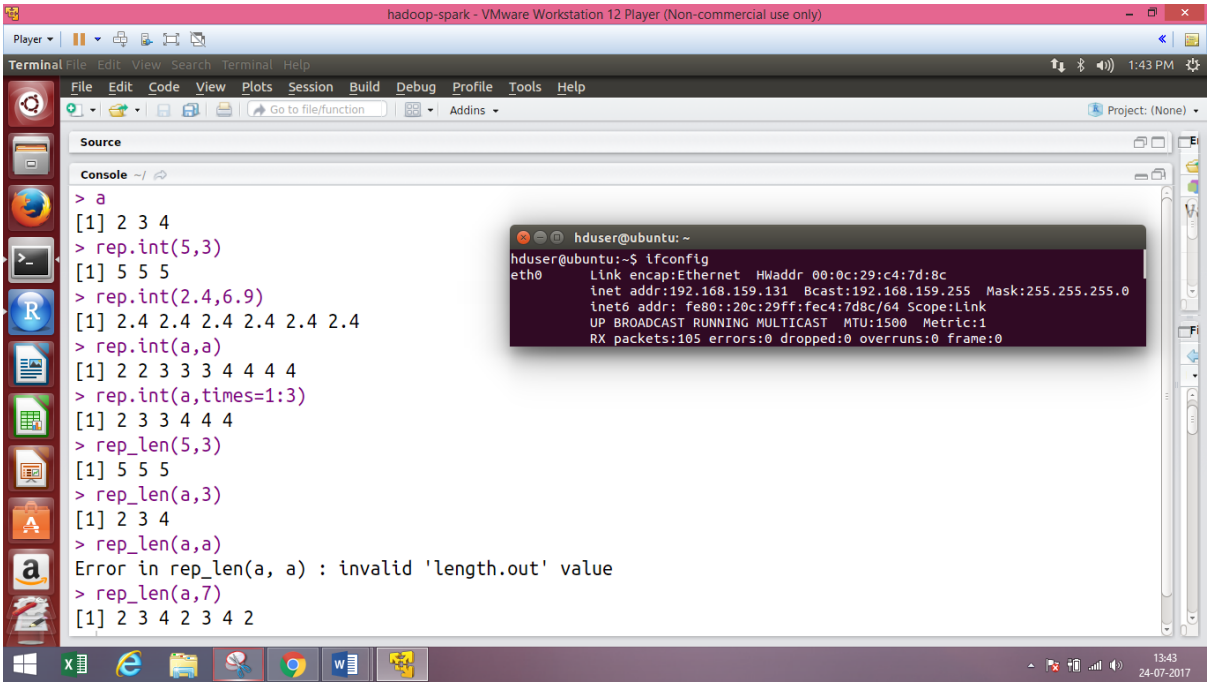


Figure 42: rep_int() and rep_len() functions

- **runif()** function: It is one of the functions in uniform distribution class from interval min to max. It generates random deviates. It generates random number from 0 to 1. For random variable in other range optional min and max parameters can be used. The characteristics of output from pseudo-random number generators (such as precision and periodicity) vary widely. See .Random.seed
  Use: runif(n, min = 0, max = 1)

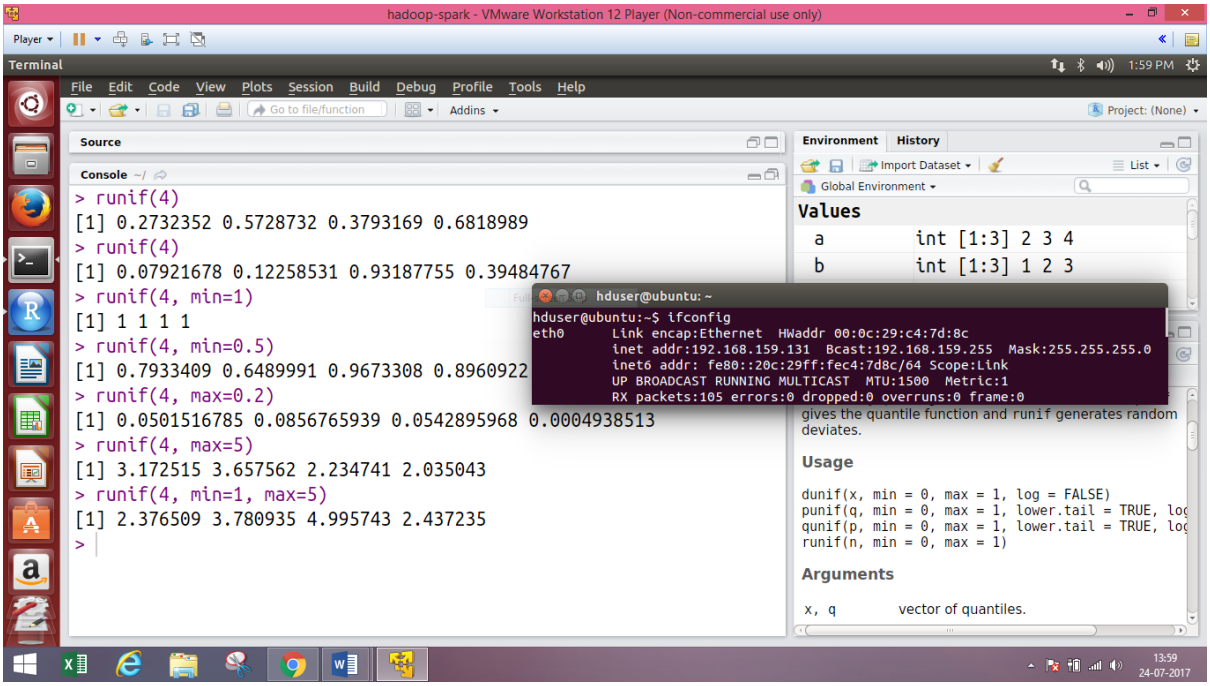


Figure 43: runif() function

- **names()** function: It assigns names to the elements of the vector. Here any object can be assigned a name which is the value defined in the vector.

Use:

names(x)

names(x) <- value

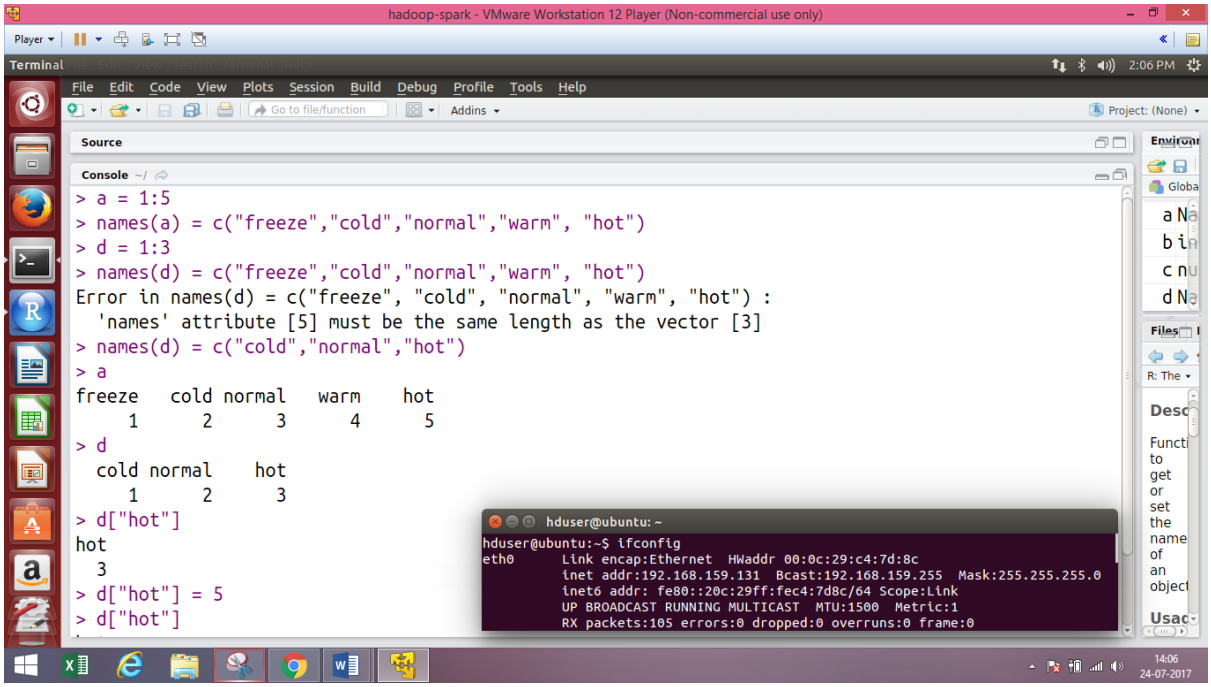


Figure 44: names() function

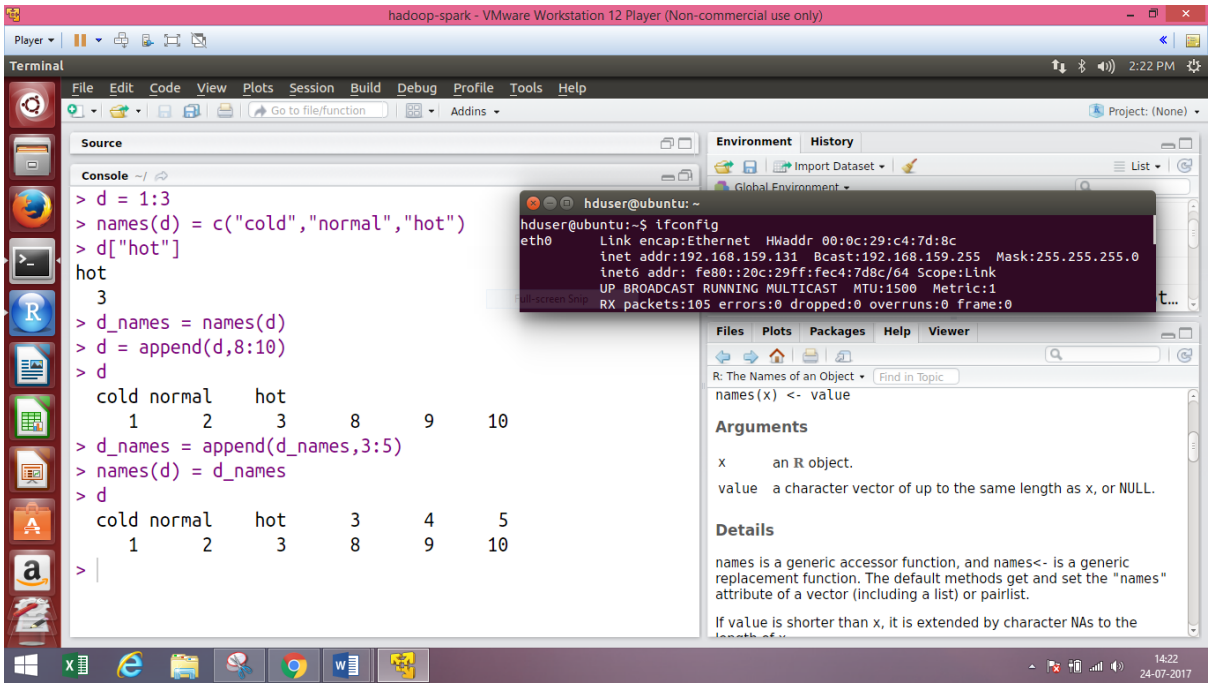


Figure 45: names - use

- **list()** and **pairlist()** functions:
  The arguments to list or pairlist are of the form value or tag = value. The functions return a list or dotted pair list composed of its arguments with each value either tagged or untagged, depending on how the argument was specified.
  Use:

  list(...)

pairlist(...)

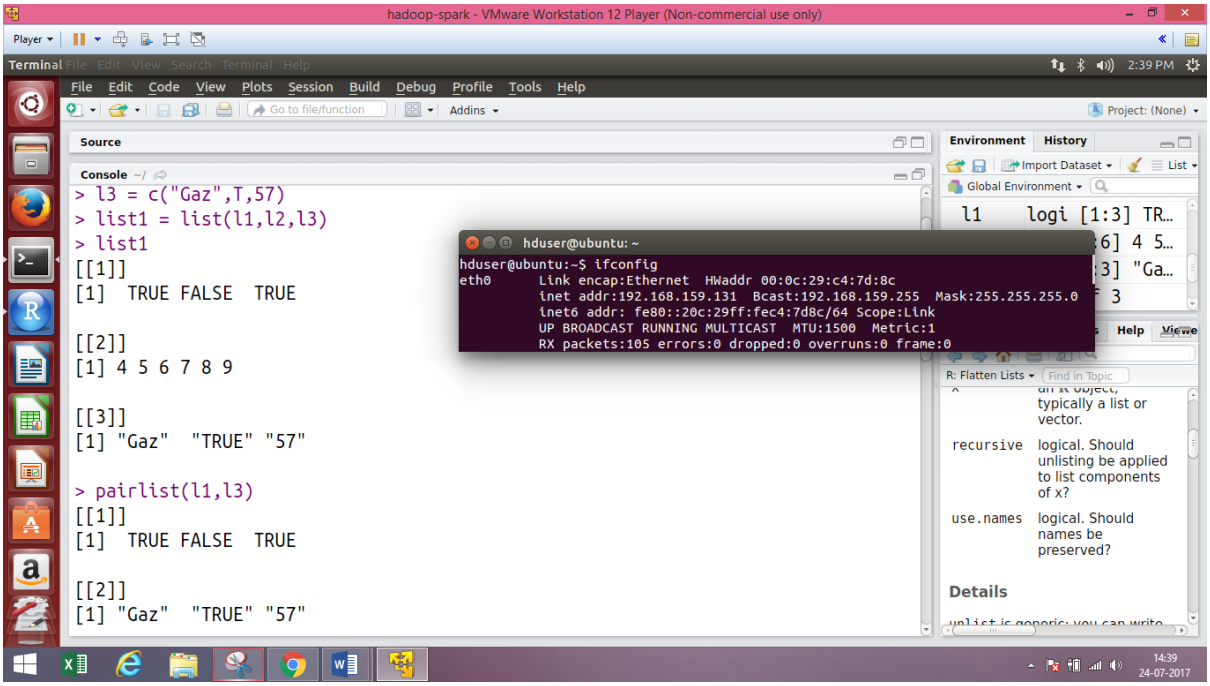


Figure 46: list() and pairlist() functions

- **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)

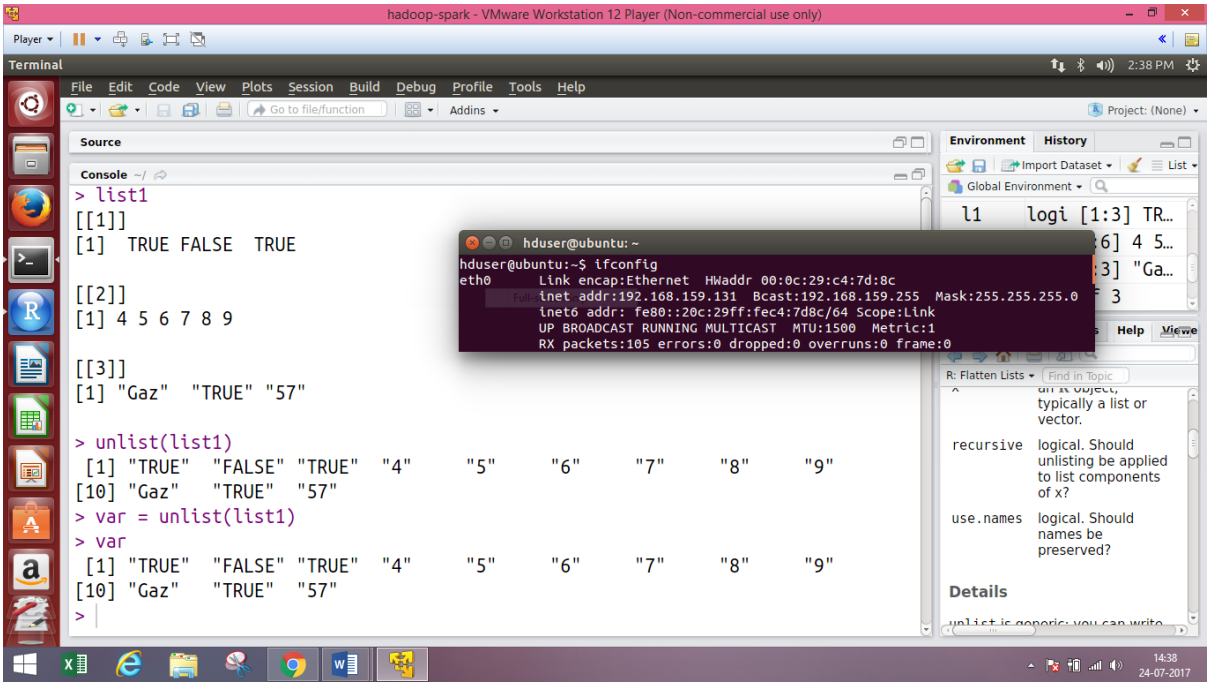


Figure 47: unlist() function

- **ls()** function: See LS
- **sort()** function: Sorts the elements in vector.
  Use:

  sort(x, decreasing = FALSE, ...)

sort(x, decreasing = FALSE, na.last = NA, ...)
sort.int(x, partial = NULL, na.last = NA, decreasing = FALSE,
method = c("shell", "quick"), index.return = FALSE)

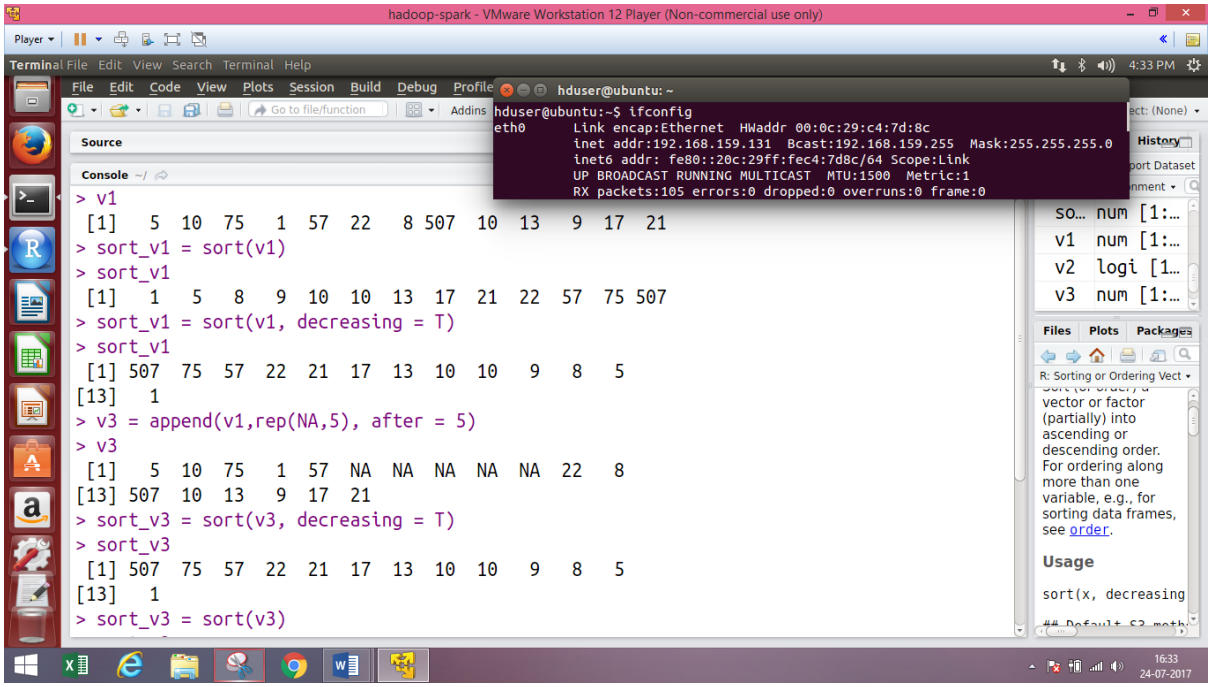


Figure 48: sort() function and decreasing parameter

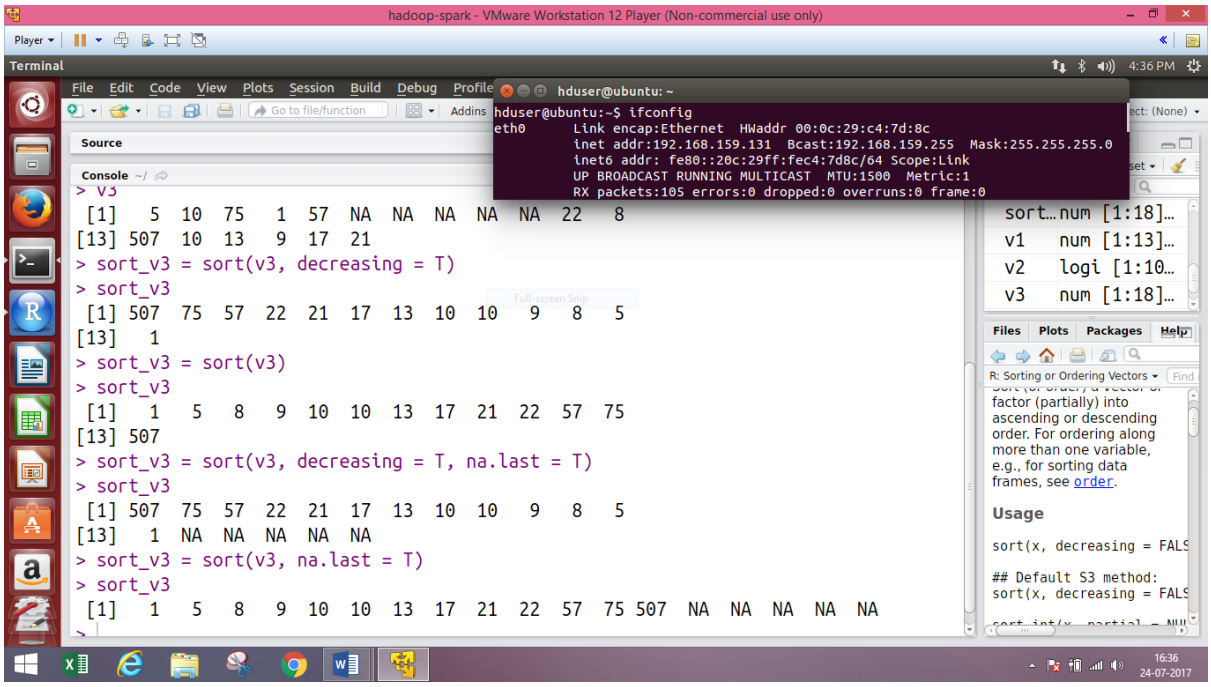


Figure 49: sort() function with decreasing and NA parameters

- **rev()** function: It provides a reversed version of its argument.

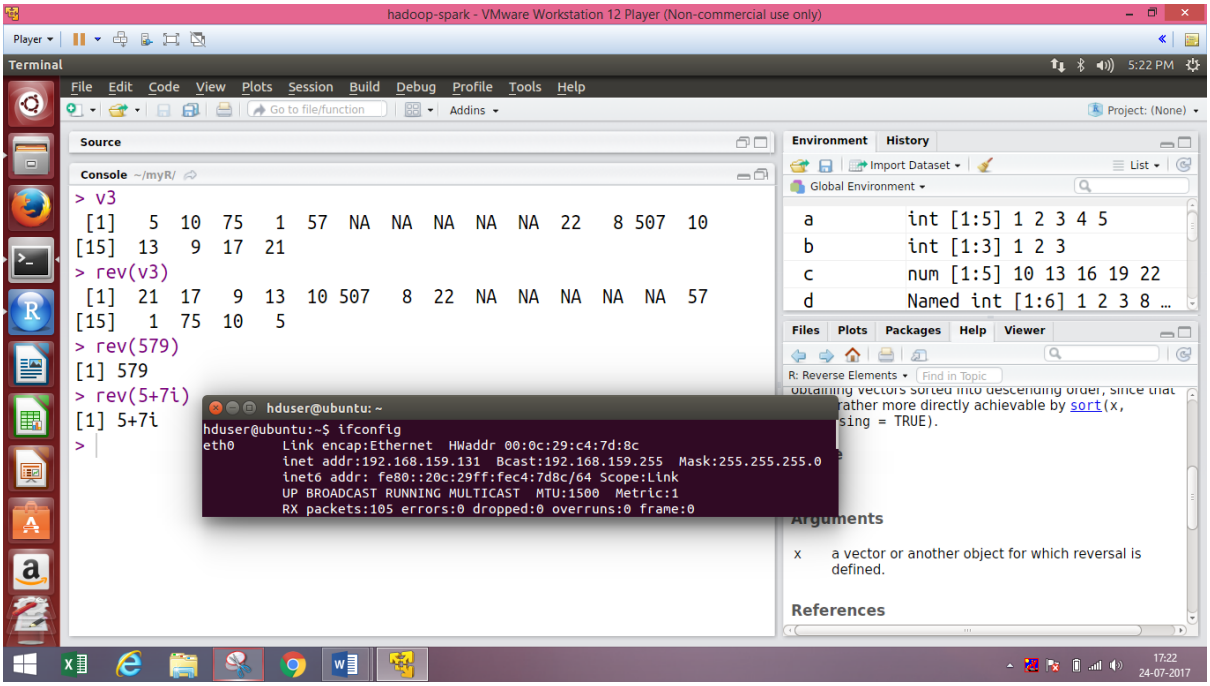


Figure 50: rev() function

- **order()** function:
  It 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. It returns the position of the value in return.
  Use:

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

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

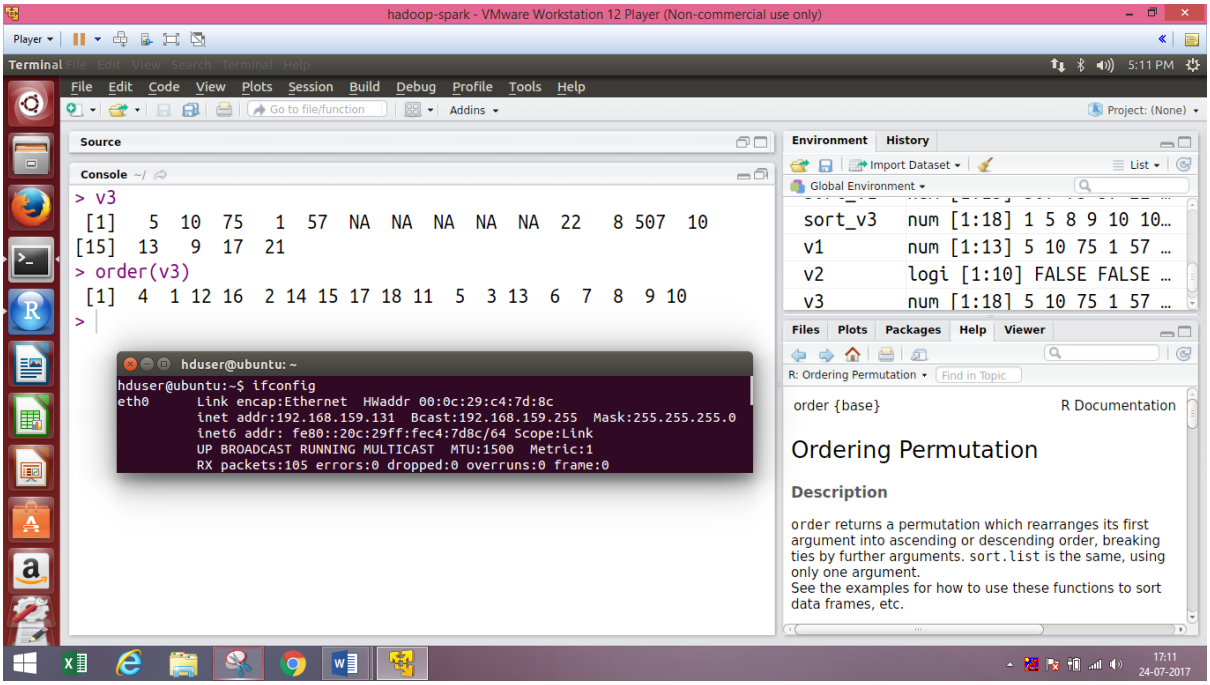


Figure 51: order() function

- **getwd()** function: getwd() returns an absolute filepath representing the current working directory of the R process. getwd returns a character string or NULL if the working directory is not available. On Windows the path returned will use / as the path separator and be encoded in UTF-8. The path will not have a trailing / unless it is the root directory (of a drive or share on Windows).
  Use: getwd()

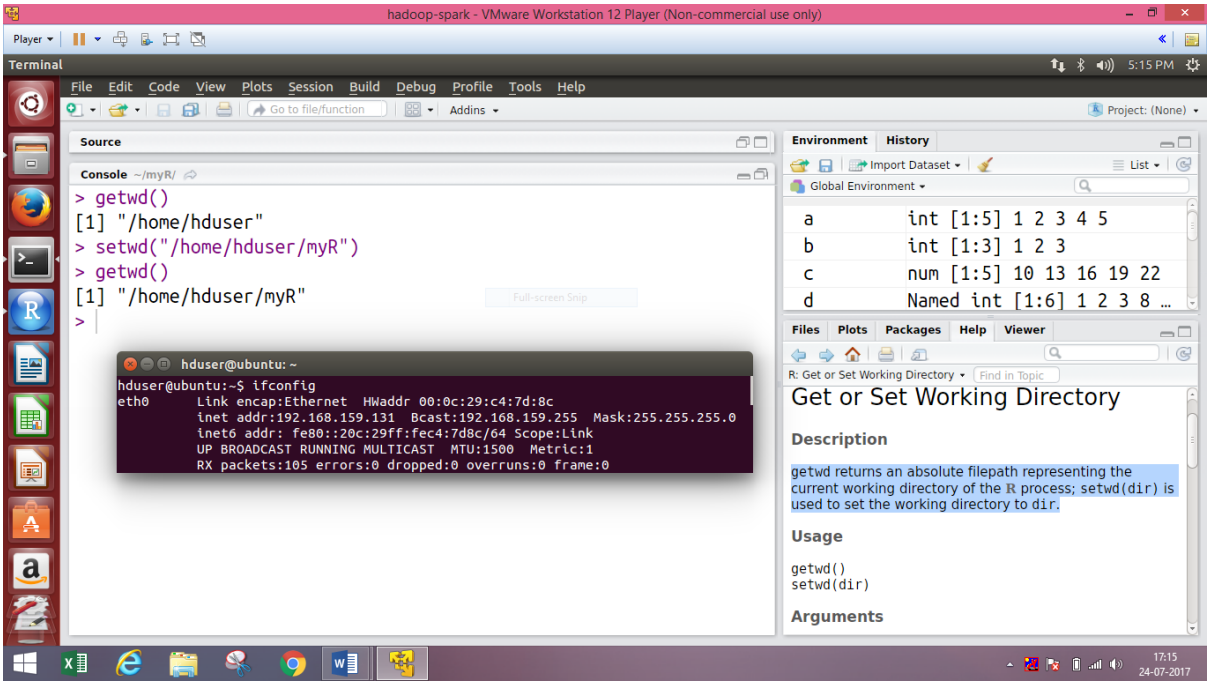


Figure 52: getwd() and setwd() functions

- **setwd()** function: setwd(dir) is used to set the working directory to dir. setwd returns the current directory before the change, invisibly and with the same conventions as getwd. It will give an error if it does not succeed (including if it is not implemented).
  Use: setwd(dir)
- **is.element()** function or **%in%** operator: Finds out if the particular element is in vector or not and returns logical vector.

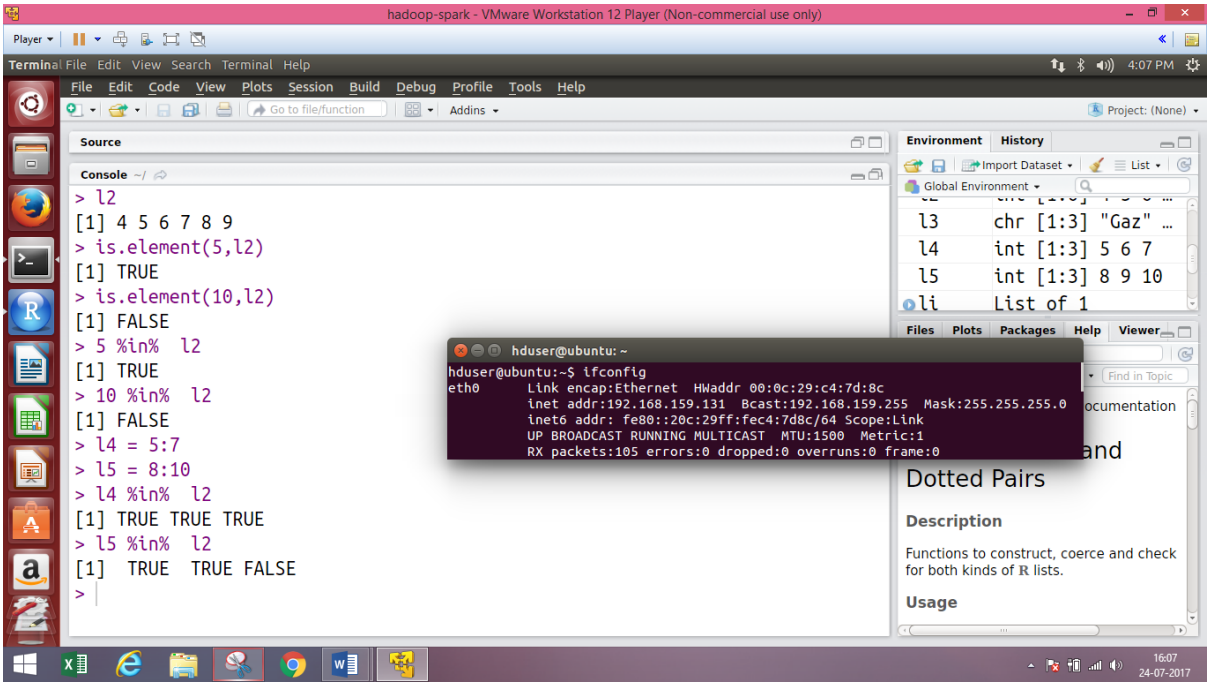


Figure 53: is.element() function and %in% operator

- **sample()** function: sample takes a sample of the specified size from the elements of x using either with or without replacement.

  Use:

  sample(x, size, replace = FALSE, prob = NULL)
  sample.int(n, size = n, replace = FALSE, prob = NULL)

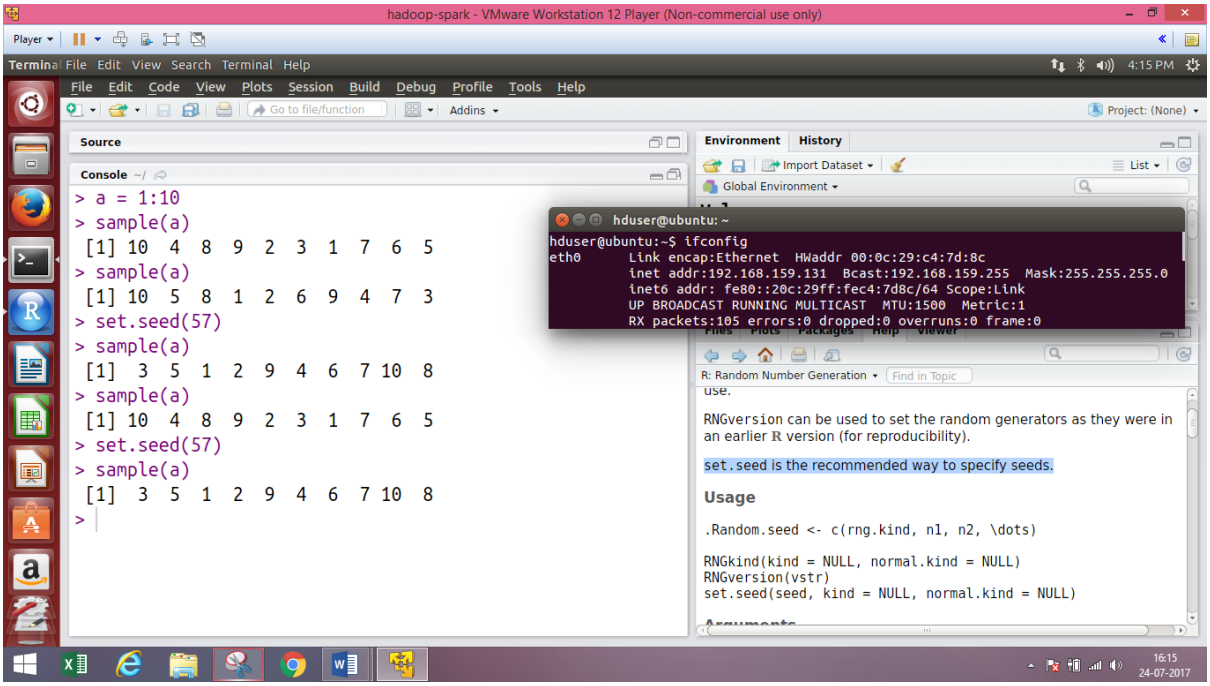


Figure 54: sample() and set.seed() function

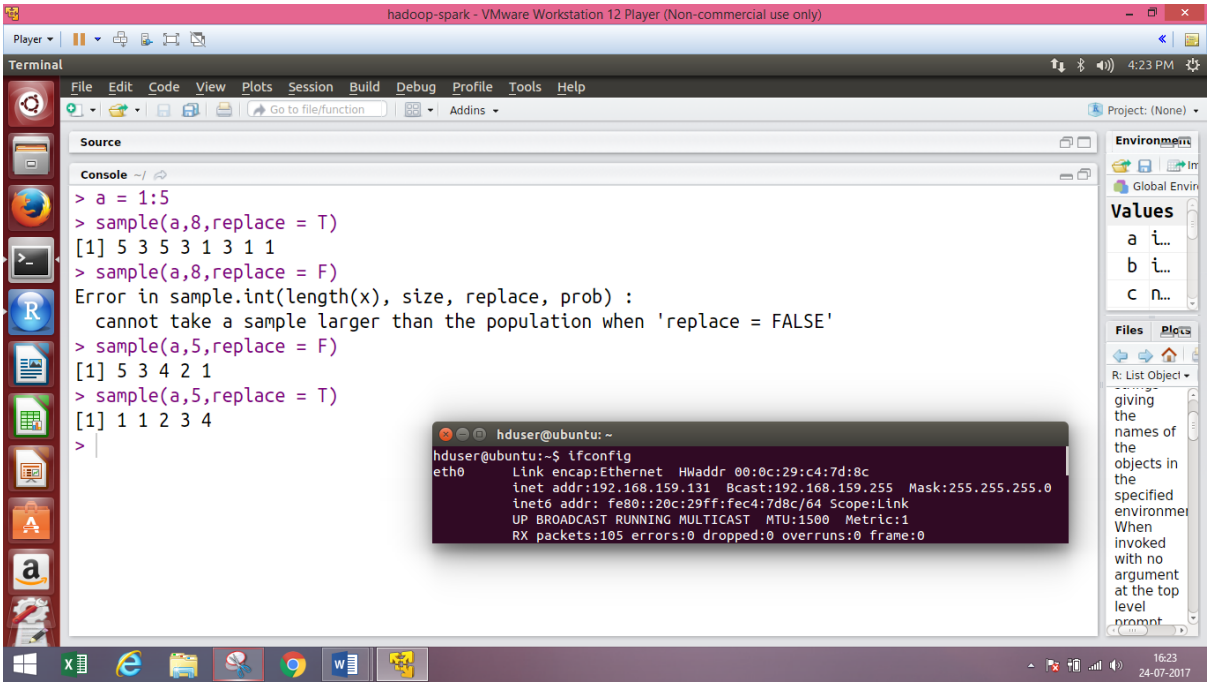


Figure 55: sample() with replace parameter

- **seed()** function: It is used to change the hidden seed variable which is used to create random variable. In this way same random number can be generated with same seed value arrangements.

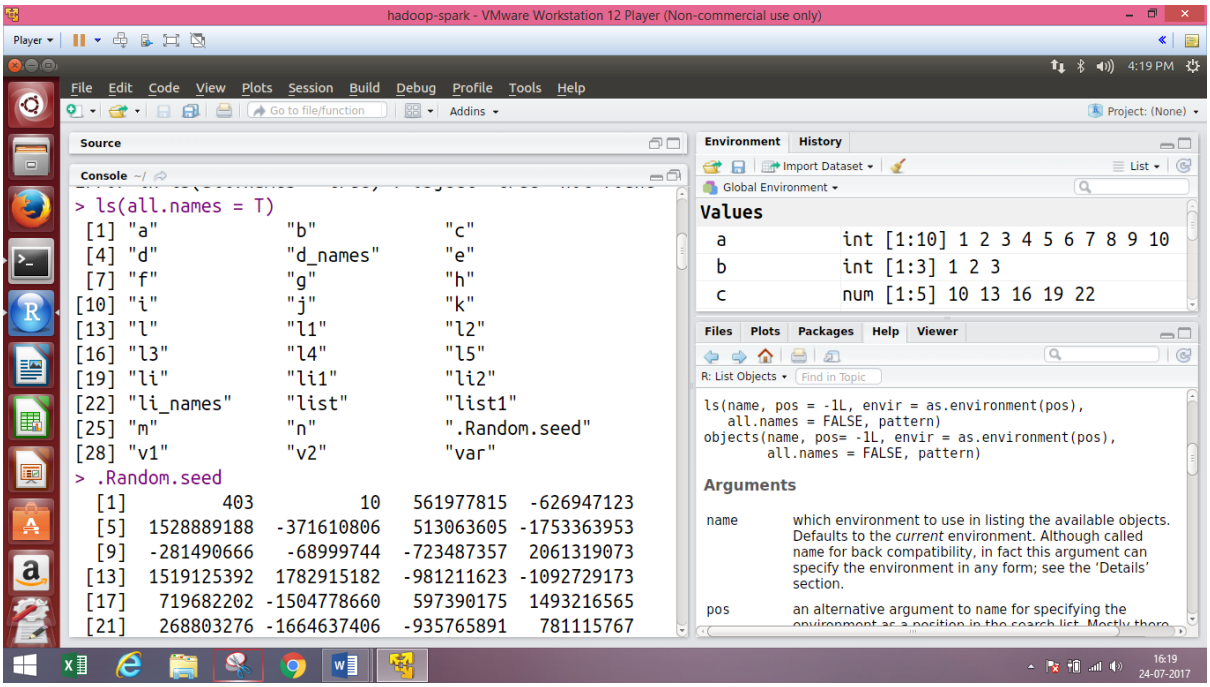


Figure 56: See Hidden Seed Value Arrangement.

- **Sys.Time()** and **Sys.Date()** function:
  Sys.time and Sys.Date returns the system's idea of the current date with and without time.
  Sys.time returns an absolute date-time value which can be converted to various time zones and may return different days.

Sys.Date returns the current day in the current timezone.

Use:

Sys.time()
Sys.Date()

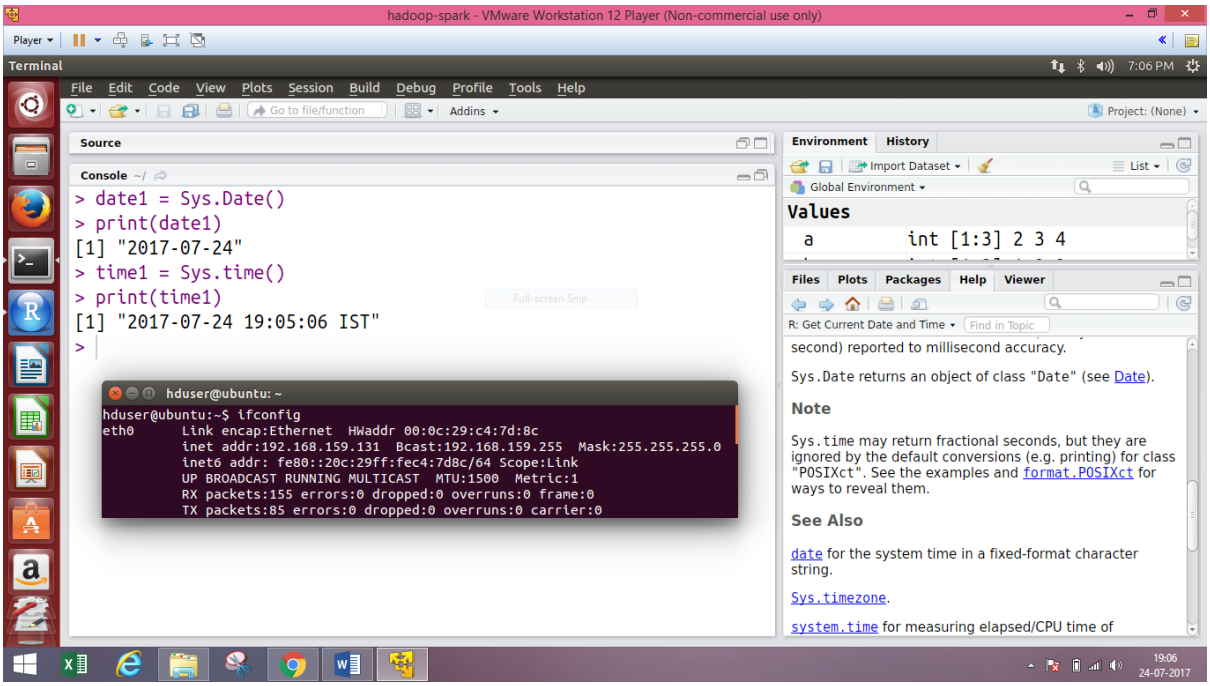


Figure 57: Get System Date and Time

- Set Operations:
  Performs set union, intersection, (asymmetric!) difference, equality and membership on two vectors.
  Each of **union()**, **intersect()**, **setdiff()** and **setequal()** functions will discard any duplicated values in the arguments, and they apply **as.vector()** function to their arguments (and so in particular coerce factors to character vectors).
  is.element(x, y) is identical to **x %in% y**.
  Use:

  union(x, y)
  intersect(x, y)
  setdiff(x, y)
  setequal(x, y)
  is.element(el, set)

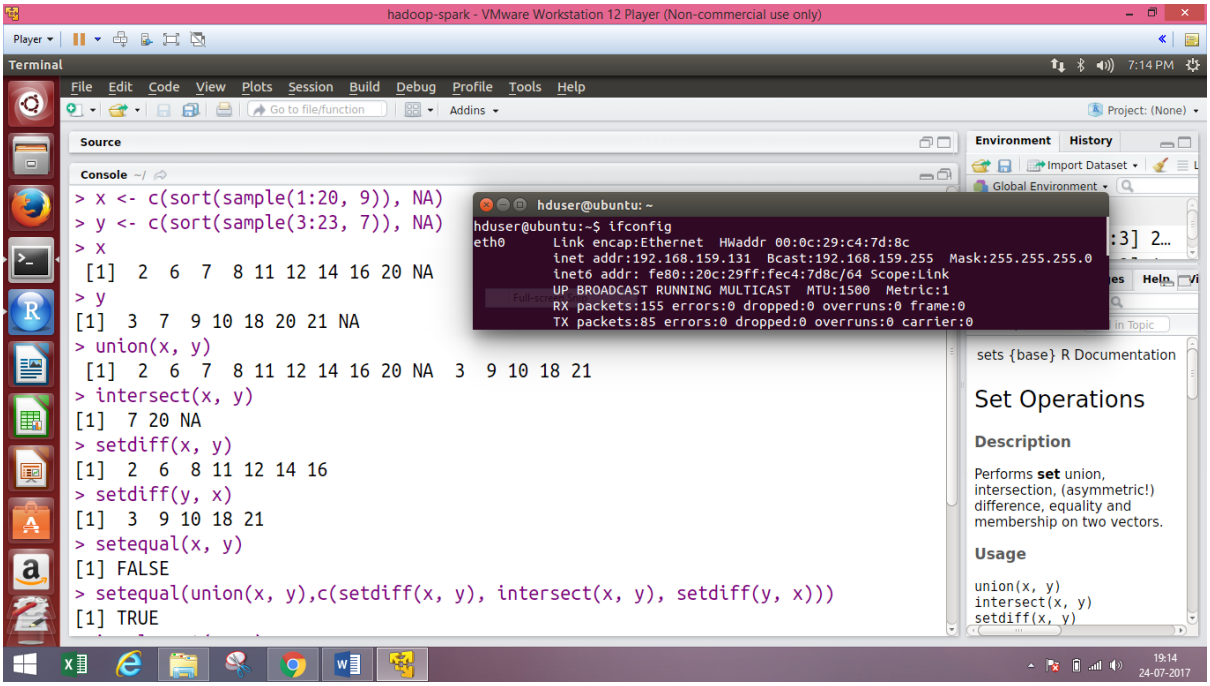


Figure 58: set operations

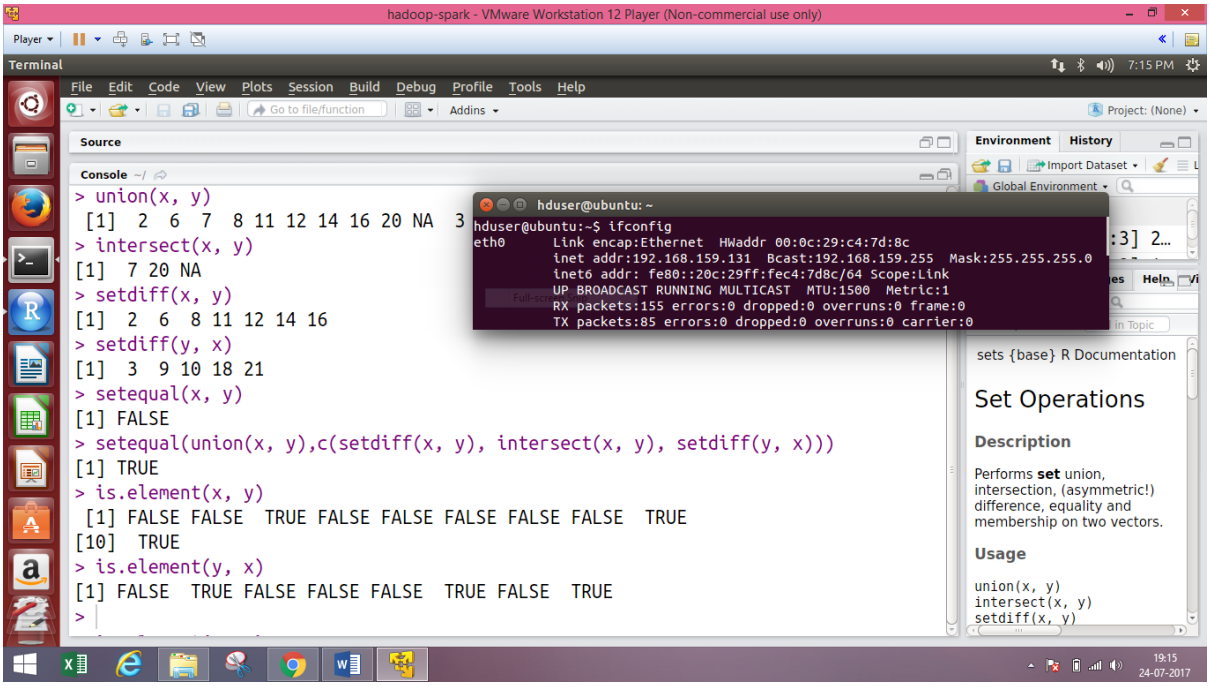


Figure 59: set operations

- **any()** function: This function will check for condition provided as parameter and returns TRUE or FALSE accordingly.

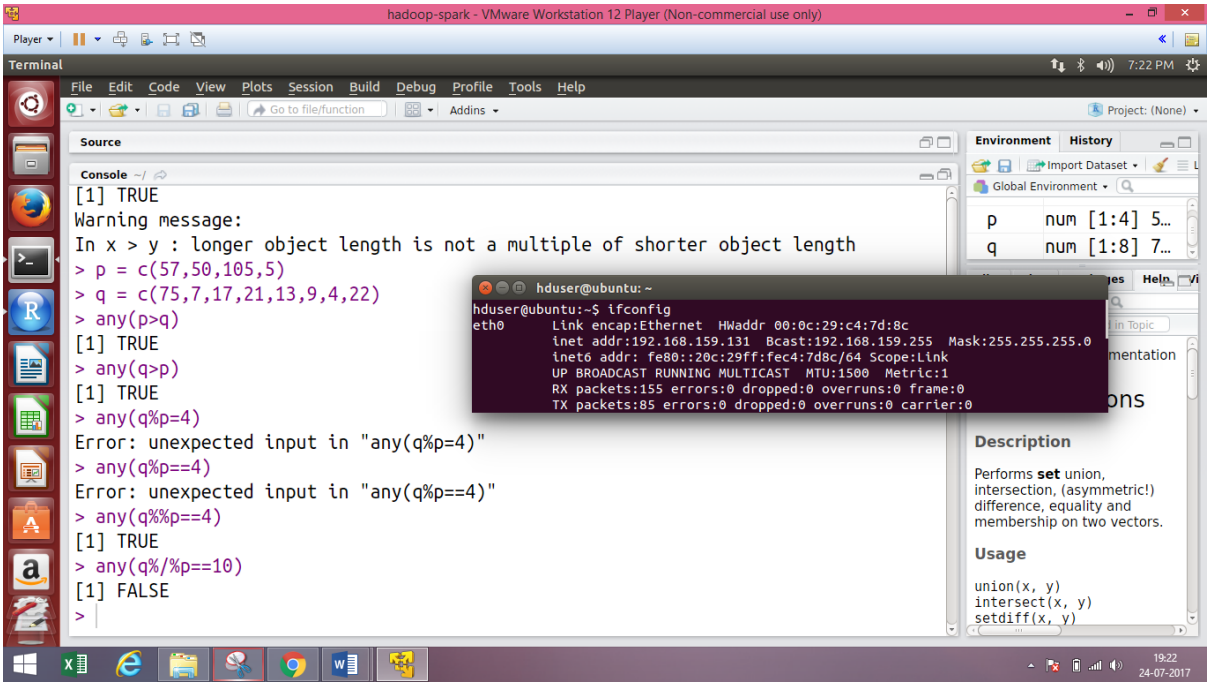


Figure 60: any() function

5. Working Directory:

Working directory value is said to be an absolute filepath: there can be more than one representation of the path to a directory and on some OSes the value returned can differ after changing directories and changing back to the same directory (for example if symbolic links have been traversed). Also GETWD and SETWD.

6. Decision Making Conditions:

Decision making structures require the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be **TRUE/T**, and optionally, other statements to be executed if the condition is determined to be **FALSE/F**.

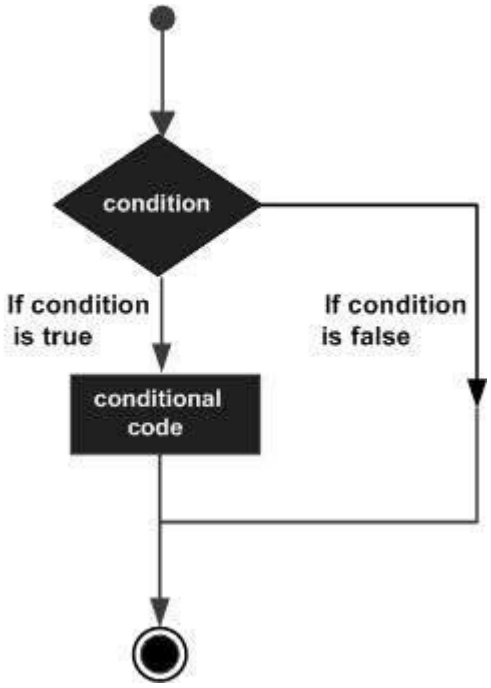


Figure 61: Condition Flow chart

- **IF Statement**
  In this condition type an if statement consists of a Boolean expression followed by one or more statements.

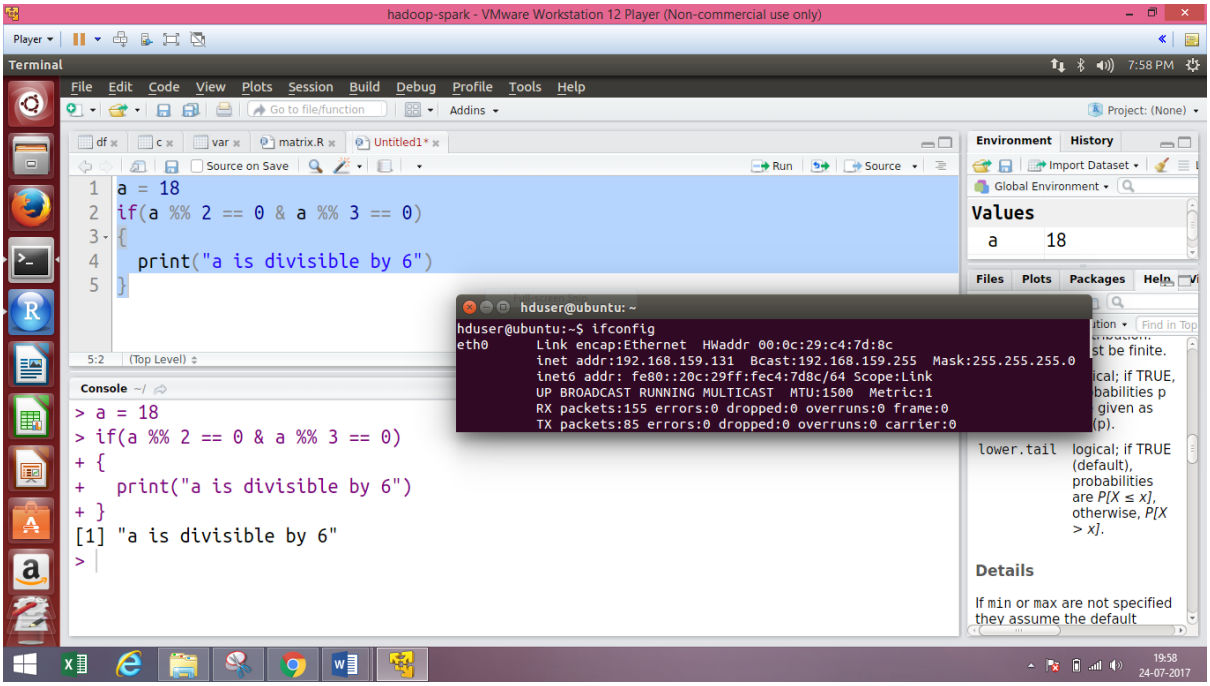


Figure 62: If condition

- **IF…ELSE Statement**
  In this condition type an if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

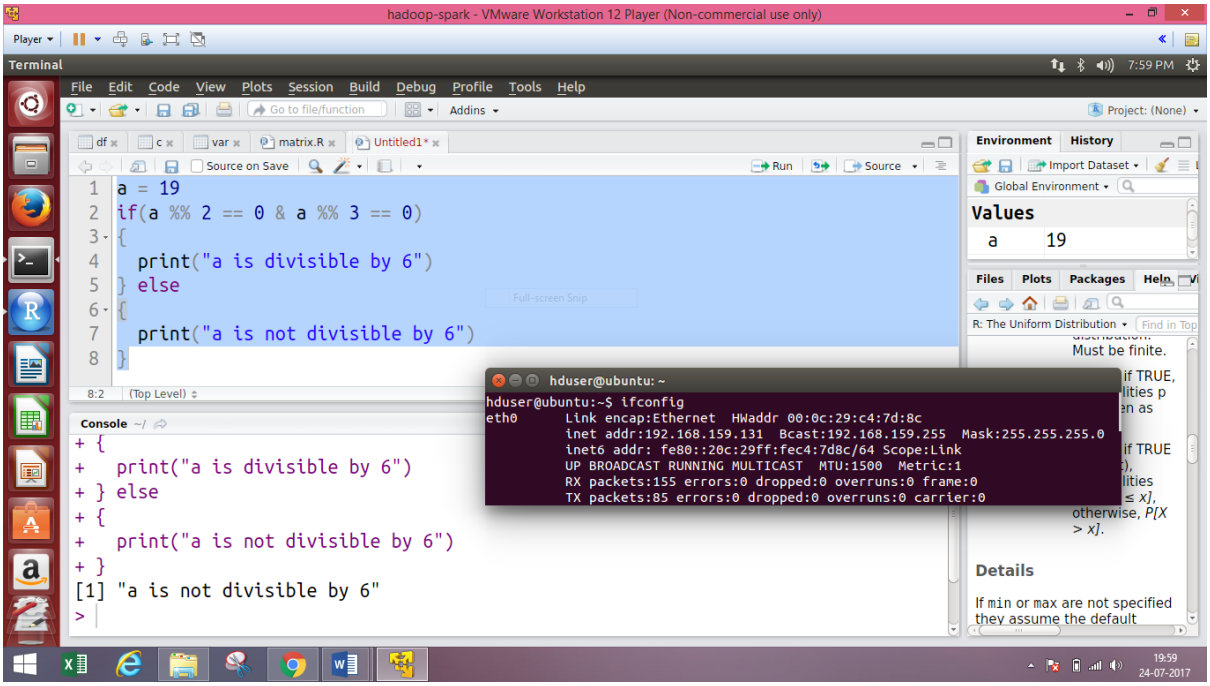


Figure 63: If - Else condition

- **IF…ELSEIF(n)…ELSE Statement**
  In this condition type an if statement can be followed by any number of else if statement and then an optional else statement, which executes when the Boolean expression is false.

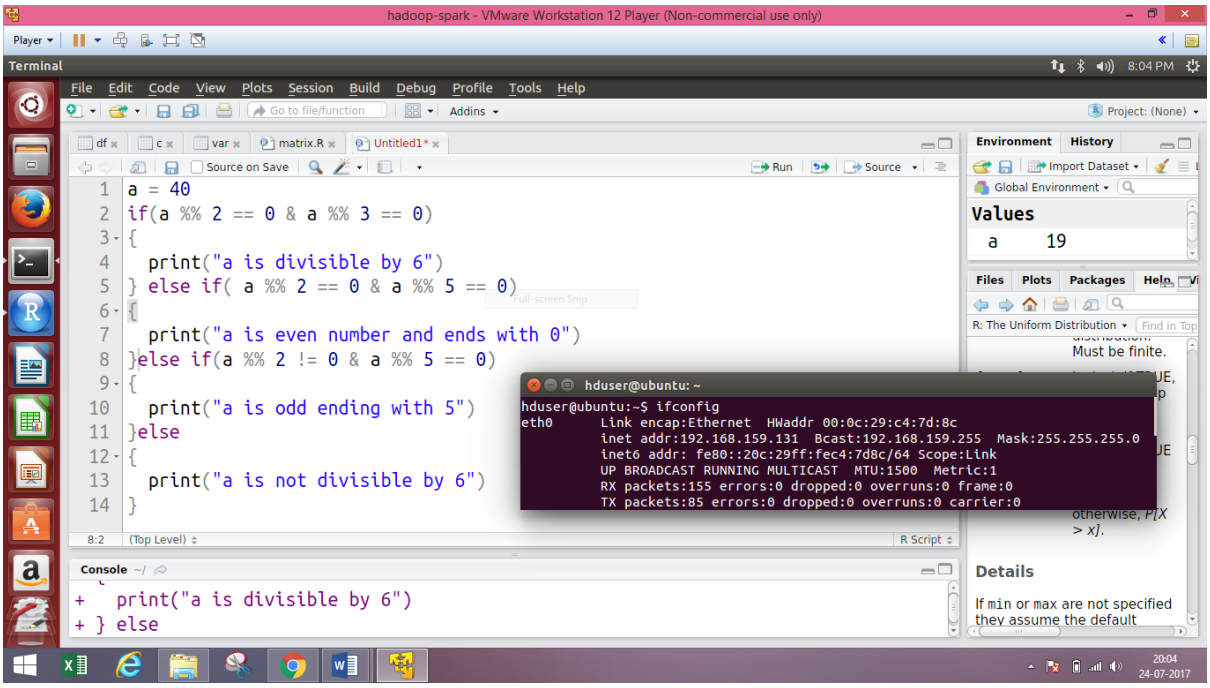


Figure 64: If-ElseIF-Else condition

- **Nested If Condition:** If condition block is applied inside other if condition.

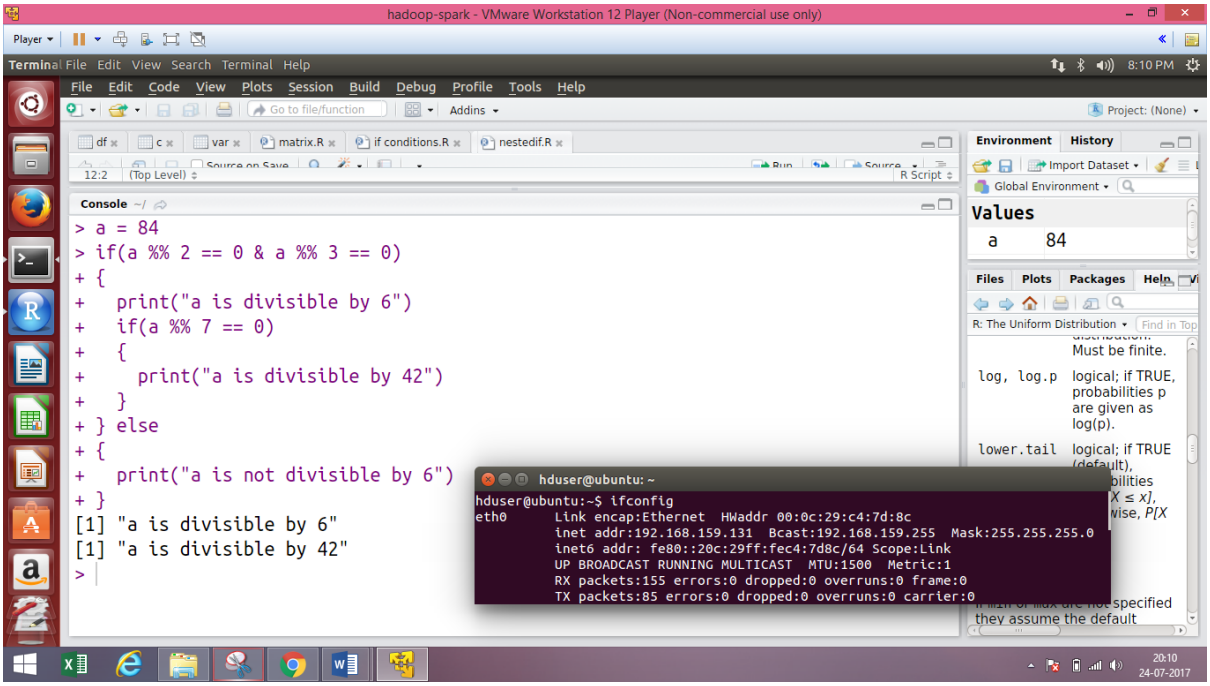


Figure 65: nested if (1/2)

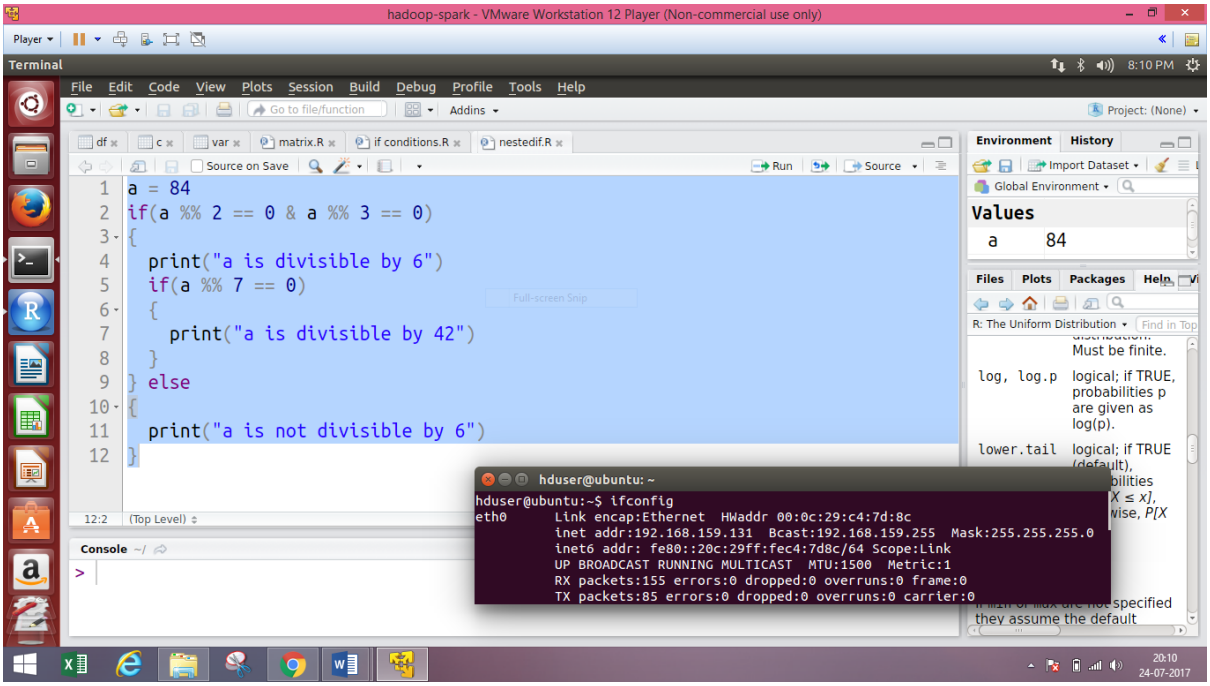


Figure 66: nested if (2/2)

- **Switch Case**

  A switch statement allows a variable to be tested for equality against a list of values.

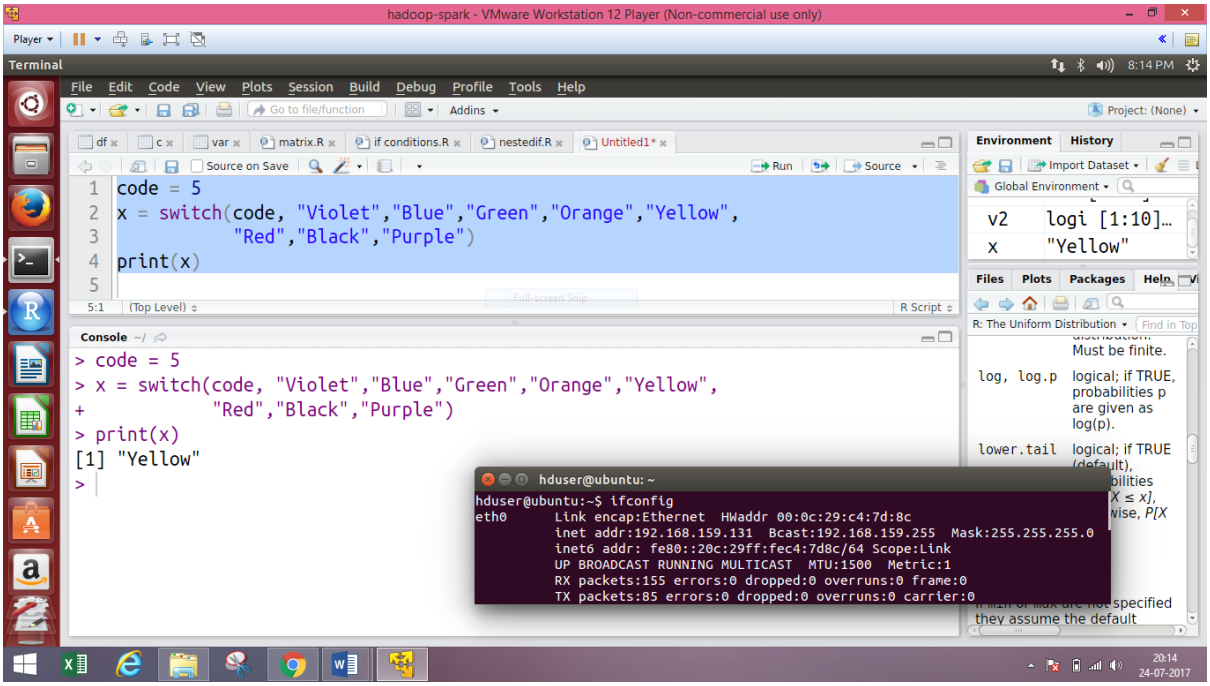


Figure 67: Switch Case

7. Uniform Distribution System

- **runif(), dunif, punif, qunif:** see runif() function.
  dunif gives the density, punif gives the distribution function, qunif gives the quantile function, and runif generates random deviates.

  The length of the result is determined by n for runif, and is the maximum of the lengths of the numerical parameters for the other functions.

  The numerical parameters other than n are recycled to the length of the result. Only the first elements of the logical parameters are used.

  Use:

  dunif(x, min = 0, max = 1, log = FALSE)
  punif(q, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)
  qunif(p, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)
  runif(n, min = 0, max = 1)

8. .Random.seed:

.Random.seed is an integer vector, containing the random number generator (RNG) state for random number generation in R. It can be saved and restored, but should not be altered by the user. To manipulate this seed, **set.seed()** function is used where seed is a single value, interpreted as an integer, or NULL. See seed.

Use:

.Random.seed <- c(rng.kind, n1, n2, \dots)
set.seed(seed, kind = NULL, normal.kind = NULL)

9. Working with lists:

- **Create list:** See LIST
- **Access list:**

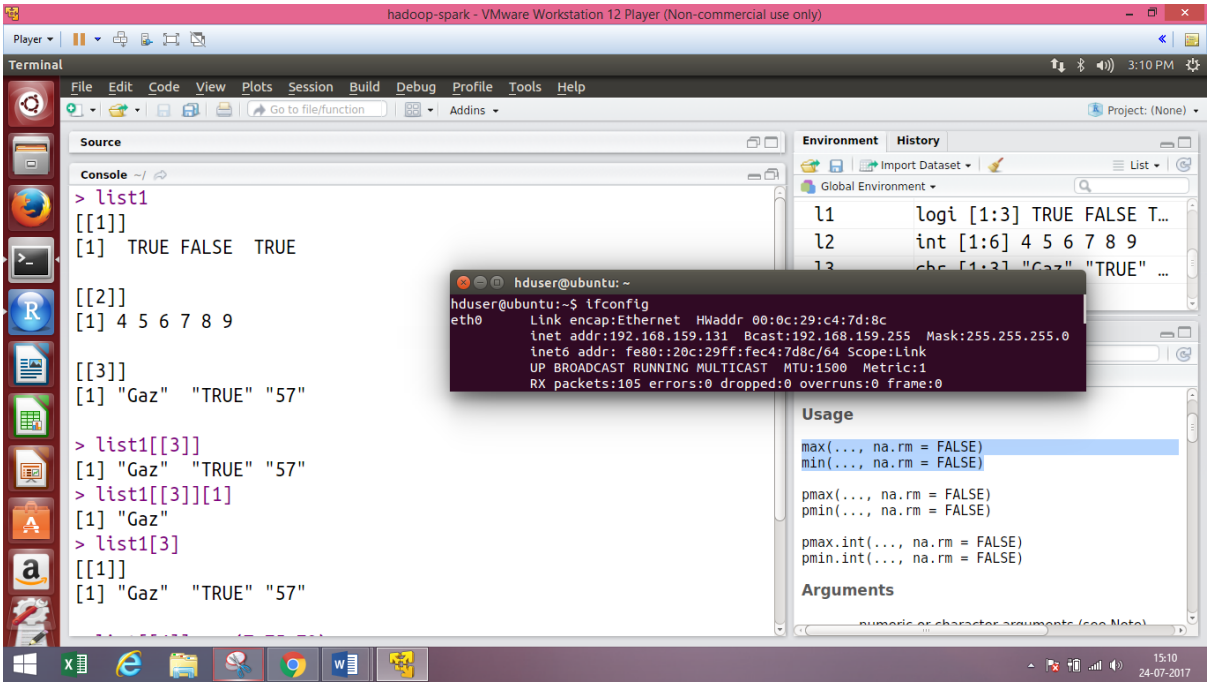


Figure 68: Access List Elements

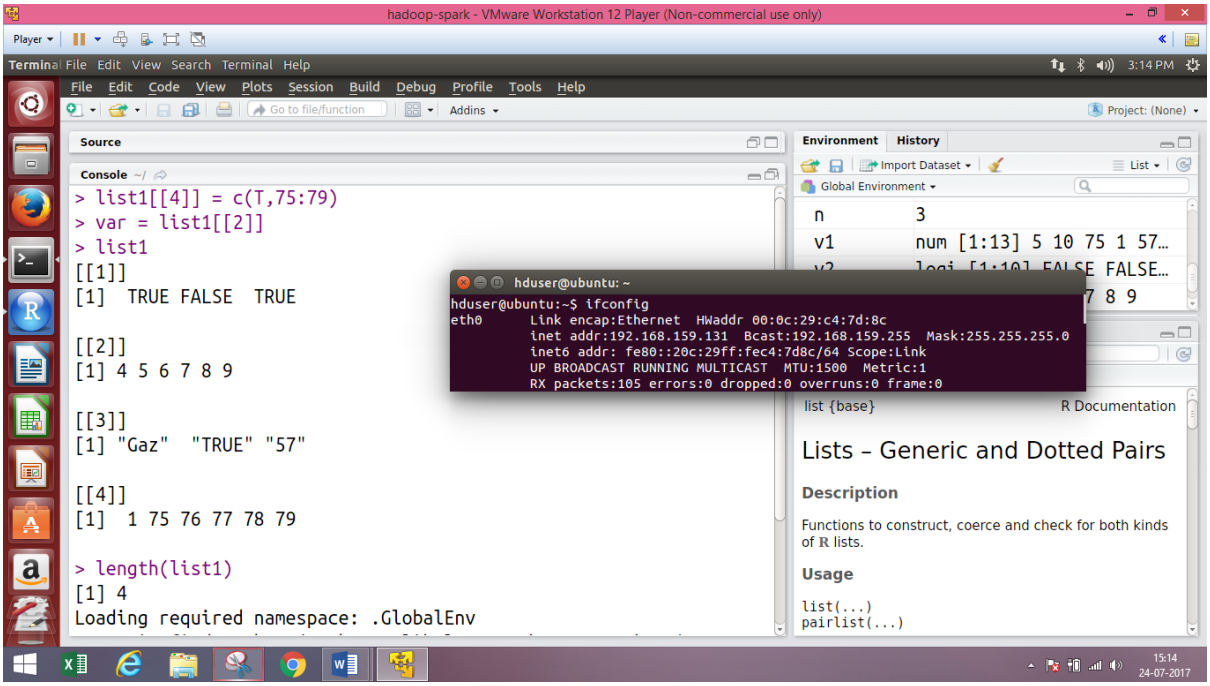


Figure 69: Access List Elements

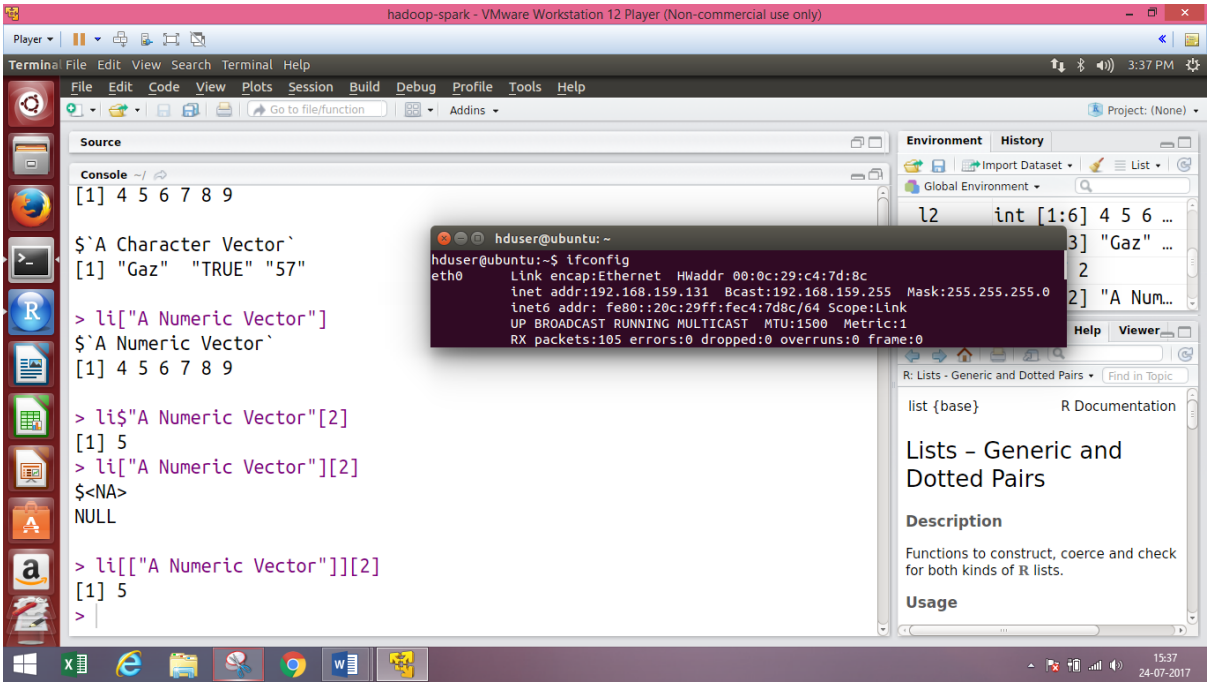


Figure 70: Access List with Names

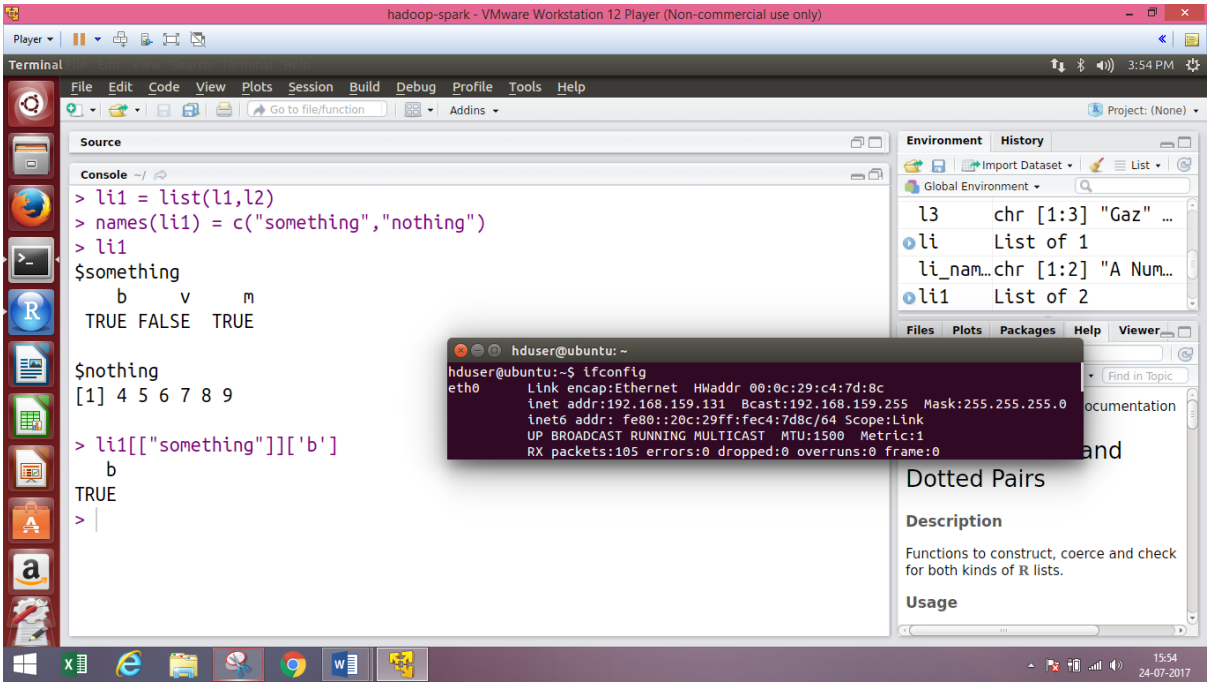


Figure 71: Access list element with list and vector name

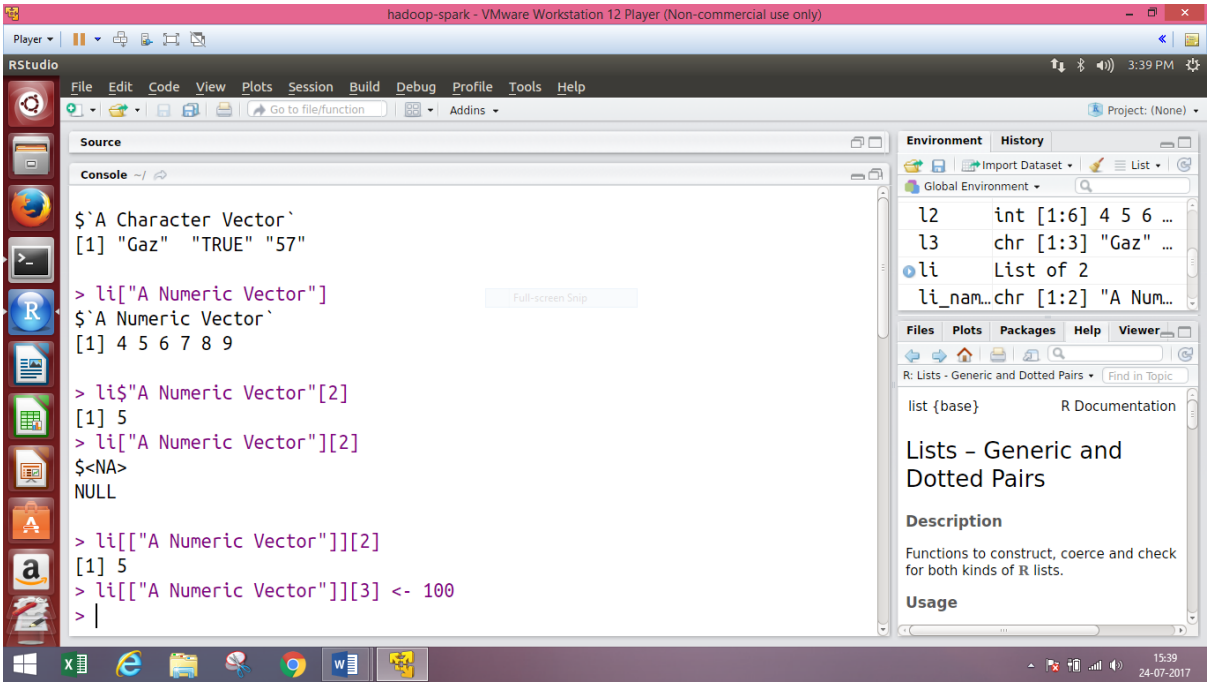


Figure 72: Access List with names

- **Names of list:**

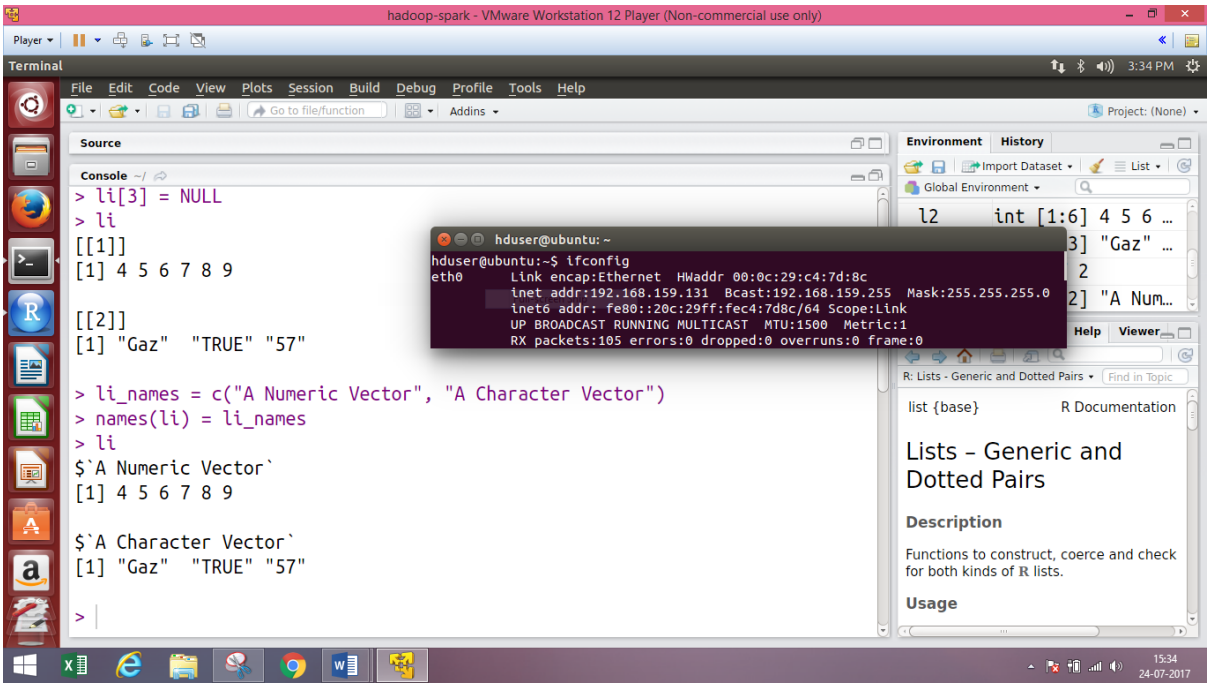


Figure 73: Naming the list

- **Unlist:** See UNLIST
- **Length of list:**

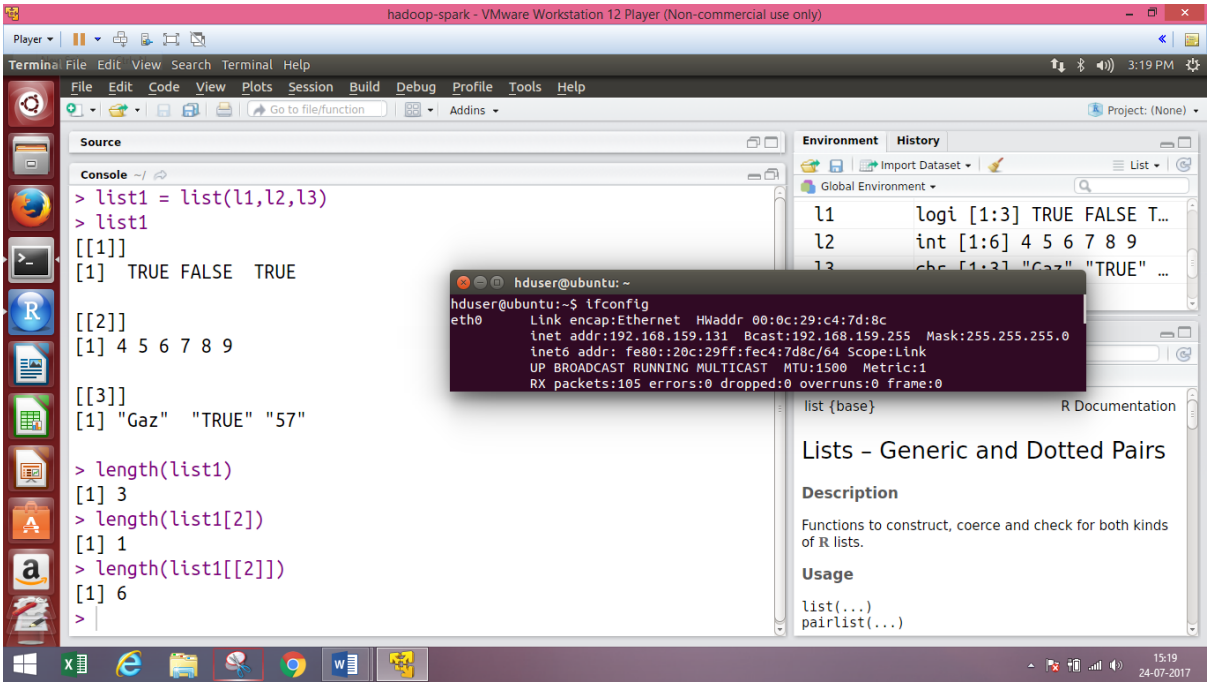


Figure 74: list length

- **List Manipulation:**

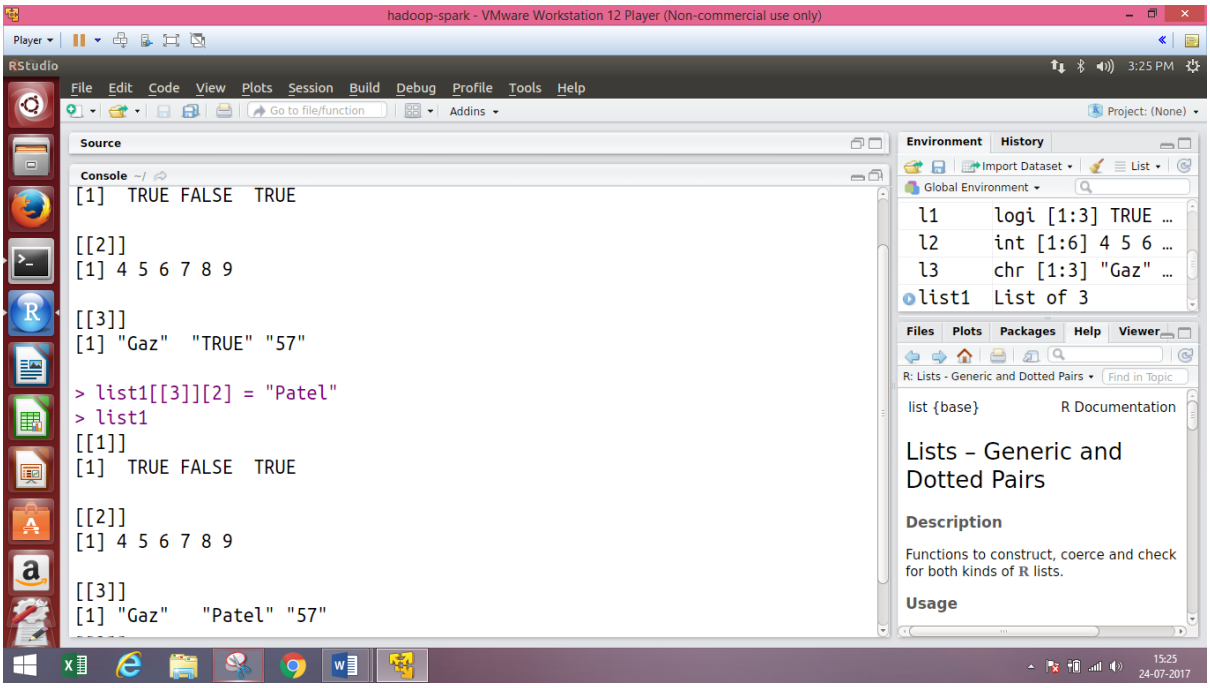


Figure 75: Modify element in List

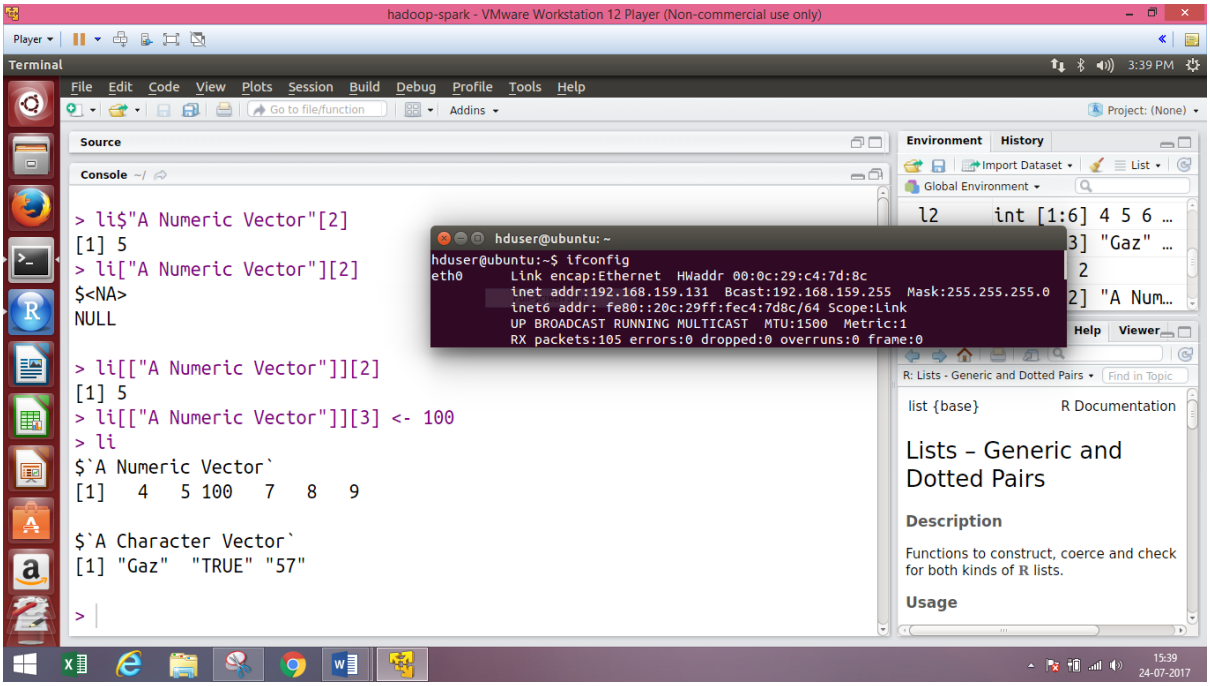


Figure 76: Modify List with Names

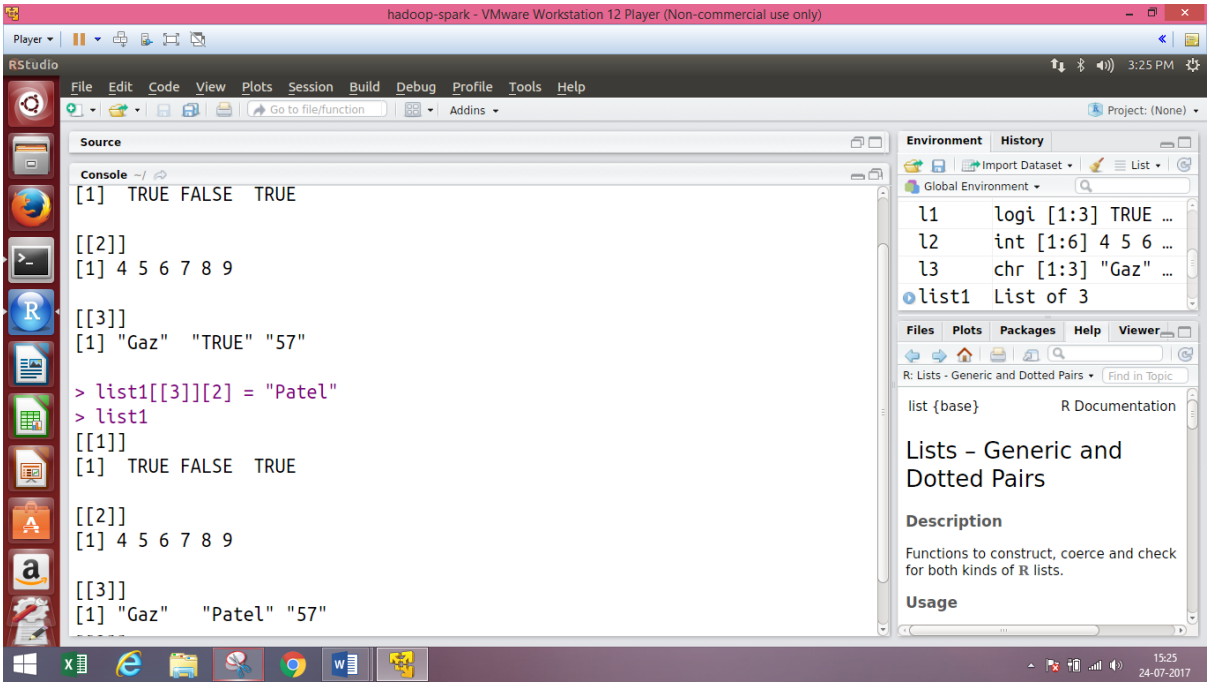


Figure 77: Add List Element

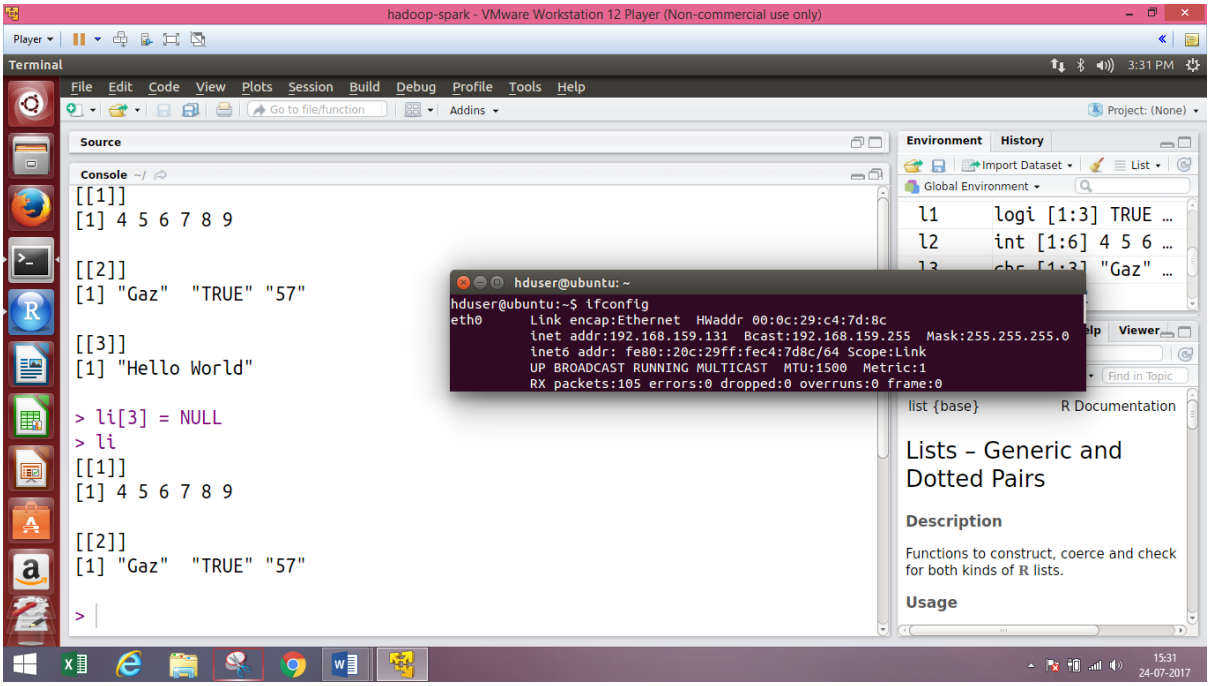


Figure 78: Delete List Element

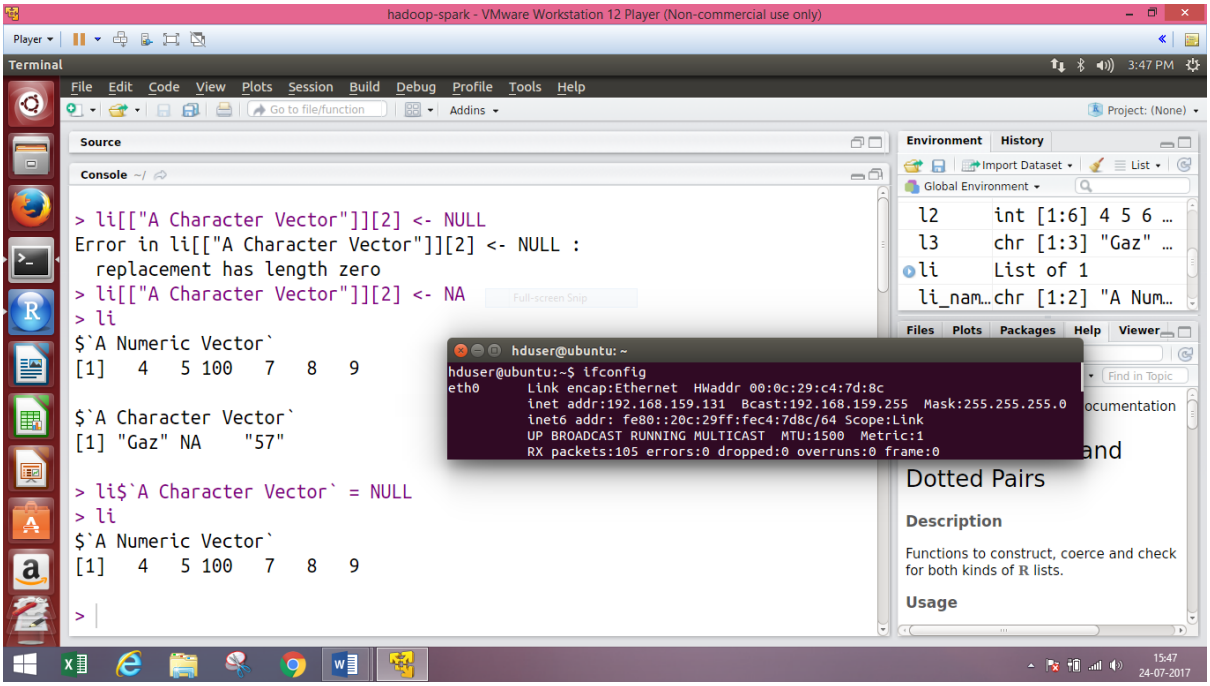


Figure 79: Delete List Element with Names

- **Merge lists:**

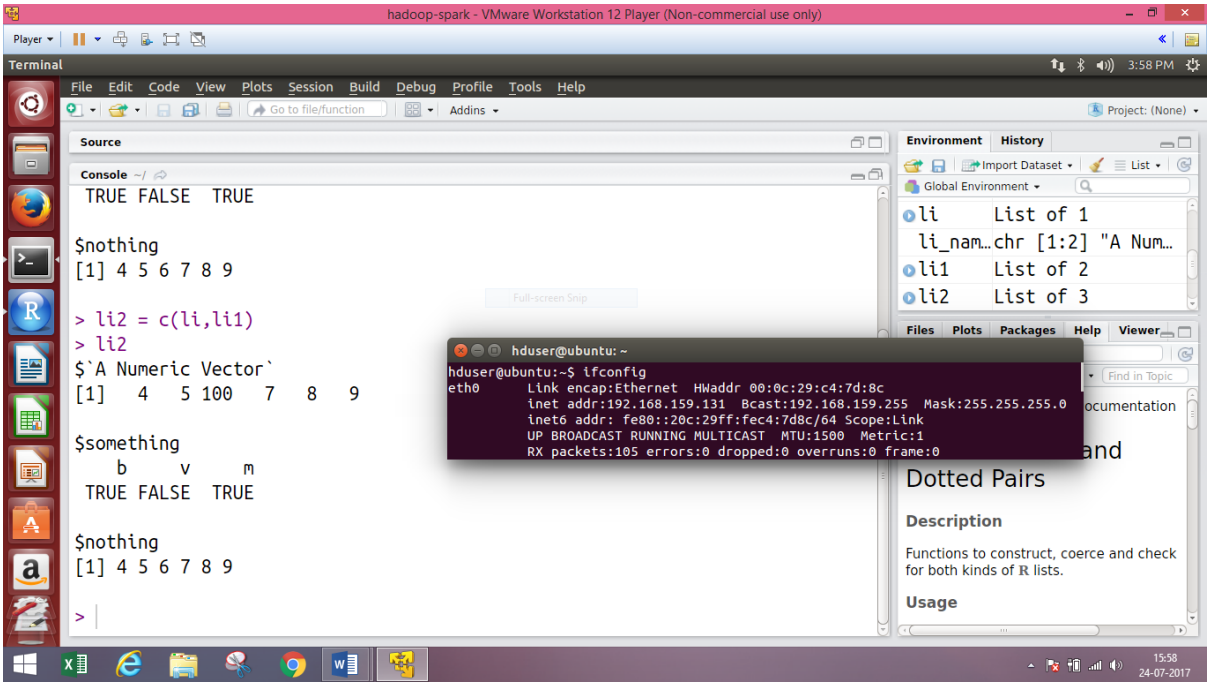


Figure 80: Merging Lists

10.Working with Matrix:

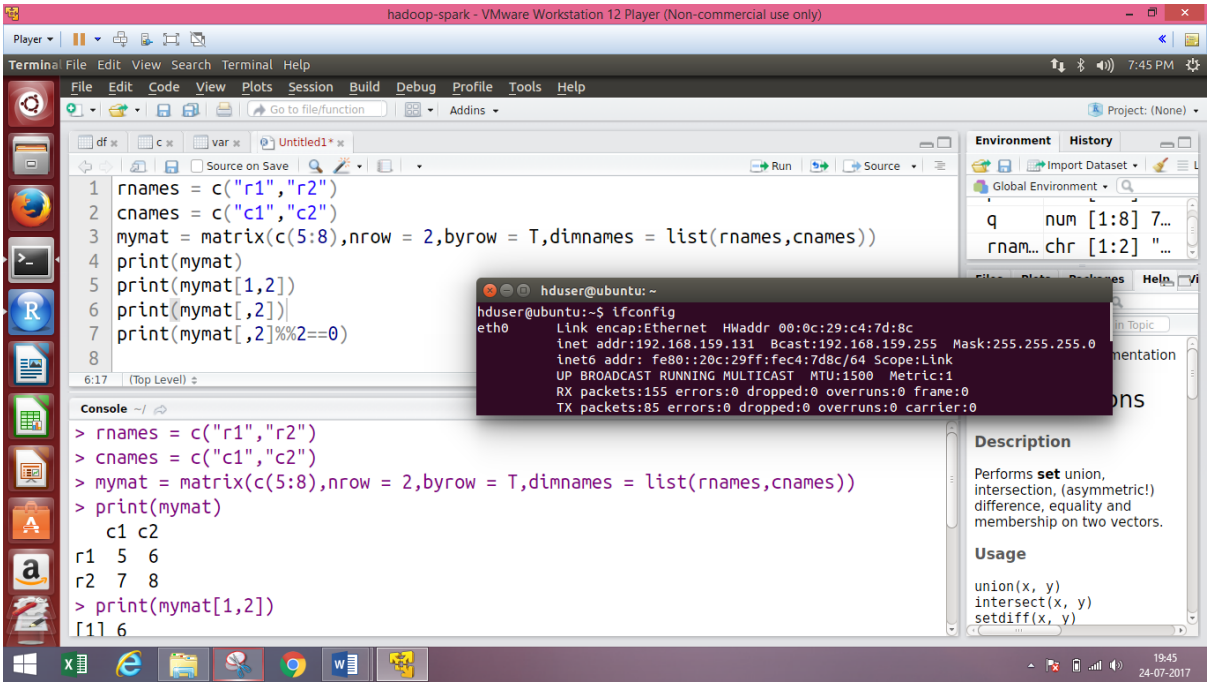


Figure 81: Create Matrix

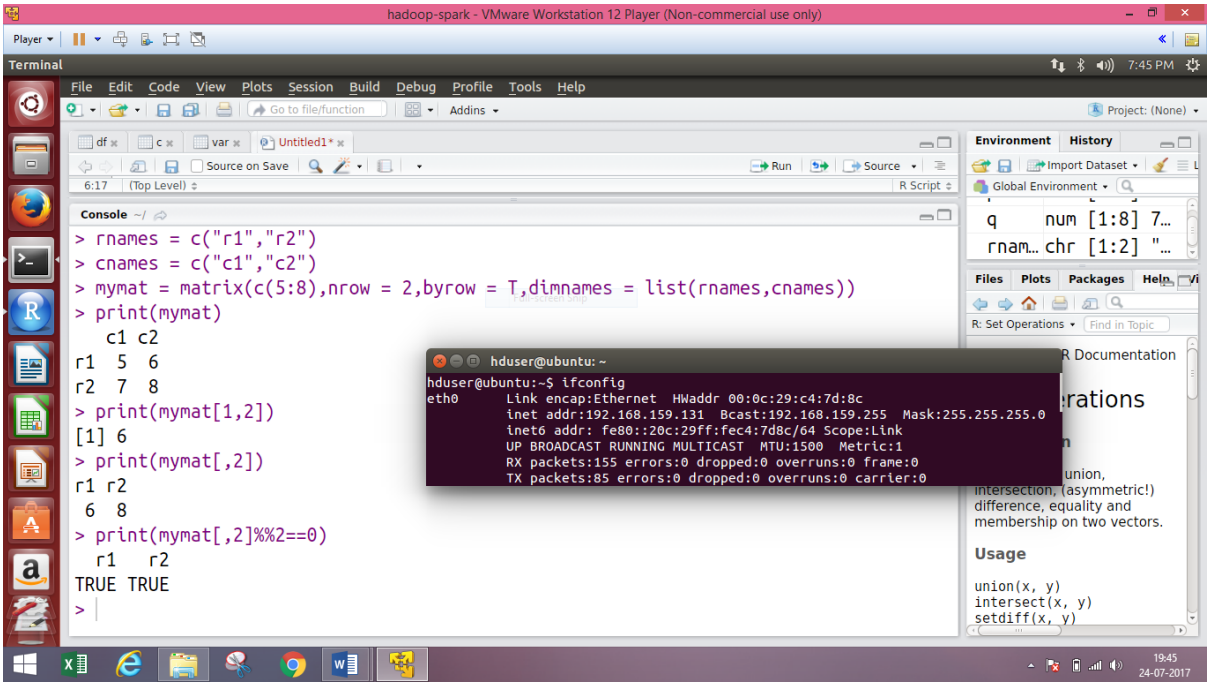


Figure 82: Access Matrix

11. Working with Randoms: See runif() and seed() functions.

12. Programs:

- **Write a program in R where you would initialize a vector with integers ranging from 1 to 10, take a sample from that vector and check if the number is greater than 7 or not.**

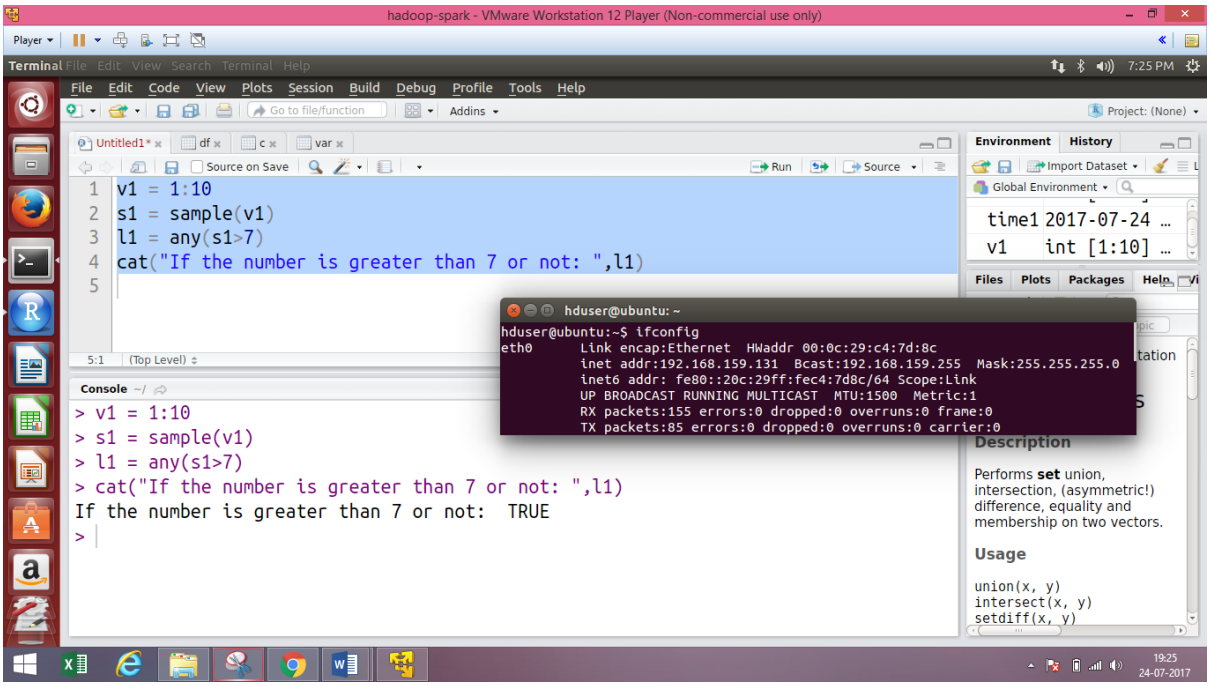


Figure 83: Program 1

- **Write a program to initiate two objects and swap their contents**

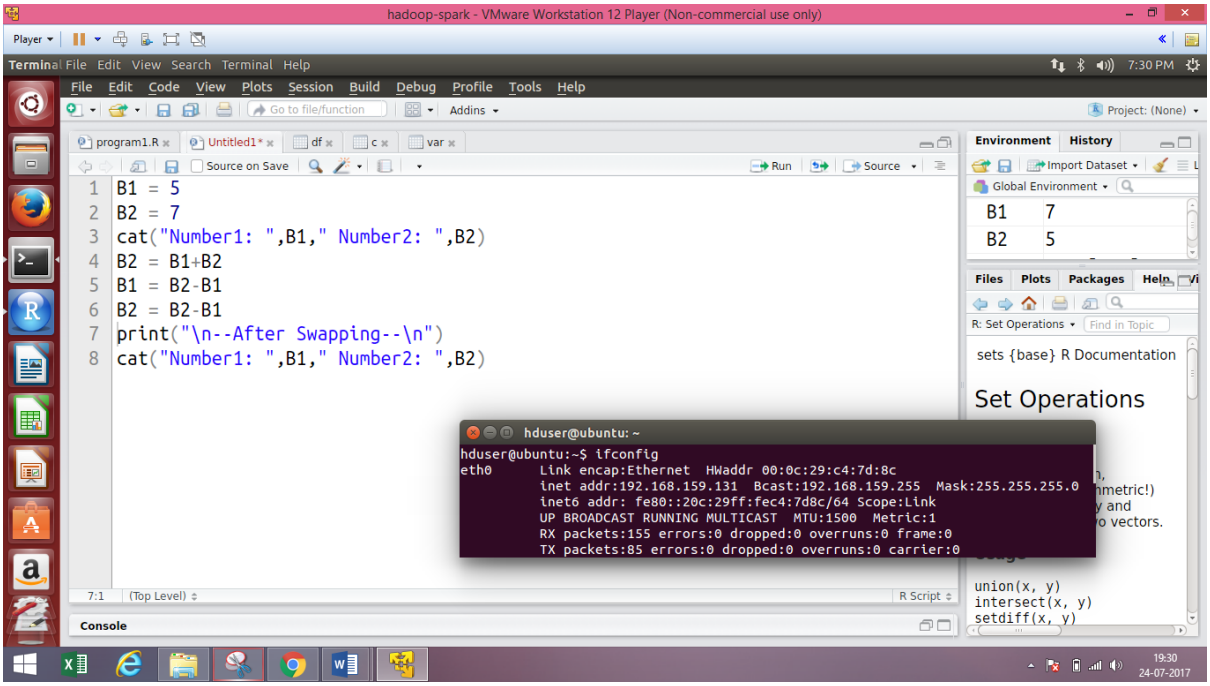


Figure 84: Program 2 - Method1- (1/2)

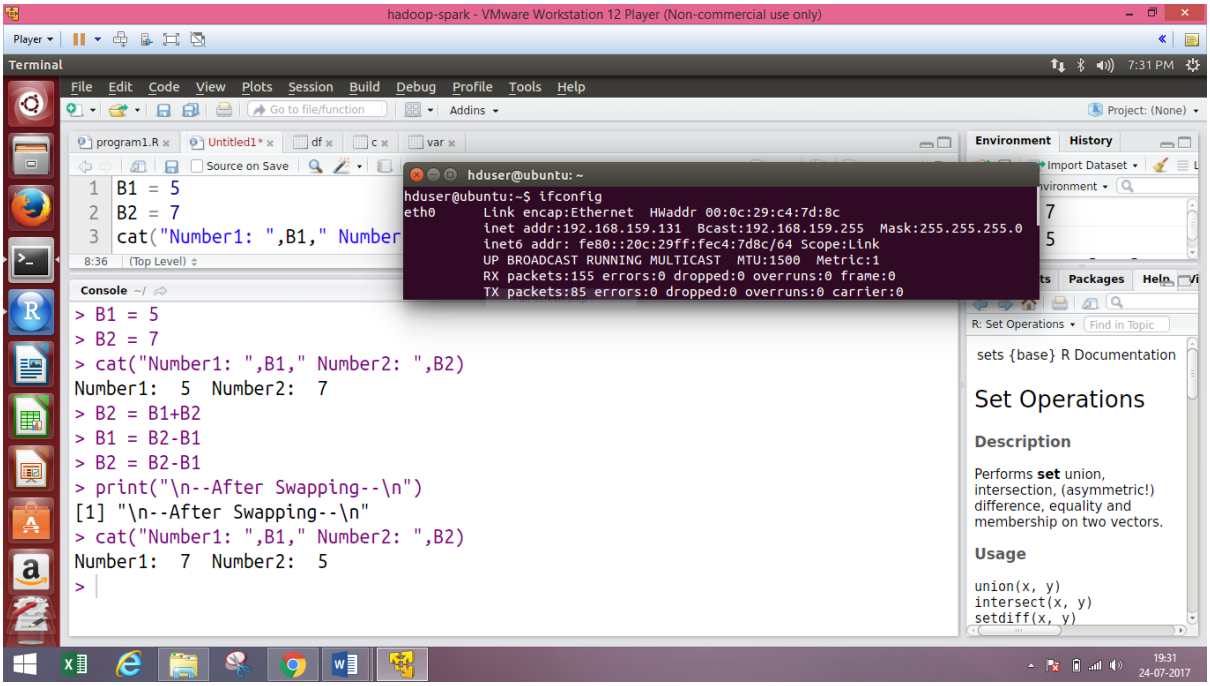


Figure 85: Program 2 - Method 1 - (2/2)

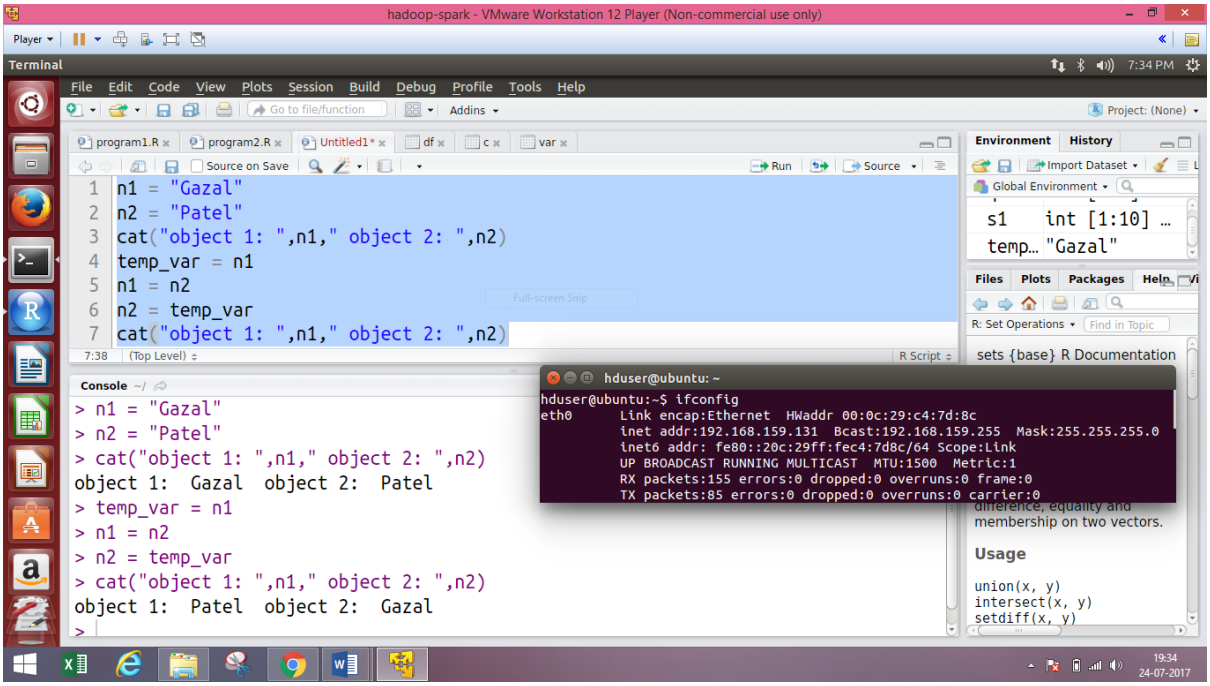


Figure 86: Program 2 - Method 2

- **Initialize 2 vectors 'a' and 'b' with 5 numbers each, divide each elements of vector 'a' with that of vector 'b' and store the result into a new vector.**

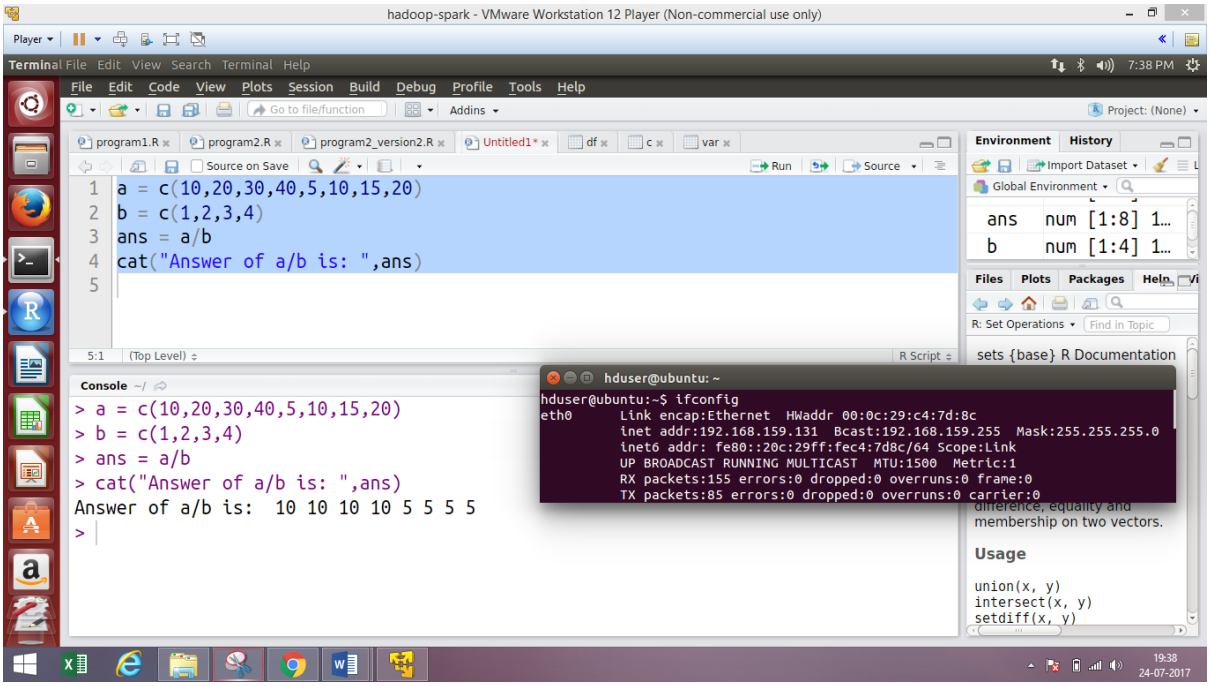


Figure 87: Program 3