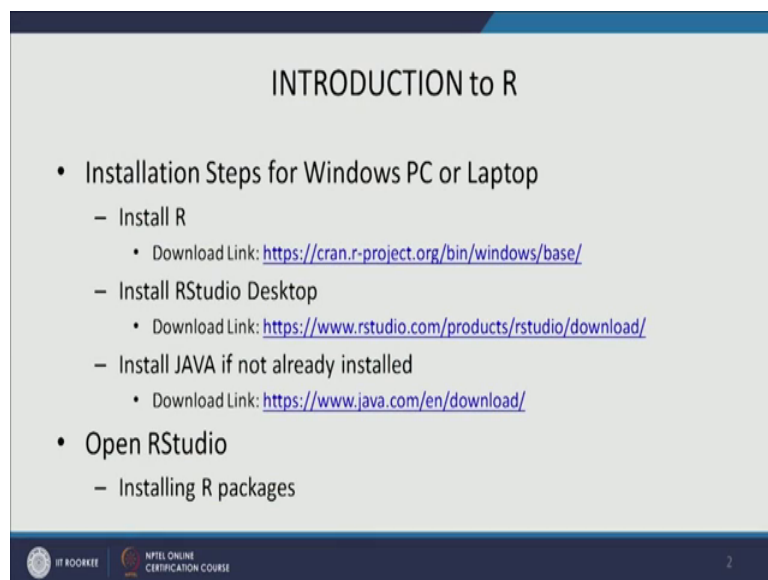


Business Analytics & Data Mining Modeling Using R
Dr. Gaurav Dixit
Department of Management Studies
Indian Institute of Technology, Roorkee

Lecture - 03
Introduction to R

Welcome to the supplementary lecture number 1 of the course business analytics and data mining modeling using R. So, this particular lecture is about introduction to R.

(Refer Slide Time: 00:34)



INTRODUCTION to R

- Installation Steps for Windows PC or Laptop
 - Install R
 - Download Link: <https://cran.r-project.org/bin/windows/base/>
 - Install RStudio Desktop
 - Download Link: <https://www.rstudio.com/products/rstudio/download/>
 - Install JAVA if not already installed
 - Download Link: <https://www.java.com/en/download/>
- Open RStudio
 - Installing R packages

IT ROORKEE NPTEL ONLINE CERTIFICATION COURSE 2

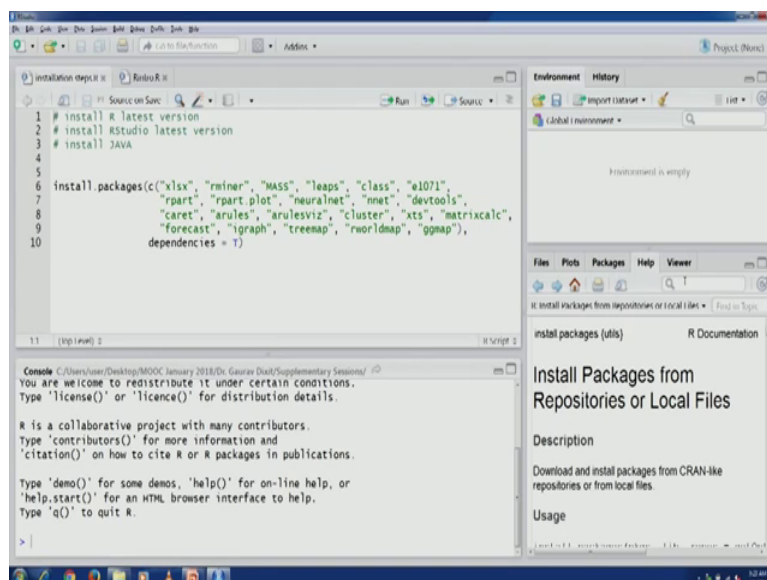
So, we are going to cover basic introductions; we are going to cover basics of R. So, before we start let us understand the installation steps, this is a specifically for windows pc or laptops that is the expectation that many of the students they would be having windows pc or laptop and these instructions are for the same.

So, first you need to install R, so the a link is given here and depending on the your operating system whether it is 32 bit or 64 bit you can download the appropriate file; installation file and the after once you are done with installing R, then you can go ahead and install this R studio desktop version, this is the GUI for R. So, the link is, download link is already given here, again depending on settings all configuration of your desktop or laptop, you can download the

appropriate installation file 32 bit or 64 bit and then go ahead with your installation.

Now, because R studio is actually based on java, so therefore, if your pc or laptop does not already have java, then you can install the java has there before proceeding further. So, the link for java is already been given, so you can download and install it as well. Now, before we can start our data mining modeling in R, we need to have certain R packages installed, so that they could be used many of the functions and libraries could be loaded and the function are available for us to use for modeling. So, therefore, installing R packages is the next step, so what will do?

(Refer Slide Time: 02:27)

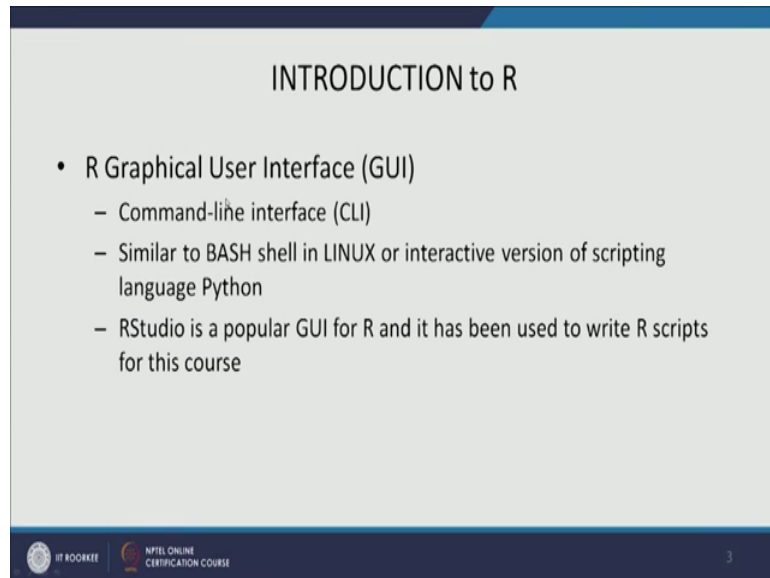


So, these are some of the packages that I have mentioned in this so installed R packages is 1 particular function that is used for installing R packages in your system. So, some of the packages that we are going to use in this particular course I have mentioned then here. So, you can see installed R packages and c is the combined function, which can combined all the strings or names of the packages and in 1 go all these packages would be installed, you would also see I have assigned dependencies the another argument as true.

So, if there are any dependencies for these packages, they would also be installed. So, once you are done with your installation of R and R studio and

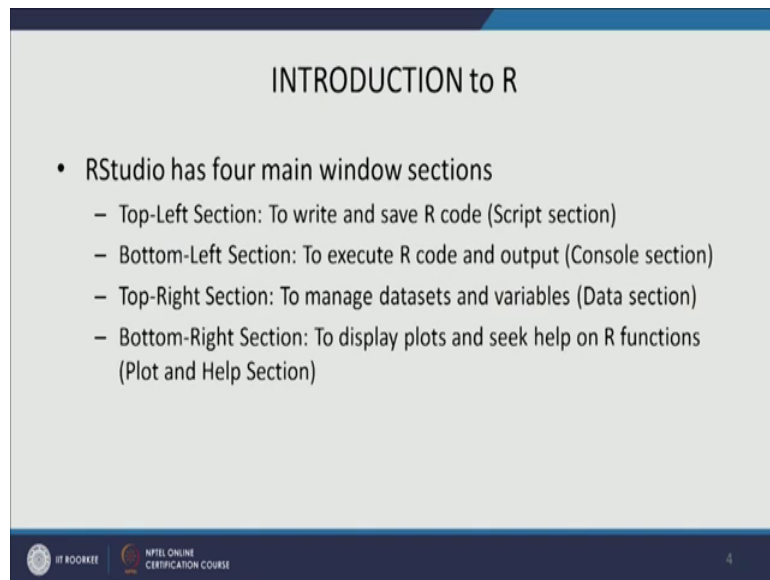
java as well then you can go ahead and use this particular function, this particular code to install these packages.

(Refer Slide Time: 03:23)



Now, let us understand about R's GUI. So, R's geographical user interface, it is mainly command line interface, it is quite similar to the bash shell that we have in Linux or interactive version of this is scripting language python. So, many people use python as well for data mining modeling and a statistical modeling. So, the interactive version of that particular language is also quite similar, is also based on command line interface. Now, R studio is one of the popular GUI for R and it has been used for this particular course, most of the R scripts have been written in this using this particular GUI R studio.

(Refer Slide Time: 04:07)



Now, as we have seen before R studio has a 4 main window sections we can see it again. This particular section is the top left section and generally we write and save our R code here in this particular section, you can see the instruction for installing packages is written in this section and has been saved as a file called installation steps dot R. So, most of the R scripts would have this extension name R dot R, so this particular file is saved as installation steps dot R.

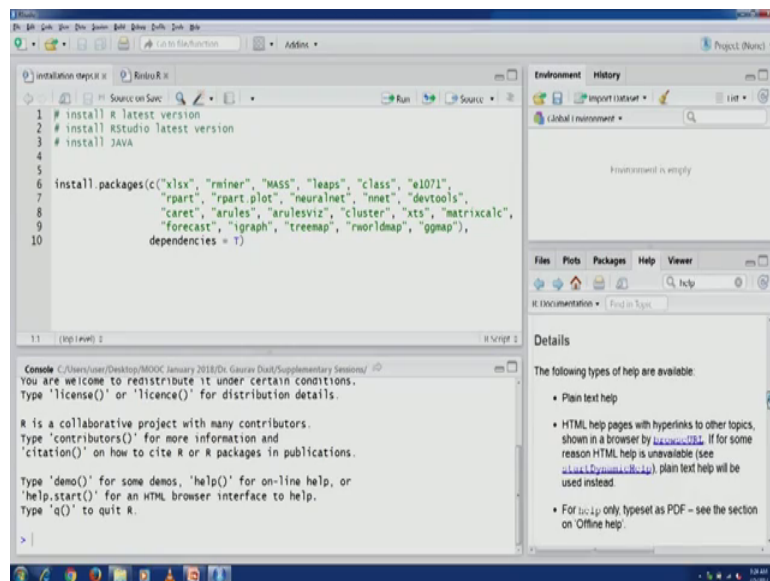
Now, bottom left section that is for actually execution of R code and to display output; to generate output is also called a console sections. So, this can be called a script section and this is called console sections here we actually see installation of execution of the R code and the related output.

Then, 3rd window section is top right section which is actually can be called data section or environment section, in this particular section we manage our data sets and variables. As we will see later in this particular lecture that many variables that we initialize are assign some values to those variables or data sets they would be a visual here, once they are loaded into R studio or R environment.

Now, the 4th window section is bottom right section, where we actually display plots and seek help for R functions, it can be called plot and help section. So, in this, particular plots right now the plot sub section is active and therefore, any

plots that we generate would be displayed here. Now, you also have help section, so if you are looking for a you know some help, related to some particular function in R you can type that particular name of the that particular function and the help is would actually be displayed there. So, you would see if we type help here, then in the documentation how the help function can actually be used and the details related to arguments and further details, examples, everything else and notes, references, examples, everything you can see over there.

(Refer Slide Time: 06:36)




So, whenever you face some difficulty in understanding how a particular function in R is supposed to be used, then you can always look for some help in this particular section.

(Refer Slide Time: 06:59)

INTRODUCTION to R

- Dataset Import
 - In this course, datasets are either imported from Excel files or created in RStudio
- Open Rstudio
 - R Basics

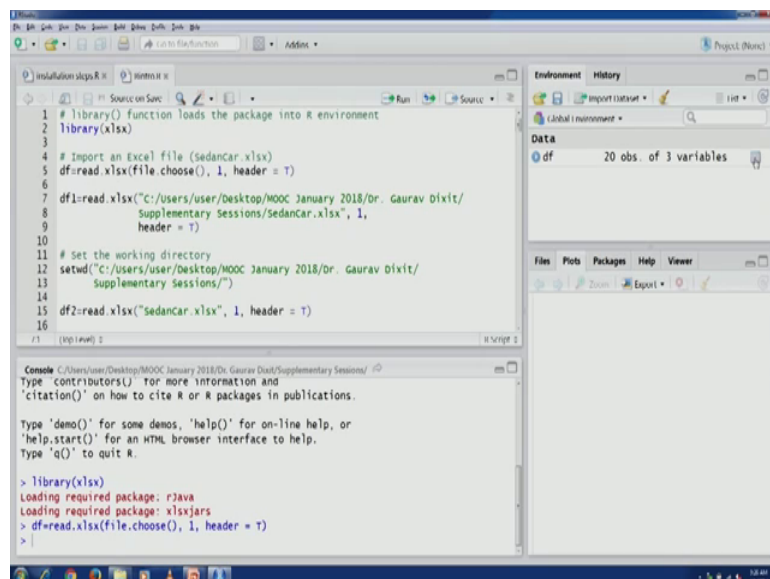


**NPTEL ONLINE
CERTIFICATION COURSE**

5

Now, data set import, this is in so far our course for this course data sets are mainly available in excel files. So, we would be importing data set from excel files or we would be creating in our studio itself. So, any hypothetical data set we would be immediately creating in R studio. So, let us with this information let us start our R basics or open R studio.

(Refer Slide Time: 07:31)



```
1 # library() function loads the package into R environment
2 library(xlsx)
3
4 # Import an Excel file (Sedancar.xlsx)
5 df=read.xlsx(file.choose(), 1, header = T)
6
7 df1=read.xlsx("C:/Users/user/Desktop/MOOC January 2018/Dr. Gaurav Dixit/
8             Supplementary Sessions/Sedancar.xlsx", 1,
9             header = T)
10
11 # Set the working directory
12 setwd("C:/Users/user/Desktop/MOOC January 2018/Dr. Gaurav Dixit/
13       Supplementary Sessions/")
14
15 df2=read.xlsx("Sedancar.xlsx", 1, header = T)
16
```

Console:

```
> library(xlsx)
Loading required package: rJava
Loading required package: xlsxjars
> df=read.xlsx(file.choose(), 1, header = T)
>
```

Environment:

Global Environment
df

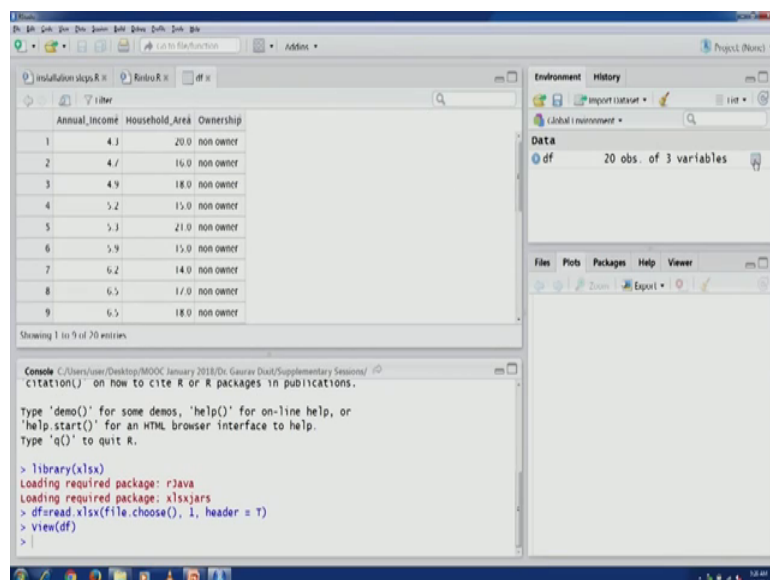
Data: df, 20 obs. of 3 variables

One of the important function that we require before we start is, understanding different packages and load them into the R environment for example, because

as we discussed the data sets would generally be imported from excel files, in this course see I have first line of this particular R script is about this library `xlsx`, so this is the package which can actually be used to import data set from a excel file.

You would see, here in the console section that this particular package has been loaded and the required package is also I have also been loaded. Now, if we are looking to you would like to import an excel file a data set. So, for example, the previous data set that we have used in the lecture 1 sedan car data set that we can actually import. You would see in this particular environment section or data section you would see a particular file has been imported.

(Refer Slide Time: 09:06)



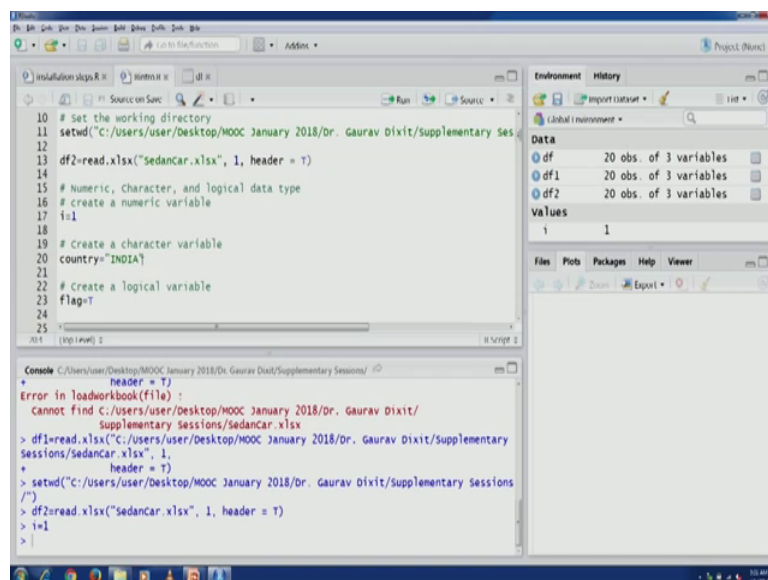
If you click on this particular link, a new window in the script section would be open; file section would be open and you can see the data set there. So, we have 3 variables in this particular data set annual income, household area and ownership and if you scroll down the data set then you can see 20 observation, the same is displayed in the data section as well, we have 20 observation and 3 variables.

Now, this is a 1 way of importing data from an excel file, in this case I have used this particular function `file.choose()` and this can actually allow this can actually allow us to browse for our file in our windows directories. Another way

to import an excel file into R environment could be using this function read dot xlsx and then giving the whole full complete path to the particular file.

One important difference that we can note here is, in R views forward slash for mentioning the complete path instead of the backward slash as is used in windows systems. So, do not forget to change from back slash to forward slash. Now, another way to import the data set is you can set your working directory, so this is the command. First let us see, how the, a data set can be imported using the full path name.

(Refer Slide Time: 10:52)



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

```
10 # Set the working directory
11 setwd("C:/Users/user/Desktop/MOOC January 2018/Dr. Gaurav Dixit/Supplementary Sessions/Sedancar.xlsx")
12
13 df2=read.xlsx("Sedancar.xlsx", 1, header = T)
14
15 # Numeric, Character, and logical data type
16 # create a numeric variable
17 i=1
18
19 # Create a character variable
20 country="INDIA"
21
22 # Create a logical variable
23 flag=T
24
25
```

The console at the bottom shows an error message:

```
Error in loadworkbook(file) :
cannot find C:/Users/user/Desktop/MOOC January 2018/Dr. Gaurav Dixit/Supplementary Sessions/Sedancar.xlsx
> df1=read.xlsx("C:/Users/user/Desktop/MOOC January 2018/Dr. Gaurav Dixit/Supplementary Sessions/Sedancar.xlsx", 1,
+ header = T)
> setwd("C:/Users/user/Desktop/MOOC January 2018/Dr. Gaurav Dixit/Supplementary Sessions/")
> df2=read.xlsx("Sedancar.xlsx", 1, header = T)
> i=1
>
```

The Environment pane on the right shows the data environment:

Object	Class	Attributes
df	data.frame	20 obs. of 3 variables
df1	data.frame	20 obs. of 3 variables
df2	data.frame	20 obs. of 3 variables

The values for the first row of df are shown as 1 and 1.

So, let us see how the excel file data set from excel file can be imported using the full path name of the file. So, you can see the same data it is actually the same data set, but second import you can see here df 1, 20 observation and 3 variables.

Another way of importing data set could be you can set your working directory using this command set wd and then, again you can give the full path name of your working directory, just execute this particular command and then, now because the your working directory has been changed where your excel file is located, then you can simply put simply write the name of this particular excel file and again import the data set, that says you this code and again you would see another file has been imported it is again same data set, so you can see.

3rd instance of the same data set being imported same 20 observation and 3 variables. Now, once you have imported your data set, you can use the data set to the start your modeling to execute different steps related to the particular modeling. So, for example, because this is an introduction to R, we are covering R's basics. So, therefore, we will try to understand some of the basic building block structures that are used in R.

So, first let us to understand the numeric character and logical data type, how are they are used in R? How we can create them and then access them? So, how a numerical variable is created in R? You just need to type this `i`, `i` could be numerical available for your code and then you can assign the value and execute.

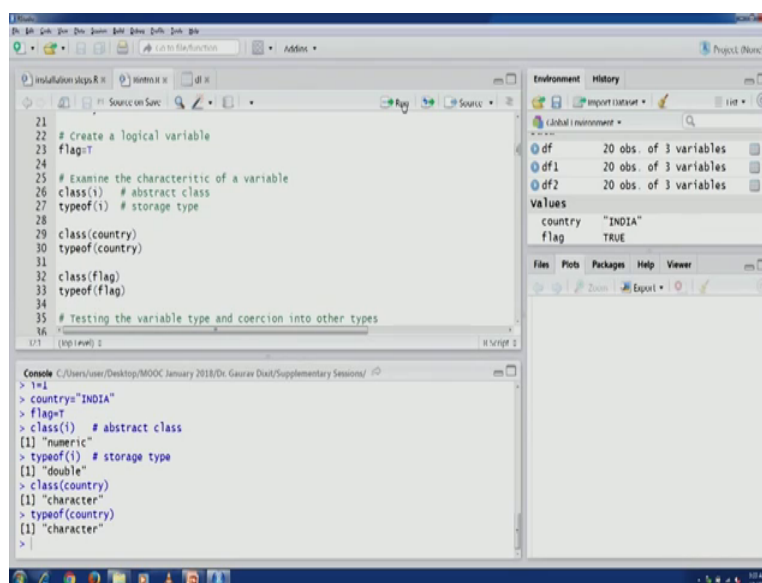
So, you would see in the data section and `i` variable has been created and it contains it has this value 1. We want to create a character variable, again you can type the name of your variable for example, in this case we have `country` and you can assign the value, in this case we have initialized this particular variable with this `India` as our country.

So, again execute in this code you would see in the data section `country` has been created and it has value `India`, similarly a logical variable can be created. So, for example, in this case we have this logical variable named as `flag` and the value is given as `true`, it could be `true` or `false` these are the 2 options for logical variables. So, once we execute this code, you would again see that `flag` is created in this data section and the value is `true`.

Now, how do we find out whether the characteristics of these variables? So, these 2 functions could be useful `class`, the `class` function can be used to find the abstract class of any particular variable and the `typeof` function can be used to find out a storage type of any particular variable for example, the variable `i` that we have just created, we can find out the abstract class of this particular variable which is numeric and then `typeof` of this particular variable which is double.

So, this particular variable is a numeric and it is being stored as a double in the memory, similarly we can check for `country`. So, you can see `country` is character the class of this variable `country` is character and a storage type is character as well. So, similarly we can check for `flag`.

(Refer Slide Time: 15:54)



So, class of this variable flag logical variable flag is again logical and this storage type is logical. Now, whether a particular variable is an integer or some other variable type; some other data type that also can be checked into that also can be examined in R and coercion from 1 variable type to another variable type can also be done.

For example, if you want to find out whether the variable we have, the whether the variable i is an integer or not. So, we can use this function is dot integer. So, this code if we exude this particular code we will get an answer as false because the i was created as a numeric variable and not as an integer variable.

So, therefore, that can be checked using the function is dot integer. Now, it is possible for us to quotes this particular variable; numeric variable into an integer variable, how that can be done? Let us create another variable j and we assign the value 1.5 to this particular variable you would see that in the data section j is created j can be seen here with value 1.5. Now, let us check whether this particular variable is integer or not because we have created it as a numeric variable it should come as false, so which is the case here.

Now, let us quotes this variable into an integer data type. So, that can be done using is dot integer command, is dot integer function. So, is dot integer function

we can pass on the same variable as argument and we can again store it store the return value in j itself and then we can display the value of j.

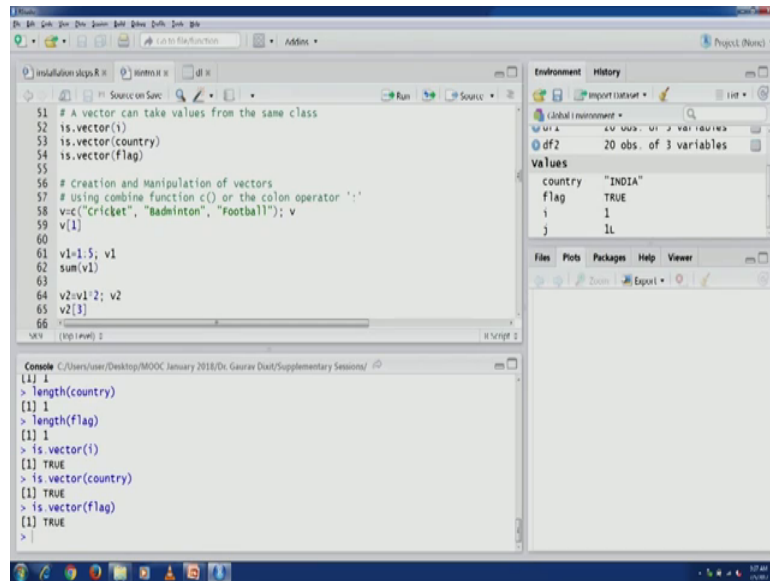
Let us execute this line, you will see 1. So, 1.5, value of 1.5 which was stored as j which was created as a numeric variable, has now been changed to 1 because the variable has been coerced into integer data type from numeric data type; now, if we again do a check whether this particular variable j is integer, now we will get an answer as true.

Now, another important function is length. So, length function can actually help us find about the length of a particular variable. So, for example, length of i is 1 length of country and flag these 3 variables length of all these variables 1. Now, next to discussion point is about vectors.

So, vectors are 1 of the basic building blocks for data in R, simple R variables that for example, I can be flag if we just created they are actually vectors, now vectors can take values from the same class. So, let us check whether the vector, whether the variables that we just created i country and flag, whether they are vector or not. So, again we can use the function like is dot vector, which is going to tell us whether these 3 variable belong R vector or not.

So, you would see all these, all 3 variables are vector. Now, creation and manipulation of vectors; this, so there is this function combined, which is combined function c or column operator would also be is used. So, c function and column operator can actually be used to create and manipulate vectors. For example, if we want to create a vector of character vector of these 3 values. Three values cricket, badminton and football.

(Refer Slide Time: 19:56)



```
51 # A vector can take values from the same class
52 is.vector(i)
53 is.vector(country)
54 is.vector(flag)
55
56 # Creation and Manipulation of vectors
57 # using combine function c() or the colon operator ':'
58 v=c("Cricket", "Badminton", "Football"); v
59 v[1]
60
61 v1=1:5; v1
62 sum(v1)
63
64 v2=v1^2; v2
65 v2[3]
66
```

Environment

Object	Class	Attributes
country	character	1
flag	logical	1
i	numeric	1
j	numeric	1

Console

```
[1] 1
> length(country)
[1] 1
> length(flag)
[1] 1
> is.vector(i)
[1] TRUE
> is.vector(country)
[1] TRUE
> is.vector(flag)
[1] TRUE
>
```

So, this vector v can be created we can use the c function and we can pass on these 3 strings as arguments, greater strings as argument, we will have this vector created as v vector has been created and it has 3 values cricket badminton and football. Now, we want to access individual values then we can do so using the brackets. So, v_1 , v_2 , v_3 will can be used to access first value and second value and third value respectively.

You can see v_1 cricket is v value in the output. Now, column operator can also be used to create a vector for example, v_1 1 to 5 so, this particular vector is going to be created with having values 1, 2, 3, 4 and 5, as you can see here. So, in column operator you mentioned the starting and ending values and the middle values are actually filled up by the column operator.

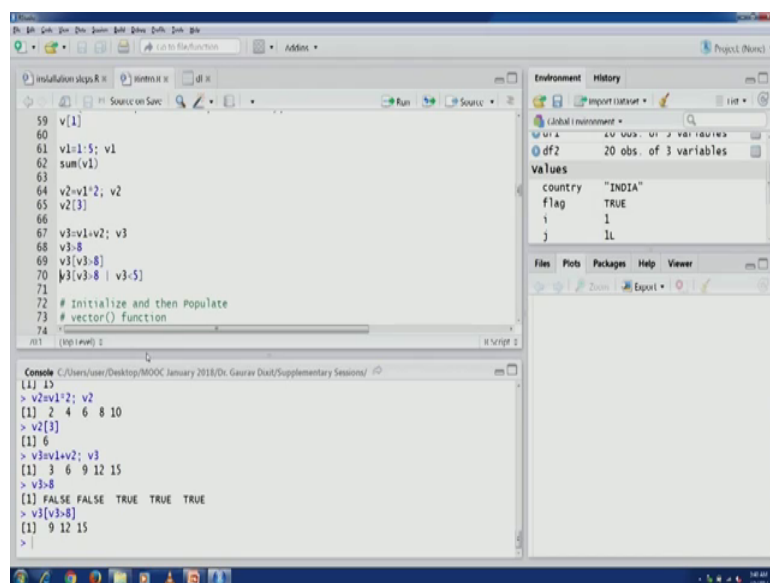
We can sum these values using the sum function, you can see sum v_1 and we get this particular output 15, similarly multiplication with a constant can be done, we want to access a particular value in this vector and that can also be done using brackets. So, we do 3, so we wanted to access the 3rd value. So, that we can do using v_2 3 in brackets and we can see output as 6.

If we want to add 2 vectors that can also be done v_1 and v_2 , but here it is important that both should be having the same number of values, similarly if we want to find out the values in a particular vector which are greater than 8 for

example, in this case there are many values which are greater than 8. So, we want to just find out those values.

So, we can execute this particular line `v3` greater than `8` and we will see the answer is false for first 2 values because they are less than 8 and the true for rest of the 3 values which are actually greater than 8. Similarly, if we want to access the values which are greater than 8 we can do in this form also `v3` and within brackets we can again use `v3` greater than 8.

(Refer Slide Time: 22:55)



The screenshot shows an RStudio window with the following components:

- Source Editor:** Contains R code lines 59 to 74. Line 70 is highlighted: `v3[v3>8 | v3<5]`.
- Environment:** Shows variables `v1` (numeric vector), `v2` (numeric vector), and `df2` (data frame with 20 observations and 3 variables).
- Console:** Displays the output of the executed code:

```
[1] 13
> v2=v1^2; v2
[1] 2 4 6 8 10
> v2[3]
[1] 6
> v3=v1+v2; v3
[1] 3 6 9 12 15
> v3>8
[1] FALSE FALSE TRUE TRUE TRUE
> v3[v3>8]
[1] 9 12 15
```

So, it will return the index for all the values which are greater than 8 and then those values would actually be accessed using the brackets. Similarly, if we want to access values which are greater than 8, but less than 5 again a similar thing can be done, you can see the value is 3, 9, 12 and 15 there which are less than 5 or greater than 8.

Now, sometimes we might be required to initialize a vector and then populate it, till now what we have been doing is at the same line at the same instant we were creating variables and vectors and immediately we were initializing them as well, we were populating them as well. So, sometimes we might want to do this process in 2 steps first initialize and then populate.

So, for that we can use a vector function, so vector function can actually be used to initialize vector, by a number in this case a vector function is being used to create a vector of length 4. Now, if you want to, so by default if you use the vector function by default a logical variable, the logical vector is created, if you want to create a numeric vector then we can do so, we have to mention the mode of mode as numeric in the vector function and that can be done.

(Refer Slide Time: 24:39)

The screenshot shows an R Studio interface with the following content:

```

68 v3<-8
69 v3[v3>8]
70 v3[v3>8 | v3<5]
71
72 # Initialize and then Populate
73 # vector() function
74 v4=vector(length = 4); v4
75
76 v5=vector(mode = "numeric", length = 4); v5
77 v5[3]=1.4; v5
78
79 v6=vector(mode = "integer", length = 0); v6
80 length(v6)
81
82 # vectors seem to be one dimensional arrays, but they are dimensionless.
83
84
85

```

The console output shows the execution of these commands:

```

[1] FALSE FALSE TRUE TRUE
> v3[v3>8]
[1] 9 12 15
> v3[v3>8 | v3<5]
[1] 3 9 12 15
> v4=vector(length = 4); v4
[1] FALSE FALSE FALSE FALSE
> v5=vector(mode = "numeric", length = 4); v5
[1] 0 0 0 0
> v5[3]=1.4; v5
[1] 0.0 0.0 1.4 0.0
>

```

The Environment pane on the right shows the following objects:

Object	Class	Attributes
Il	chr	[1:3] "cricket" "badmi.."
v	chr	[1:3] 1 2 3 4 5
v1	int	[1:5] 1 2 3 4 5
v2	num	[1:5] 2 4 6 8 10
v3	num	[1:5] 3 6 9 12 15
v4	logi	[1:4] FALSE FALSE FALSE
v5	num	[1:4] 0 0 1.4 0

If you want to reassigns 1 of the values for example, 3rd the value, that can also be done you can see 1.4 they are just which varies assigned. Similarly, if we want to create an integer vector that can also be done, we have to mention and the mode argument it as mode argument has to be assigned as a integer, you can see the this particular vector v6 we had created with length 0. So, we will check for it is length again you get you get the right answer as 0, here length function can actually be used.

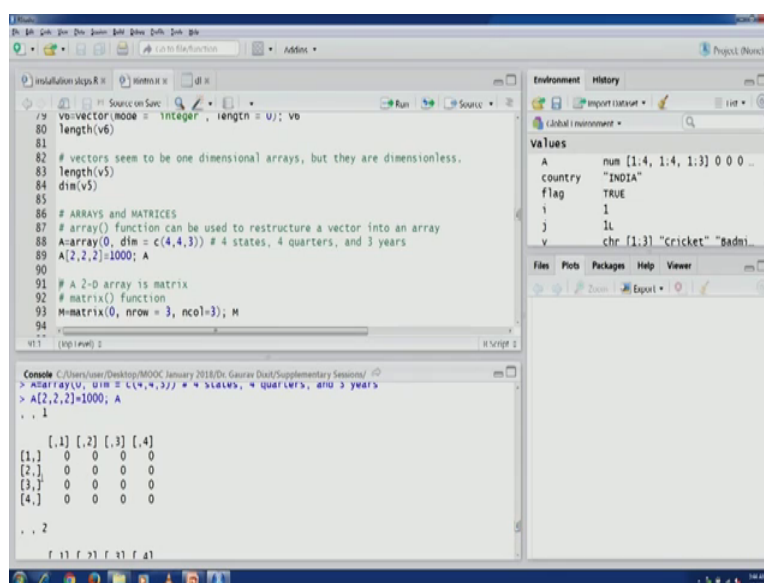
Now, whatever we have done so far it might look like that vectors are 1 dimensional arrays, right till now whatever examples that we have gone through they give this impression, but if you really look in R whether they are 1 dimensional arrays or not, but in R they are they are actually defined as dimensionless, that can be checked using these 2 function if we check for the length of this particular vector v5 we will get answer as 4, but if we check for

the dimension using the dim function dim function we would find as null, so which is undefined.

So, vectors are not actually 1 dimensional edits they are actually dimensionless. That brings us to the next discussion point that is arrays and matrices. So, R also has these building blocks arrays and matrices. Now, we have array function which can actually be used to restructure a vector into an array, so how that can be done we can see this through this example.

Now, array function the initialization 0 is given and we are creating an array of these dimensions 4 states, 4 quarters and 3 years. So, this particular array would be created with these dimensions; with these 3 dimensions I am having. So, this we can see here, now we I assigned one of the value as 1000. So, that can also we can check here.

(Refer Slide Time: 27:04)



The screenshot shows an RStudio interface with a script editor, environment pane, and console. The script editor contains the following R code:

```
## Initialization steps R
# Vectors
v6 = vector(mode = "integer", length = 10); v6
length(v6)
# vectors seem to be one dimensional arrays, but they are dimensionless.
length(v5)
dim(v5)
# ARRAYS and MATRICES
# array() function can be used to restructure a vector into an array
A = array(0, dim = c(4,4,3)) # 4 states, 4 quarters, and 3 years
A[2,2,2] = 1000; A
# A 2-D array is matrix
# matrix() function
M = matrix(0, nrow = 3, ncol = 3); M
```

The console shows the output of the executed code:

```
> A=array(0, dim = c(4,4,3)) # 4 states, 4 quarters, and 3 years
> A[2,2,2]=1000; A
, , 1
[1,] 0 0 0 0
[2,] 0 0 0 0
[3,] 0 0 0 0
[4,] 0 0 0 0
, , 2
[1,] 0 0 0 0
[2,] 0 0 0 0
[3,] 0 0 0 0
[4,] 0 0 0 0
```

The environment pane on the right shows the objects created:

Object	Value
A	num [1,4, 1,4, 1,3] 0 0 0 ..
country	"INDIA"
flag	TRUE
i	1
j	1L
v	chr [1,3] "cricket" "badmi"

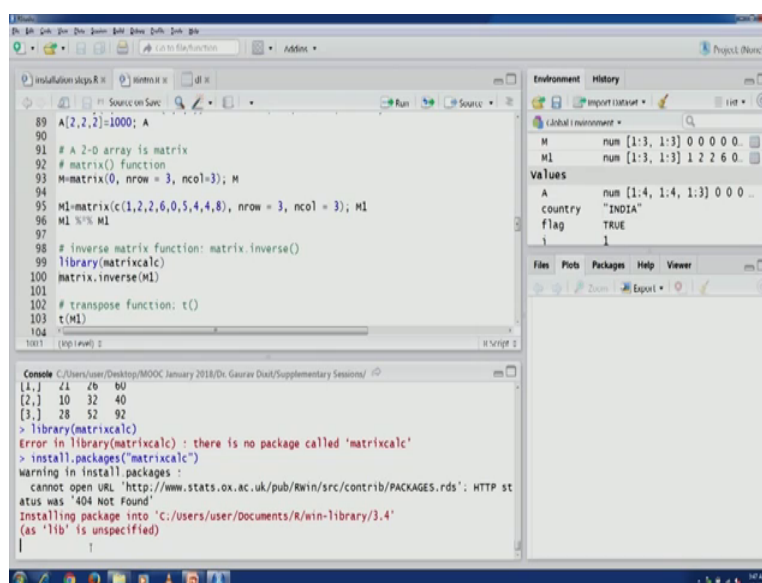
You would see 3 dimensional array; so this is our 1st year, this particular matrix looking structure in this array is for the first year, then this is for the second year, then for third year. So, we had created this array for 3 years, so we can see 3 matrix structures back to back matrix lecture have been created. Now, that brings us to the matrix so a 2 D array is a matrix. So, array can be thought of as a series of matrix and a 2 D array being a matrix. So, we can use matrix function to initialize an array, you can see the matrix initialization with the value of 0 for all

the elements and you can define a number of row as 3 and number of column as 3 in this case so and row and column argument can be used to define.

Now, with a different initialization we can create this program matrix M1 you can see here, now a matrix multiplication can be used using this operator percentage as trick and then followed by percentage this. So, this particular operator can be used for matrix multiplication in R. So, this is the result of matrix multiplication M1 multiplied by with M1 itself.

If we want to find out the inverse matrix though so we have the matrix do not inverse function for that, but to be able to access this function, to be able to use this function we first need to load this particular library matrix calc, so we will just do that. So, we can just load this particular library matrix calc, once it is loaded then we can access this function matrix or inverse and for any matrix we will get the inverse of it.

(Refer Slide Time: 29:25)



The screenshot shows an RStudio window with a script editor, environment pane, and console. The script editor contains the following R code:

```
89 A[2,2]=1000; A
90
91 # A 2-D array is matrix
92 # matrix() function
93 M=matrix(0, nrow = 3, ncol=3); M
94
95 M1=matrix(c(1,2,2,6,0,5,4,4,8), nrow = 3, ncol = 3); M1
96 M1 %*% M1
97
98 # inverse matrix function: matrix.inverse()
99 library(matrixcalc)
100 matrix.inverse(M1)
101
102 # transpose function: t()
103 t(M1)
104
```

The environment pane on the right shows the following objects:

Object	Class	Value
M	matrix	[1,3, 1,3] 0 0 0 0.
M1	matrix	[1,3, 1,3] 1 2 2 6 0.
A	matrix	[1,4, 1,4, 1,3] 0 0 0.
country	character	"INDIA"
flag	logical	TRUE
1	numeric	1

The console shows the output of the first three lines of code:

```
[1,] 21 26 60
[2,] 10 32 40
[3,] 28 52 92
```

Below the output, an error message is displayed:

```
Error in library(matrixcalc) : there is no package called 'matrixcalc'
> install.packages("matrixcalc")
Warning in install.packages :
cannot open URL 'http://www.stats.ox.ac.uk/pub/Rwin/src/contrib/PACKAGES.rds': HTTP st
atus was '404 Not Found'
Installing package into 'C:/Users/user/Documents/R/win-library/3.4'
(as 'lib' is unspecified)
```

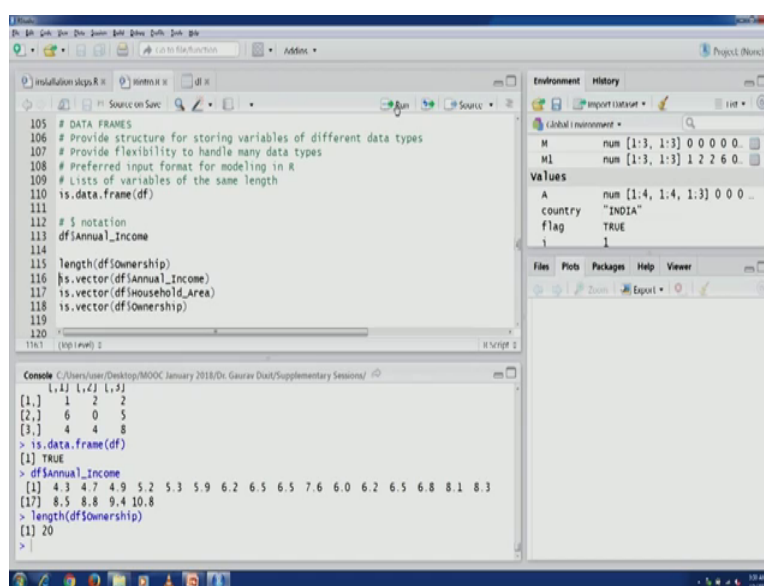
You can see the inverse matrix here, if we want to transpose a matrix then there is this function t transpose function that can actually be used to get a transpose of a matrix. Now, the next discussion point is about data frames, now data frames provide a structure for storing a variables of different data types. So, till now we whatever we discuss vectors, arrays and matrices we were using the

same data type for a storing values of same data type. Data frames can be used for storing variables of different data types.

Now, they provide flexibility to handle many data types preferred input, now they have also because of that they have also become the preferred input format for modeling in R, they can also be understood as a list of variables of the same length for example, the data set that we earlier imported is this sedan car data set we can see 3 variables of same length annual income, household area and ownership and you can see 2 of the variables annual income and variable household area are of the same type, they are numeric, but the ownership is different.

So, we can check whether a particular variable is data frame or not, using this program; using this particular function is dot data dot frame. So, we will just check this for df which is true because the in the start of this particular script the files that we imported they were actually in the data frame; they were actually stored as data frame. Now, another important aspect related to data frame is the dollar notation. So, using dollar notation you can access any of the variables which are there in stored in a data frame for example, annual income. We can access using this particular line df\$dollar annual income, see you will get the values that are stored in this particular variable.

(Refer Slide Time: 32:10)



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

```
105 # DATA FRAMES
106 # Provide structure for storing variables of different data types
107 # Provide flexibility to handle many data types
108 # Preferred input format for modeling in R
109 # Lists of variables of the same length
110 is.data.frame(df)
111
112 # $ notation
113 df$Annual_Income
114
115 length(df$ownership)
116 is.vector(df$Annual_Income)
117 is.vector(df$household_Area)
118 is.vector(df$ownership)
119
120
```

The Environment pane on the right shows the following objects:

Object	Class	Attributes
M	num	[1:3, 1:3] 0 0 0 0
M1	num	[1:3, 1:3] 1 2 2 6 0
A	num	[1:4, 1:4, 1:3] 0 0 0
country	chr	"INDIA"
flag	lgl	TRUE
i	int	1

The Console pane at the bottom shows the output of the commands:

```
[1,] 1 2 2
[2,] 6 0 5
[3,] 4 4 8
> is.data.frame(df)
[1] TRUE
> df$Annual_Income
[1] 4.3 4.7 4.9 5.2 5.3 5.9 6.2 6.5 6.5 7.6 6.0 6.2 6.5 6.8 8.1 8.3
[17] 8.5 8.8 9.4 10.8
> length(df$ownership)
[1] 20
>
```

We can also check the length of individual variables stored in a data frame using the same notation and passing it as argument to the length function, you can see the answer is 20. Similarly, we can also check, so all these variables as we discussed before the variables are actually vectors in R. So, we can check for the same is dot vector for annual income and household area and ownership and we will see all these variables are actually vectors.

So, data frames can also be considered as having vectors of same length. Now, if we look at 1 particular variable ownership it is actually a categorical variable which is actually called a factor in R. So, if we want to check whether this particular variable is a factor or categorical variable we can do so by using this particular function is dot factor. So, you will see the answer true.

So, ownership is actually stored as a; this is actually a categorical variable and it is stored as a factor in R. Now, another function is str, this function is to display the structure of a data frame or a list will later see the data frames are also a form of list. So, let us execute this line you see that in the data frame you have 20 observations 3 variables and you get some details about the variables or vectors that are there, annual income numeric, num household area and numeric and ownership if you would see factor with 2 levels non owner and owner.

(Refer Slide Time: 34:01)

```

116 is.vector(df$Annual_Income)
117 is.vector(df$Household_Area)
118 is.vector(df$Ownership)
119
120 # factor: is a categorical variable
121 is.factor(df$Ownership)
122
123 # str() function
124 # structure of a data frame
125 str(df)
126
127 # subsetting operator: []
128 df[,3] # 3rd column
129 df[1:5,] # first 5 rows
130 df[,c(1,3)] # 1st and 3rd columns
131

```

Console Output:

```

> is.vector(df$Household_Area)
[1] TRUE
> is.vector(df$Ownership)
[1] FALSE
> is.factor(df$Ownership)
[1] TRUE
> str(df)
'data.frame': 20 obs. of 3 variables:
 $ Annual_Income : num  4.3 4.7 4.9 5.2 5.3 5.9 6.2 6.5 6.5 7.6 ...
 $ Household_Area : num  20 16 18 15 21 15 14 17 18 19 ...
 $ Ownership      : Factor w/ 2 levels "non-owner","owner": 1 1 1 1 1 1 1 1 1 1 ...

```

Environment History:

Object	Class	Attributes
M	num	[1:3, 1:3] 0 0 0 0.
M1	num	[1:3, 1:3] 1 2 2 6 0.
A	num	[1:4, 1:4, 1:3] 0 0 0 .
country	chr	"INDIA"
flag	lgl	TRUE
i	int	1

So, this kind of a structure of these variables can actually be displayed using `str` and the `structure` command. Now, next important operator is sub setting operator, so actually the brackets can also be used to subset a data frame for example, if you want to access just the 3rd column of a data frame, so instead of using the dollar notation we can actually use this particular subset operator that is actually brackets. So, first 1 is for rows and the second is for columns. So, as I have mentioned 3 year, so we will be accessing 3rd column.

(Refer Slide Time: 34:47)

The screenshot shows an RStudio interface with the following components:

- Source Editor:** Contains R code for creating a data frame, subsetting it, and checking its structure.


```

122 # str() function
123 # structure of a data frame
124 str(df)
125
126 # Subsetting operator: []
127 df[,3] # 3rd column
128 df[1:5,] # first 5 rows
129 df[,c(1,3)] # 1st and 3rd columns
130 df[,c("Annual_Income", "Ownership")]
131 df[df$Annual_Income > 8,] # retrieve records with annual income more than 8 lpa
132
133 class(df)
134 typeof(df)
135
136
137
138
      
```
- Environment:** Shows the objects in the global environment.

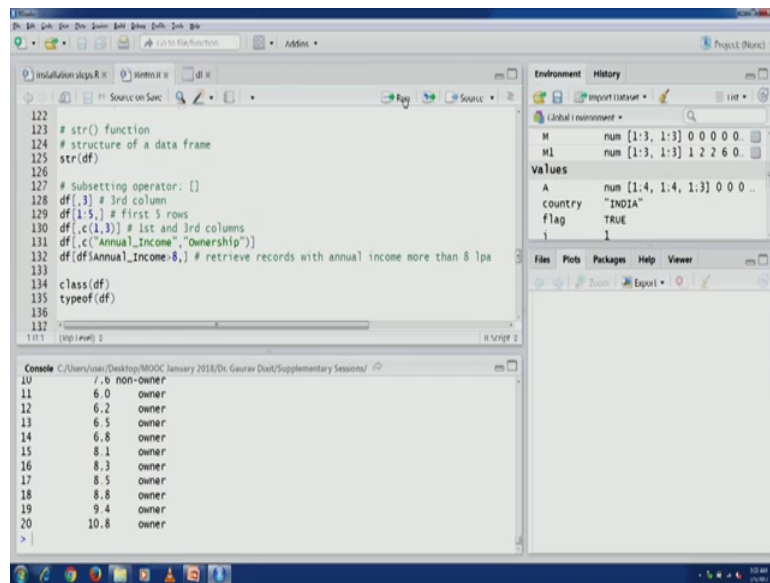
Object	Class	Attributes
M	num	[1:3, 1:3] 0 0 0 0 0
M1	num	[1:3, 1:3] 1 2 2 6 0
A	num	[1:4, 1:4, 1:3] 0 0 0
country	character	"INDIA"
flag	logical	TRUE
1	integer	1
- Console:** Shows the output of the R commands.


```

[1] TRUE
> str(df)
'data.frame': 20 obs. of 3 variables:
 $ Annual_Income : num 4.3 4.7 4.9 5.2 5.3 5.9 6.2 6.5 6.5 7.6 ...
 $ Household_Area : num 20 16 18 15 21 15 14 17 18 19 ...
 $ Ownership : Factor w/ 2 levels "non-owner", "owner": 1 1 1 1 1 1 1 1 1 1 ...
> df[,3] # 3rd column
[1] non-owner non-owner non-owner non-owner non-owner non-owner non-owner non-owner
[9] non-owner non-owner owner owner owner owner owner owner owner
[17] owner owner owner owner owner owner owner owner
Levels: non-owner owner
      
```

You can see 3rd column the ownership related values are there. Now, if you want to access this first 5 rows, you can mention that the same here 1 to 5 using the column operator and for the column the space that nothing is mentioned there, so all the columns would be displayed, would see. Another, if you want to access 2 columns in 1 go we can do so by using combined function in the brackets, if you want to; if we do not remember the index whether it was a first column or third column we can actually use the combined function mention and mention the actual name of those variables or vectors and then again we can access the same data you can see here same data has been accessed.

(Refer Slide Time: 35:37)



The screenshot shows an RStudio interface. The main editor window contains R code for creating and manipulating a data frame. The Environment pane on the right shows the objects 'M' and 'M1' as numeric vectors and 'A' as a data frame. The Console pane at the bottom shows the output of the code, displaying a list of cars with their annual income and ownership status.

```
122 # str() function
123 # structure of a data frame
124 str(df)
125
126
127 # Subsetting operator: []
128 df[,3] # 3rd column
129 df[1:5,] # first 5 rows
130 df[,c(1,3)] # 1st and 3rd columns
131 df[,c("Annual_Income", "Ownership")]
132 df[df$Annual_Income>8,] # retrieve records with annual income more than 8 lpa
133
134 class(df)
135 typeof(df)
136
137
```

Environment:

Object	Class	Attributes
M	num	[1:3, 1:3] 0 0 0 0
M1	num	[1:3, 1:3] 1 2 2 6 0
A	data.frame	country, flag, values

Console:

```
10 / 0 non-owner
11 6.0 owner
12 6.2 owner
13 6.5 owner
14 6.8 owner
15 8.1 owner
16 8.3 owner
17 8.5 owner
18 8.8 owner
19 9.4 owner
20 10.8 owner
```

Now, if you want to create; if you want to display retrieve few records using these you know these greater than using these operators for example, annual income having greater than 8 lpa, so that can be retrieved using this particular command. So, all the rows where the annual income is greater than 8 all such columns would be displayed, you can see here all the values are greater than 8, 8.1, 8.2 up to 10.8.

Now, we want to check for the class of df or type of df, that can also be done. You can see class of df is mention as data frame and type is a list. So, data frames are basically a list, are actually list. So, what are lists? So, list are a collection of objects of various types including other lists. So, list function can be used to again create and a great list, double bracket notation also will come across the double bracket notation, we will see what is that.

So, a list can be generated for example, this lists. So, different objects can actually be used as obviously, we have created these objects. So, i, j, v, m and this is another addition that we are using for this list initialization and creation. So, this using a list function we can create lists. So, you can see here, list has been created. So, list can store objects of various type and they can be of different length.

So, 1 big difference is the objects and list they can be of different types various types and they can be of different length as well, while in data frame the variables I tribes could be different, but they have to be of same length. For example, you would not see angle the see double bracket notation and see first you know a list element is this Roorkee which being slide and second list element was actually i and values 1 the 3rd list element was actually j value was again 1 and the 4th element which actually this particular character vector cricket, badminton and football and then 5th element is actually this matrix.

(Refer Slide Time: 38:25)

```

132 df[df$Annual_Income>8,] # retrieve records with annual income more than 8 lpa
133
134 class(df)
135 typeof(df)
136
137 # LISTS
138 # A collection of objects of various types including other lists
139 # list() function
140 # double bracket notation [[]]
141 L=list("Roorkee",i,j,v,M); L
142
143 class(L[4])
144 length(L[4])
145
146 class(L[[4]])
147

```

Console Output:

```

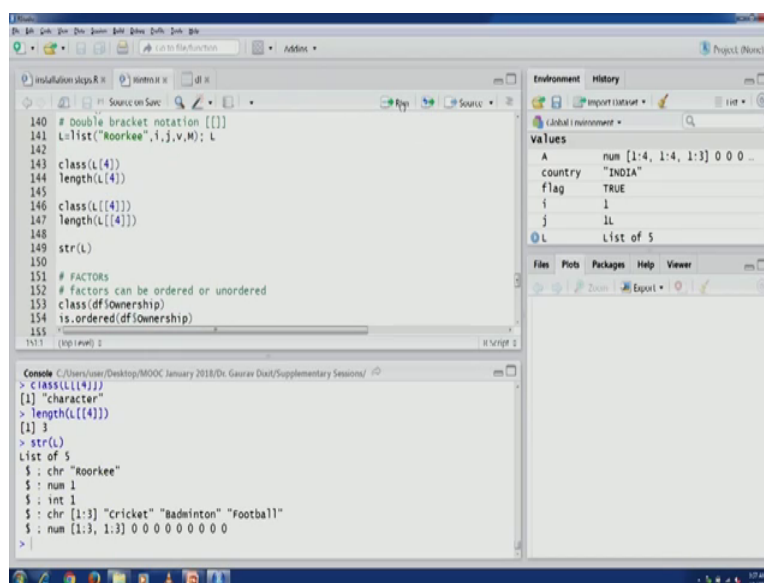
[[1]] 1
[[4]]
[1] "Cricket" "Badminton" "Football"

[[5]]
[,1] [,2] [,3]
[1,] 0 0 0
[2,] 0 0 0
[3,] 0 0 0

```

So, if we check for class and length of a particular element of a list we will see this we will get this. So, we want to actually check for a particular vector, then we will have to use double brackets now you would see that 1 and for within double brackets you will see a character vector, which was actually just we saw and the output this one. So, this is the character vector and length of that character vector is also displayed as 3. So, a structure command can also be used with lists as well as data frame, which are also lists.

(Refer Slide Time: 39:14)



The screenshot shows the R Studio environment. The script editor on the left contains the following R code:

```
140 # Double bracket notation [[]]
141 L=list("Roorkee",1,j,v,M): L
142
143 class(L[4])
144 length(L[4])
145
146 class(L[[4]])
147 length(L[[4]])
148
149 str(L)
150
151 # FACTORS
152 # factors can be ordered or unordered
153 class(df$ownership)
154 is.ordered(df$ownership)
155
```

The Environment pane on the right shows the variable 'L' as a 'List of 5'.

The Console pane at the bottom shows the output of the commands:

```
> class(L[[4]])
[1] "character"
> length(L[[4]])
[1] 3
> str(L)
list of 5
 $ : chr "Roorkee"
 $ : num 1
 $ : int 1
 $ : chr [1:3] "cricket" "badminton" "Football"
 $ : num [1:3, 1:3] 0 0 0 0 0 0 0 0
```

So, you can see 5 elements of list and the a structure of all those elements is displayed for example, Roorkee this was character, then you have numeric vector and integer vector and then the and the character vector followed by a numeric. Now, our next discussion point is factors, so factors are as we discussed before they are categorical variable, they are called factors in R. So, they can be ordered or unordered as we discussed in before in previous lecture that factors categorical variable could either be nominal or ordinal. So, here in this case categorical variables are called factors and ordered or unordered they are called ordered variable can are generally called ordered and nominal variable are generally called unordered.

So, we have ownership variable already in our data frame, let us check whether; let us check the type of class of that particular variable and find out whether it is a ordered or unordered variable. So, as you can see the class of this particular variable is factor which is categorical and then let us check whether it is order or not. So, you will get answer as false because there was no specified order there.

Now, there is another this function head which can be used to actually display first 6 values and if it is a factor variable then levels would also be displayed. So, let us run this and you would see that levels non over an owner for a factor variables are displayed. So, all the first 6 values in case of a integer or numeric

variable we would have just seen the values, in case of a factor variable R knows that it is a factor variable, so in the output they also display the levels that are used in the and this particular variable; factor variable.

(Refer Slide Time: 41:21)

```

153 class(df$ownership)
154 is.ordered(df$ownership)
155
156 head(df$ownership) # display first 6 values and levels
157
158 income_group=vector(mode = "character", length = length(df$Annual_Income))
159 income_group[df$Annual_Income<=6]="lower middle"
160 income_group[df$Annual_Income>6 & df$Annual_Income<=9]="middle"
161 income_group[df$Annual_Income>9]="upper middle"
162 income_group
163
164 # factor() function
165 income_class=factor(income_group, levels = c("lower middle","middle",
166                                             "upper middle"), ordered = T)
167
168 ***
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

> class(df$ownership)
[1] "factor"
> is.ordered(df$ownership)
[1] FALSE
> head(df$ownership) # display first 6 values and levels
[1] non-owner non-owner non-owner non-owner non-owner non-owner
Levels: non-owner owner

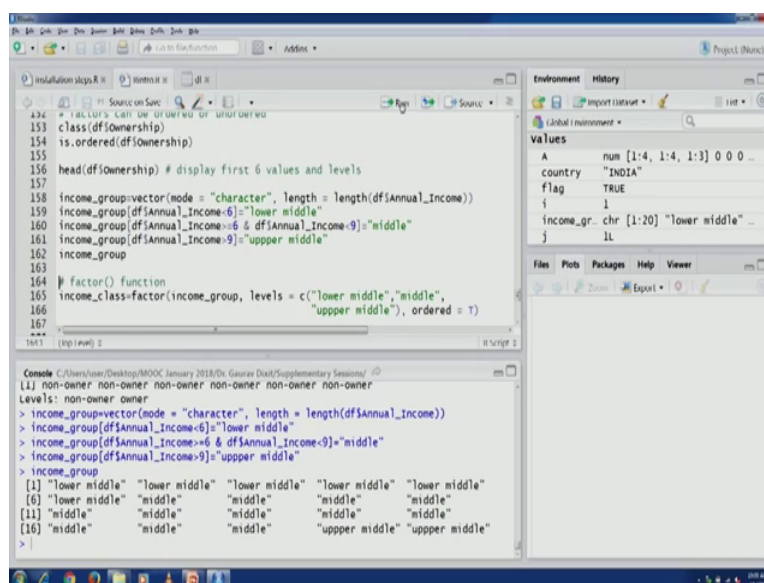
```

Now, how can we go about; how can we actually go about creating factor variables for example, we have 1 particular variable in our data set that is annual income if we want to create groups off those and those households and we want to create a factor categorical variable, where it says some houses would belonging to lower middle class, some would be belonging to the middle class and some would be belonging to the upper middle, how that can be done?

So, let us first create a vector of called income groups and mode is going to be the character because we are going to be storing these strings, we did last lower middle class and upper middle class and the length would be same as the annual income. So, let us create this vector and then any record which is having annual income less than 6 can we called lower middle class.

So, let us assign this value, any household which is having income greater than or equal to 6 and less than 9 can be called middle class so let us do that and then any household having income greater than 9 can be called upper middle class, now let us check the values you would see all the values have been assigned with lower middle class or middle class or the upper middle class.

(Refer Slide Time: 42:40)



```
## Factors can be ordered or unordered
153 class(df$ownership)
154 is.ordered(df$ownership)
155
156 head(df$ownership) # display first 6 values and levels
157
158 income_group=vector(mode = "character", length = length(df$Annual_Income))
159 income_group[df$Annual_Income<6]="lower middle"
160 income_group[df$Annual_Income>=6 & df$Annual_Income<9]="middle"
161 income_group[df$Annual_Income>=9]="upper middle"
162 income_group
163
164 # factor() function
165 income_class=Factor(income_group, levels = c("lower middle","middle",
166 "upper middle"), ordered = T)
167
168
```

Console:

```
[1] non-owner non-owner non-owner non-owner non-owner non-owner
Levels: non-owner owner
> income_group=vector(mode = "character", length = length(df$Annual_Income))
> income_group[df$Annual_Income<6]="lower middle"
> income_group[df$Annual_Income>=6 & df$Annual_Income<9]="middle"
> income_group[df$Annual_Income>=9]="upper middle"
> income_group
[1] "lower middle" "lower middle" "lower middle" "lower middle" "lower middle"
[6] "middle" "middle" "middle" "middle" "middle"
[11] "middle" "middle" "middle" "middle" "middle"
[16] "middle" "middle" "middle" "upper middle" "upper middle"
>
```

Environment:

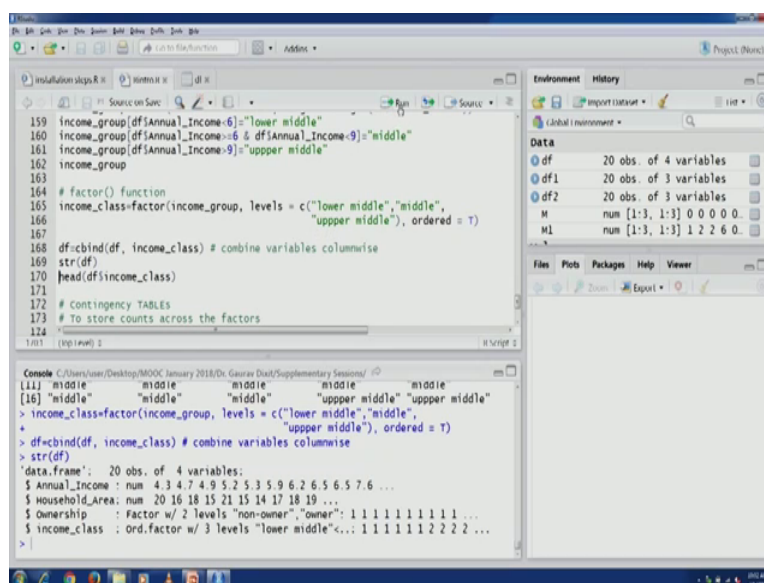
Object	Class	Attributes
A	num	[1:4, 1:4, 1:3] 0 0 0 ...
country	chr	"INDIA"
flag	lgl	TRUE
i	chr	1
income_group	chr	[1:20] "lower middle" ...
j	chr	1L

Now, how do we create a factor of it? Factor variable from this now let us create another variable income class which is now going to be a factor variable, now we are using the income group the character vector that we had just created and levels so because the factor variable are going to have levels. So, therefore, we need to assign some levels.

So, in this case we already know lower middle, middle and upper middle and you can see here ordered is true. So, we are ordering these variables, so lower middle, then middle and then upper middle, so this is the order. So, we are giving the names of the levels and also we have paired this ordered or women dash true, now this particular factor ordered factor would actually be created.

Now, we can combine this variable in our data frame df. So, c bind command can actually be used to combine variables column bias, so we can use this c bind command. So, income class, but now we actually be combined in the data frame, we can check the same using the structure command, str command you can see income glass ordered factor with 3 levels.

(Refer Slide Time: 43:56)



The screenshot shows an RStudio interface. The main editor window contains R code for creating an ordered factor variable 'income_class' based on 'income_group'. The code includes comments and uses the 'factor()' function with 'levels' and 'ordered = T'. The console window at the bottom shows the execution of the code, displaying the structure of the resulting data frame 'df' and the 'income_class' variable. The Environment pane on the right shows the objects 'df', 'df1', 'df2', 'M', and 'M1' in the global environment.

```
159 income_group[df$Annual_Income<6]="lower middle"
160 income_group[df$Annual_Income<6 & df$Annual_Income<9]="middle"
161 income_group[df$Annual_Income<9]="upper middle"
162 income_group
163
164 # factor() function
165 income_class=factor(income_group, levels = c("lower middle","middle",
166                                             "upper middle"), ordered = T)
167
168 df=cbind(df, income_class) # combine variables columnwise
169 str(df)
170 head(df$income_class)
171
172 # Contingency TABLES
173 # To store counts across the factors
174
```

Console output:

```
[[1]] middle middle middle middle middle
[16] "middle" "middle" "middle" "upper middle" "upper middle"
> income_class=factor(income_group, levels = c("lower middle","middle",
+                                             "upper middle"), ordered = T)
> df=cbind(df, income_class) # combine variables columnwise
> str(df)
'data.frame': 20 obs. of 4 variables:
 $ Annual_Income: num 4.3 4.7 4.9 5.2 5.3 5.9 6.2 6.5 6.5 7.6 ...
 $ Household_Area: num 20 16 18 15 21 15 14 17 18 19 ...
 $ Ownership : Factor w/ 2 levels "non-owner","owner": 1 1 1 1 1 1 1 1 1 1 ...
 $ income_class : Ord.factor w/ 3 levels "lower middle"<... 1 1 1 1 1 2 2 2 ...
```

Environment pane:

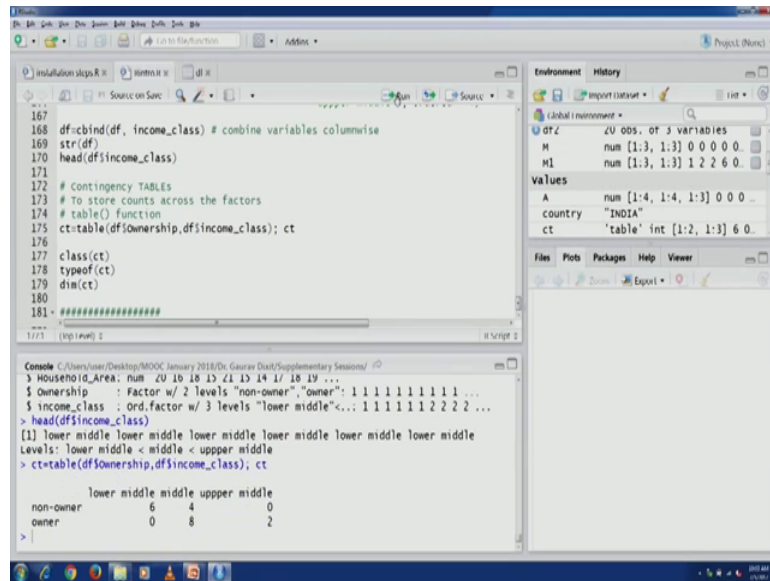
Object	Class	Attributes
df	data.frame	20 obs. of 4 variables
df1	data.frame	20 obs. of 3 variables
df2	data.frame	20 obs. of 3 variables
M	matrix	num [1:3, 1:3] 0 0 0 0 0
M1	matrix	num [1:3, 1:3] 1 2 2 6 0

We can again run the head command to find out a clear visibility of what are these levels and their order you can see now the values are mentioned and levels in the level you can see lower middle is less than middle and which is again less than upper middle, so you can see an order in these variables. So, this particular factory you has been created as a ordered variable.

Now, the another discussion point is on a contingency table. So, in our in this particular course we would be for especially for classification tasks, we would be using contingency tables to understand the results of a particular classification related technique or algorithm. So, first in we need to understand these tables, so table is the command which can actually be used to create. So, it is generally used contingency table generally is used to store counts across the factors. So, let us see through an example, so we have df, we have ownership variable and income class that we have just created.

Now, let us see how many owner and what are their classes? What are their numbers across different classes and for how many non owners? What are their numbers across different classes? So, let us create a contingency table using table command. So, we just need to mention these 2 factor variables and it will be done.

(Refer Slide Time: 45:22)



The screenshot shows the R Studio environment. The script editor on the left contains the following R code:

```
167 df$cbind(df, income_class) # combine variables columnwise
168 str(df)
169 head(df$income_class)
170
171 # Contingency TABLES
172 # To store counts across the factors
173 # table() function
174 ct=table(df$ownership,df$income_class); ct
175
176 class(ct)
177 typeof(ct)
178 dim(ct)
179
180 #####
181
```

The console on the bottom left shows the output of the code:

```
> Household_Area: num 20 16 18 13 11 13 14 17 18 19 ...
$ ownership : Factor w/ 2 levels "non-owner", "owner": 1 1 1 1 1 1 1 1 1 ...
$ income_class : Ord.factor w/ 3 levels "lower middle"<...: 1 1 1 1 1 2 2 2 ...
> head(df$income_class)
[1] lower middle lower middle lower middle lower middle lower middle lower middle
Levels: lower middle < middle < upper middle
> ct=table(df$ownership,df$income_class); ct
      lower middle middle upper middle
non-owner      6      4      0
owner           0      8      2
```

The Environment pane on the right shows the objects created in the R session:

- df: ZU obs. of 3 variables
- M: num [1:3, 1:3] 0 0 0 0 0
- M1: num [1:3, 1:3] 1 2 2 6 0
- Values: num [1:4, 1:4, 1:3] 0 0 0
- country: "INDIA"
- ct: 'table' int [1:2, 1:3] 6 0

You would see that first row is about non owner, second row is about owner and you will see 3 columns as lower middle, middle and upper middle; you would see a non owner you know 6 lower middle class non owners are there and then 4 middle class owners are there and the 0 upper middle class non owners are there.

Now, if we look at the owner row you would see that 0 lower middle class owners are there, 8 middle class owners are there and 2 upper middle class owners are there. So, this kind of count we can always get using table command. So, this particular command is going to be useful for us when we do our classification task. Now, we want to find out the class and type off and dimension of this particular table. So, we can execute these lines you can see table and integer and the dimensions are also mentioned 2 and 3. So, this ends the introduction of R. So, in the next class, next supplementary lecture we are going to cover the basic statistical technique.

Thank you.