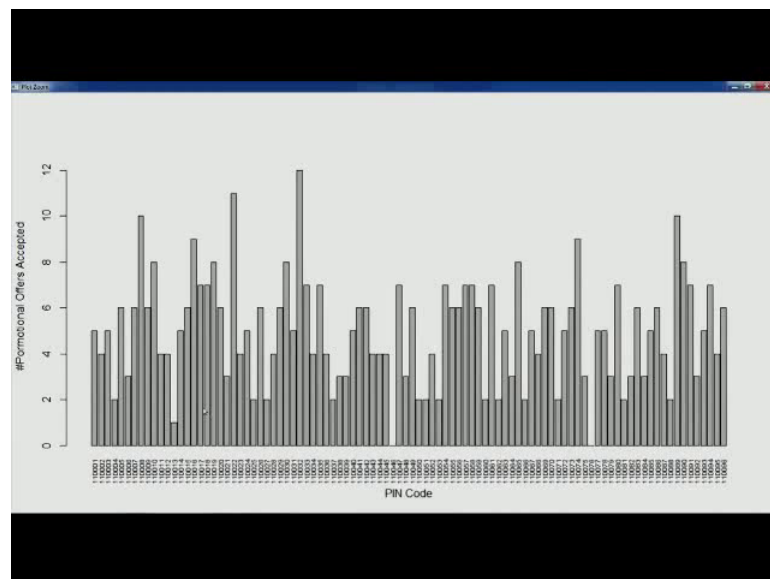


Business Analytics & Data Mining Modeling Using R
Dr. Gaurav Dixit
Department of Management Studies
Indian Institute of Technology, Roorkee

Lecture - 40
Classification and Regression Trees- Part V

Welcome to the course business analytics and data mining modelling using R. So, in the previous lecture, we were discussing classification and regression trees; and specifically we were talking about the data set on the promotional offers. We were looking at the variables of those data sets, and we identified the pin code particular variable which is going to be categorical variable, this is a categorical variable and then too many categories 96 of them which we saw in the previous lecture. And we wanted to find out phase where we can actually reduce the number of categories to a fewer number, so that the dimensionality problem is solved there.

(Refer Slide Time: 01:08)

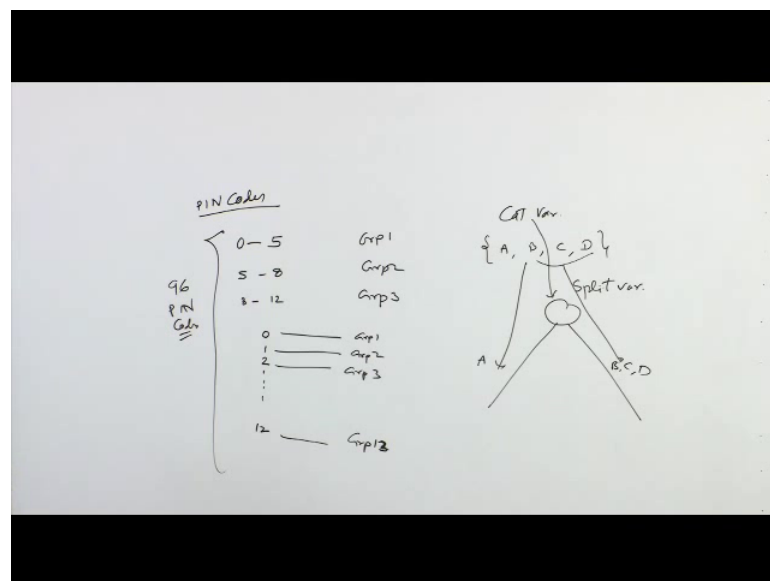


So, we generated a bar plot which you can see here, and we were we wanted to analyze this particular bar plot to understand whether some of the categories can be grouped together. So, if we look at this particular graphic we can see that some of the bars having zero values right, some of the pin codes and they have zero values, so that therefore, zero offers have been accepted in those locations here and here. So, of course, when the task is to predict the class of you know this our outcome variable promotional offer, so with

respect to that we can club these two locations in one group. Because the level of acceptance or rejection in these two particular locations two or three, and depending on because this is quite a you know big plot.

So, all those locations which have similar acceptance level, we probably can group them. So, for example, other bars we can see for example, this one first bar and the third bar they also have the similar acceptance level five, so probably we can group them. So, we can also identify many other you know locations for this one right. So, many other bars which are at the similar acceptance level, and probably we can group them. We can also group them by having a different range right. So, depending on the exercise and depending on the suitability of that particular grouping with respect to our model and its performance we can start grouping.

(Refer Slide Time: 02:52)



So, for example the pin codes with 0 to 5, you know acceptance count probably we can group them, then you know 5 to let us say 8 group 2, then 8 to 12 group 3. So, in this fashion also we can group these locations. So, we look at this particular scenario then will end up with this three groups. However, what we are going to perform here in this our exercises depending on the acceptance whether it is 0 or 1 or 2 because we have only you know maximum value is 12. So, for each of the acceptance level, we are going to create you know different groups. So, as you can see rather it is 13, yes, so depending on

the different grouping strategy can be done. So, we can have we can also do this range based you know grouping, we can also do this.

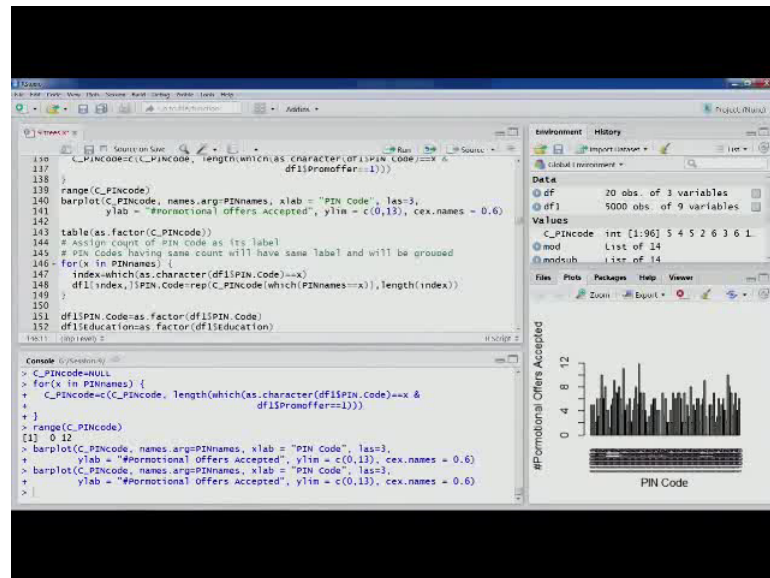
So, this one seems less suitable, but in our exercise we are going to perform this and this could be another grouping mechanism. So, but however, we will have to justify we will have to try and understand why this range and the why this particular range, we will have to understand probably these locations are having lower levels of acceptance. Probably these locations are having medium level of acceptance of promotional offers. Probably these locations having these acceptance numbers they are having slightly higher in our data set as per our data sets slightly higher level of acceptance.

So, in this fashion also we can perform grouping. However, for our exercise we would like to have you know so as you can understand we started from 96-pin locations pin codes right. So, for 96 we can have a this situation also three groups, and we can also have this one as well where we end up with thirteen groups. So, for our exercise we are going to perform this one; however, this can also be done. So, once this is understood we will have to do a few more computations, so that we are able to group all those records in appropriate category, new categories that we are going to create.

So, what we are going to do is the count of a pin code, the count that we have computed all right, the count of pin code that will treat as a level. So, if a particular pin code has zero acceptance count, so that becomes its level 0. If the pin code has a particular pin codes or number of pin codes which have one acceptance count, so that could be their level one. And if they have 5 or 10 or 12 acceptance count, so that is going to be level. So, all the locations depending on the acceptance counts that they have, so that is something that we are going to treat as the level of that particular location. This is mainly to simplify our coding and simplify our computation, so that we can easily group them.

Later on we want we can give them appropriate name instead of saying 1, 2, 3 or 13 or 0, we can also say group 0, group 1. So, later on that kind of transformation can be done, but for our purpose, we will stick to 0, 1 and up to 13 these thirteen levels and we will later on convert it into a factor variable with thirteen groups.

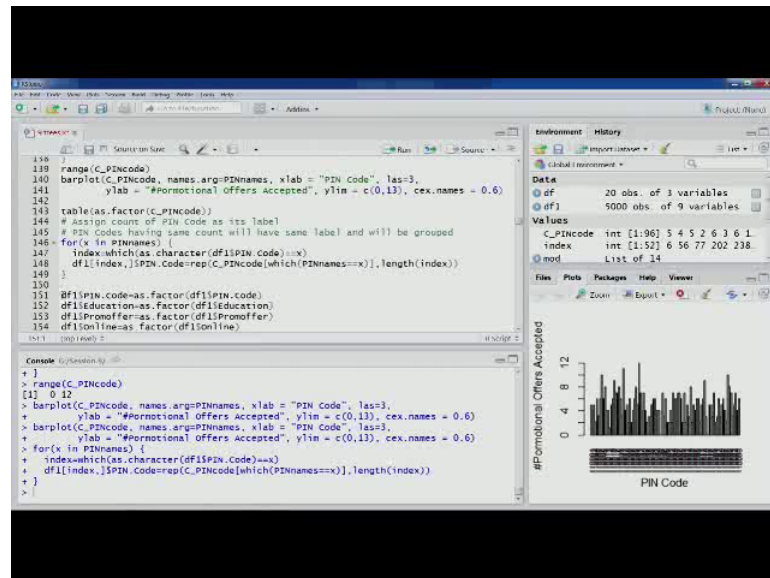
(Refer Slide Time: 06:56)



So, this particular loop as you can see. So, what we are trying to do here is a assign count of pin code as its level. So, pin codes having same count will have same level, and therefore that is how they will be grouped. So, loop you can see x and pin names, so for all the pin names all the pin codes that are there. So, loop will run for those many pin names for each of them. And first we try to compute the index where this particular you know x values they are same. So, first we select all the records, all the records having the same pin code. And once those records indices of those records is known with using index variable, then pin code of you know pin code of those indices that is being assigned this number which is nothing but the count of you know pin code, the acceptance count for that particular pin code.

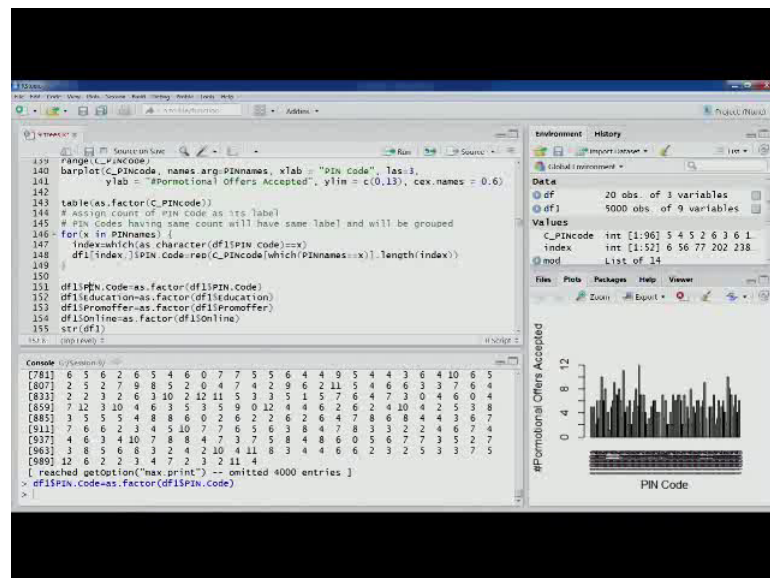
You can see C pin code where we had the counts we are trying to identify again you know the pin code indices where the same pin code is there; and once it is known the length of index that we already know, so that is going to be repeated. So, C pin code count of all the locations, all the records where same you know the same pin name is appearing same pin code is appearing. So, for all those you know we are going to give that count rep is the function, so it is going to repeat. So, this is going to be equal to the indices that we have already computed right. So, let us execute this particular code and then it will be more clear.

(Refer Slide Time: 08:50)



Once it is done, you would see that d f 1 pin code, you can also look at the environment section as well.

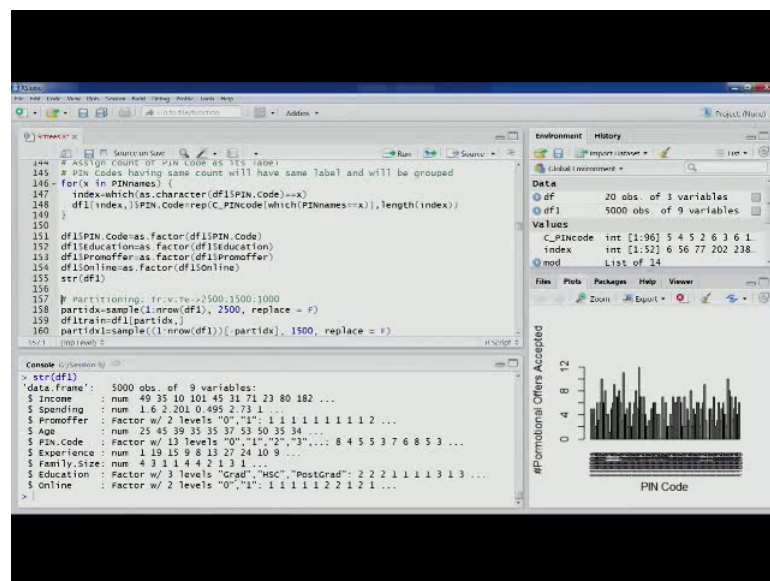
(Refer Slide Time: 09:03)



But if you compute this, you would see all the this particular variable different records that we had 5000s of them. So, all the records in the pin code variable, now we have the count right. So, 7, 3, 4, 4 these counts actually represent the acceptance level of those third particular location. So, because if the count is same, so they are again in the same level, so they can be easily grouped. So, in this fashion, we are trying to group them.

Now, once we convert this variable into a factor variable. So, now, if we again look at the values of this variable once it has been converted, so you can see levels you can easily see 0 to 12, so 12 levels are there. And now this particular variable has been converted into a factor variable. So, other variables that we wanted to transform into factor variables for education. So, three levels we had. So, let us convert it the promotional offer and also online.

(Refer Slide Time: 10:18)



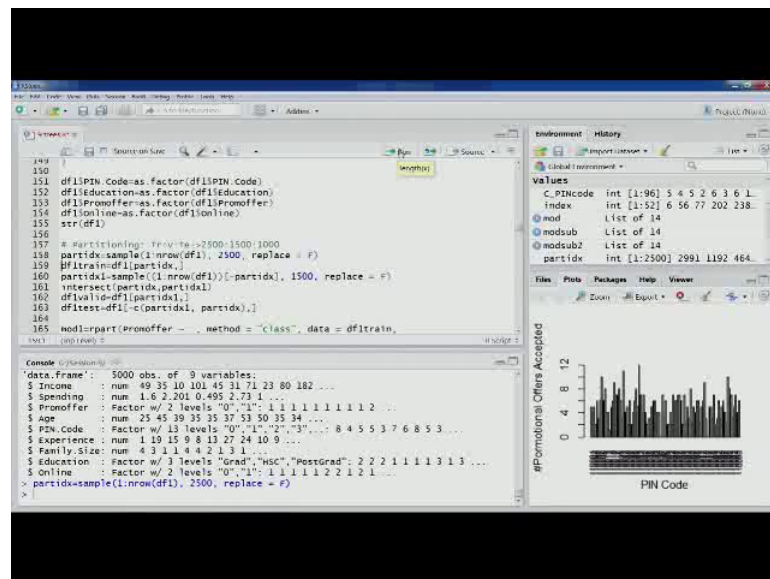
So, let us look at this structure now. However, education was I think already it was factor, so we repeated the exercise. So, in this fashion, you can see now the promotional offers factor variable appropriately mentioned here. The pin code now you can see 13 levels. So, we drop down the dimensionality from 96 to 13. So, one of them is going to be taken as the reference category. Now, the education and online or so 3 and 2 levels respectively. So, now, all the variables are in their desired variable type.

Now, we can go ahead and start without partitioning exercise. So, in this particular data set, we have 5000 observations; out of this 5000 observation, we will take the 50 percent of them that is 2500 observation in the training partition; out of the remaining 2500 observation, we will take first 1500 observation in the validation partition and the remaining 1000 observation in the test partition.

So, let us sample. So, the partitioning in this particular exercise slightly different, as you have been watching that in the previous other techniques, other lectures when we did

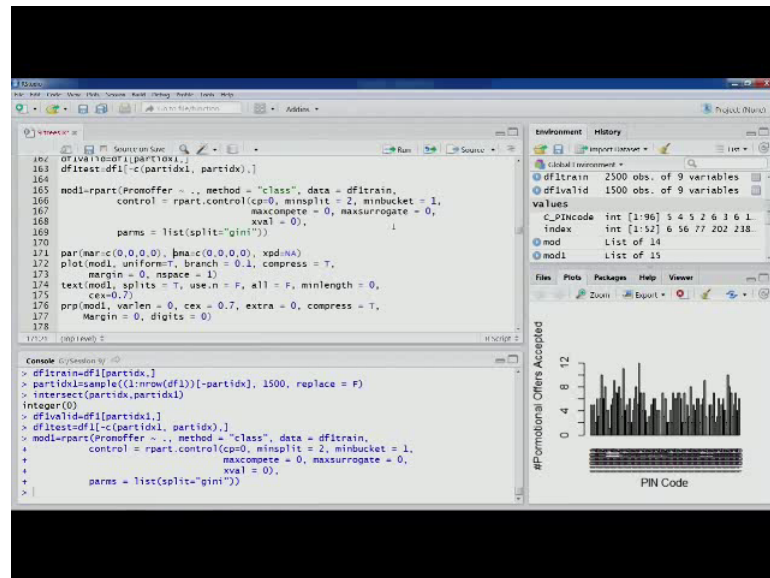
partitioning we just created two partition training and test partition. So, they are the training the sample and indices be we computed using the sample function randomly drawn indices and that were the part of the training partition, the remaining indices they were assigned to the test partition. Now, if you look at this these four-five lines of code for partitioning, first we are trying to randomly draw 2500 observation from the sample part any partition.

(Refer Slide Time: 12:03)



So, let us do this. So, you can see part index has been created in the environment section integer vector of 2500 observation. So, now, these observation can be safely assigned to the training partition. So, this is done training partition is created with the randomly drawn 2500 observations of all nine variables. Now, the second we again you know call sample function. And, now in this case, you would see that all the observations which are remaining now you can see this vector indices vector. And the remaining observation we do minus part idx. So, the remaining observations, so remaining indices, now, out of those indices, we can again randomly draw 50, 1500 you know further observation observations for our validation partition right. So, in this fashion that is again create this index.

(Refer Slide Time: 12:57)



Now, to because the way we have a randomly drawn these indices right replacement is also set to false. So, there is no overlapping observation in the training and the validation partition. If you want to check the same, you can check using intersect function. So, intersect function will give us if there are any if there are any similar rows similar values, so part idx and part idx 1. If we run intersect function here you would see we see no values. So, these are two different you know two different set of indices. So, now we can safely create our validation partition by selecting the 1500 randomly drawn indices. And the remaining indices, so remaining ones which is the where we part idx and part idx 1 both of them once we remove them out, the remaining one are going to be the part of the test partition. So, in this fashion we can actually go we can actually do our partitioning.

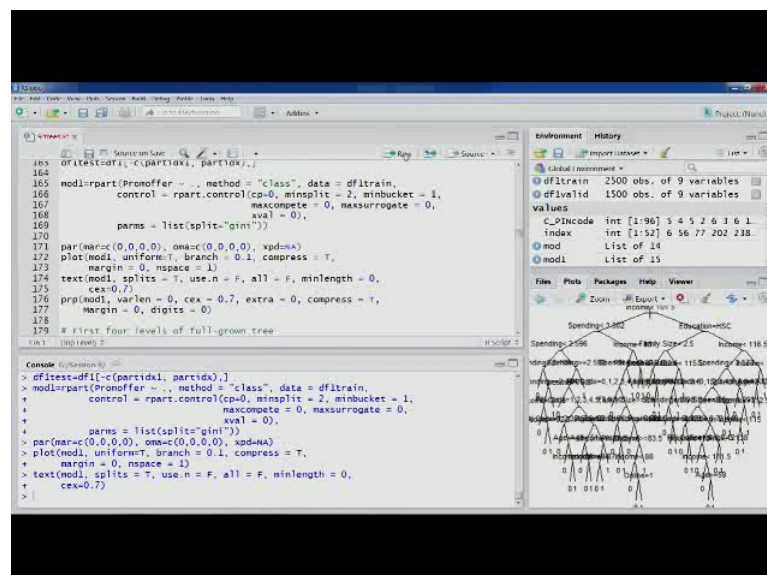
Now, we come to the next part that is once our partisans have been created we can use the training partition to build our model. So, if similar you know exercise that we did for the sedan car owner that sedan car dataset. So, here our outcome variable is we are using r part function within the r part function, the first argument is the formula. In the formula, you can see promo offer is our outcome variable and this is being modelled against all other variables which are predictors. Method is class for classification model. The data is appropriately mentioned as df1 train the training partition. R part control function which we talked about in the previous lecture took control a certain aspect of our tree model right; it is complexity parameter is 0, because we want to grow full you know full grown tree and we. So, minimum is split this two observations in min bucket

the bucket one observation and so all these parameters we have already talked about x value 0.

So, if we do not particularly you know I specify this value zero the default value is 10. So, some observation is going to be used for cross validation by r part function, which we do not want to do. So, we would like to use all the observations just for the training you know building the model. And the validation we have the validation partition for validate to form the validation. So, we do not want to make any we do not want to use any observation for this cross validation exercise that is inbuilt in the r part function. So, x value has to be 0. Other parameters are split this the gini metric that we have discussed in previous lectures. So, let us execute this code and will build the model.

So, now mod one is created as you can see in the environmental section. Now, let us look at the tree. So, let us set the parameters graphics parameters, margin outer margin. And x p d you can look at the parameter function par function for more detail. So, this is a basically x p d is basically to a you know generate your plot in the device region. So, you can if you are interested in more detail, you can find out from the help section. Now, once the pyramorphix parameter are set then we can generate the plot.

(Refer Slide Time: 16:40)

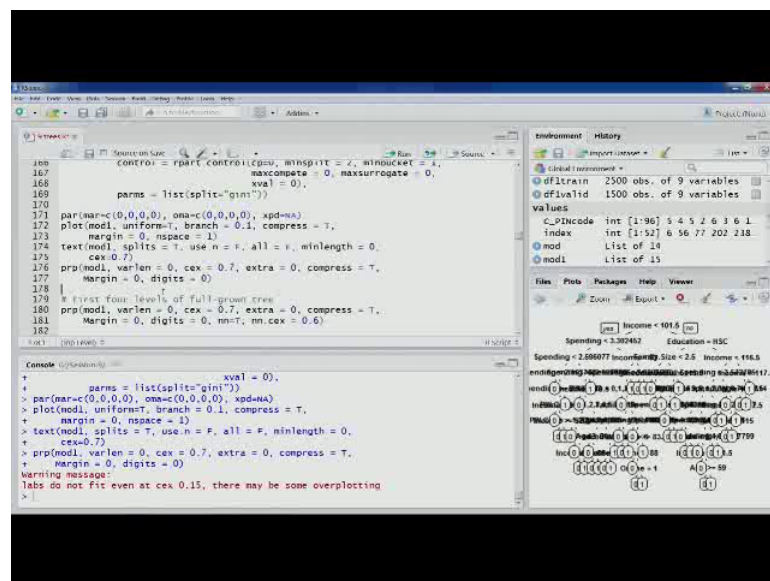


So, this is our basic plot. Let us add the information. Now, you can see the part is quite messy here. So, this is what we were talking about. If we have very large data set and we you know generate full grown tree, so it is going to be quite messy. So, you can see the

number of splits too many splits are there because as we talked about in the full grown tree, we also did the exercise where we were partitioning the observations right. In this (Refer Time: 17:13) dataset and we kept on partitioning till the all the observations were classified correctly.

So, the same kind of thing happens in a full grown tree where we continue to build our tree model till all the observations are classified till all the partitions that we create are your homogenous partition; that means, all the observations belong to the same class. So, because of that too many partitions and the full grown tree is going to be quite big as you can see here. If you want the nicer version or pretty version of this particular plot, so as we have been doing as we have done previously `prp` is the function that can be used. The relevant package we have already talked about.

(Refer Slide Time: 17:59)



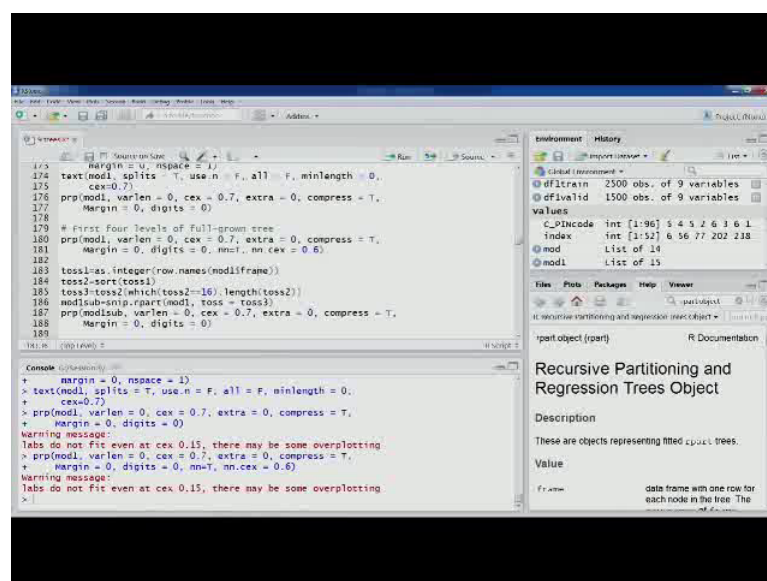
So, we can generate this. And you would see that this is the another way of representing this full grown tree. So, this is slightly better version, but again because the tree is quite big, so because of that this one also looks messy, but however, we can look at few things for example, first split is done using income variable and this is value is 101.5. And then the if you can look at other splits right, the spending and education, right then further spending here, income here, family size and income. So, you can also see pin code and you would see the different categories of pin codes, they have been used using comma. Had we used pin code as an ordinal variable right, then we have would have seen some

numeric kind of value right. We the budget I would have been treated as you know because it already had too many categories. So, we could have treated it as numeric variable. And then it would have some numeric value, because we have treated its categorical variable we can see specific categories as part of different sub trees.

So, we will discuss more on this as we go along, let us come back. So, the sniping exercise that we had done in with using the previous data set something similar we will have to perform in this particular case because this is quite a you know large tree and the full grown tree is quite large in this case. So, if we want to this, if we want to see just first four levels, so that we are able to understand what are the rules, what are the important variables, and how the split is happening if you want to visualize that. So, first four levels how we can go about this. So, first we need to do the node numbering as we talked about in the previous data set.

So, you can see node numbering has been done. So, all the nodes have been numbered now 1, 2, 3, starting 4, 5. So, because this tree is quite large, so you can see most of the node numbers visible in this case right the earlier some of the numbers were missing, but now you would see 1, 2, 3 is all the initial all the initial node numbers are there right up to 13, 14, 15. So, quite you know in a sequence any node numbers can be seen here. So, let us using these node numbers, we can always snip off the tree part that we are not interested.

(Refer Slide Time: 20:39)



```
173 margin = 0, nspace = 1)
174 text(mod1, splits = T, use.n = F, all = F, minlength = 0,
175     cex=0.7)
176 prp(mod1, varlen = 0, cex = 0.7, extra = 0, compress = T,
177     Margin = 0, digits = 0)
178
179 # First four levels of full-grown tree
180 prp(mod1, varlen = 0, cex = 0.7, extra = 0, compress = T,
181     Margin = 0, digits = 0, mnt = nn.cex = 0.6)
182
183 toss1=as.integer(row.names(mod1$frame))
184 toss2=sort(toss1)
185 toss3=toss2[which(toss2==16):length(toss2)]
186 mod1sub=snip.rpart(mod1, toss = toss3)
187 prp(mod1sub, varlen = 0, cex = 0.7, extra = 0, compress = T,
188     Margin = 0, digits = 0)
189
```

Console (RStudio):

```
> margin = 0, nspace = 1)
> text(mod1, splits = T, use.n = F, all = F, minlength = 0,
+ cex=0.7)
> prp(mod1, varlen = 0, cex = 0.7, extra = 0, compress = T,
+ Margin = 0, digits = 0)
Warning message:
In prp(mod1, varlen = 0, cex = 0.7, extra = 0, compress = T,
+ Margin = 0, digits = 0, mnt = nn.cex = 0.6) :
  Tabs do not fit even at cex 0.15, there may be some overplotting
> prp(mod1, varlen = 0, cex = 0.7, extra = 0, compress = T,
+ Margin = 0, digits = 0, mnt = nn.cex = 0.6)
Warning message:
In prp(mod1, varlen = 0, cex = 0.7, extra = 0, compress = T,
+ Margin = 0, digits = 0, mnt = nn.cex = 0.6) :
  Tabs do not fit even at cex 0.15, there may be some overplotting
>
```

Environment:

Object	Class	Attributes
dfftrain	data.frame	2500 obs. of 9 variables
dffvalid	data.frame	1500 obs. of 9 variables
C_P1ncode	int	[1:96] 5 4 5 2 6 1 6 1
Index	int	[1:52] 6 56 77 202 218
mod1	rpart	List of 14
mod1sub	rpart	List of 15

Files: RStudio | Packages: Help | Viewer

Recursive Partitioning and Regression Trees Object

Description

These are objects representing fitted rpart trees.

Value

From: data frame with one row for each node in the tree. The

So, `toss` was one of the argument, the second argument that we use in the `snip` are `r` function as you can see here in this particular line. So, we need to compute this we need to create this particular you know argument right variable the number of node actual node numbers which we want to snip off. So, let's first `k = toss[1]`. So, this is the function `mod` one and the frame. So, we talked about `r` part dot object, and one of the element there one of the attribute was frame. So, within frame we also have row names. So, row names will actually have these node numbers. So, more detail you can always find using the using the `r` part, this upper curve, you can see `r` part object. You can see it look at this page in the help section, you can see frame.

(Refer Slide Time: 21:40)

```

174 text(mod1, splits = T, use.n = F, all = F, minlength = 0,
175      cex = 0.7)
176 prp(mod1, varlen = 0, cex = 0.7, extra = 0, compress = T,
177      Margin = 0, digits = 0)
178
179 # First four levels of full-grown tree
180 prp(mod1, varlen = 0, cex = 0.7, extra = 0, compress = T,
181      Margin = 0, digits = 0, nncf, nn.cex = 0.6)
182
183 toss1 = as.integer(row.names(mod1$frame))
184 toss2 = sort(toss1)
185 toss3 = toss2[which(toss2 == 16).length(toss2)]
186 mod1sub = snip.rpart(mod1, toss = toss3)
187 prp(mod1sub, varlen = 0, cex = 0.7, extra = 0, compress = T,
188      Margin = 0, digits = 0)
189
190
191 Console (RStudio)
> text(mod1, splits = T, use.n = F, all = F, minlength = 0,
+      cex = 0.7)
> prp(mod1, varlen = 0, cex = 0.7, extra = 0, compress = T,
+      Margin = 0, digits = 0)
warning message:
tabs do not fit even at cex = 0.15, there may be some overplotting
> prp(mod1, varlen = 0, cex = 0.7, extra = 0, compress = T,
+      Margin = 0, digits = 0, nncf, nn.cex = 0.6)
warning message:
tabs do not fit even at cex = 0.15, there may be some overplotting
> toss1 = as.integer(row.names(mod1$frame))
>

```

Environment History

Global Environment	
mod1sub2	list of 14
partidv	int [1:7500] 2991 1192 464
partidid	int [1:1100] 1176 963 5128
pinnames	chr [1:96] "110001" "110000"
toss1	int [1:109] 1 2 4 8 16 17
cpin	table 'int [1:96(1d)] 14
v	"110000"

File Plot Packages Help Viewer

at: recursive partitioning and regression using object

Frame:

data frame with one row for each node in the tree. The row names of frame contain the (unique) node numbers that follow a binary ordering indexed by node depth. Columns of frame include `var`, a factor giving the names of the variables used in the split at each node (leaf nodes are denoted by the level `<leaf>`), `n`, the number of observations reaching the node, `ur`, the sum of case weights for observations

So, for any `r` part object there is going to be a frame attribute which will be actually a data frame with one row for each node in the tree. And the row dot names of frame contain the unique node numbers that follow a binary ordering indexed by node depth. So, these are the node numbers that we actually shown in the tree. The same node numbers are stored in this particular in this particular attribute frame. So, from this we row, row dot names we are trying to extract that information and then those number those names and we are trying to convert it into integer variable.

So, let us compute this `toss` one. So, you can see `toss` one having 109, since we have 109 total nodes that tree the full grown tree that we had with it has 109 you know nodes. So, all those nodes the node numbers the unique node numbers that are assigned by the

algorithm, so that has that has been captured. Once these numbers have been captured, we can sort them. So, all this we are doing, so that we are able to identify the nodes which we want to snip off.

So, once we solved, so these numbers might not be in order. So, if you want to have a look at this you would see that 1, 2, 4, 8, 16, 17, then is suddenly 34. So, the this numbering is slightly the way this is recorded in row names is slightly different right. You can see 1, 2, and 4, 8. So, you can see the row names they are recorded in this fashion 1, 2, 4, 8, and then 16, 17 then 34, 68 right. So, in this fashion these row numbers are recorded. So, this is slightly different. So, therefore, we will have to sort this out.

(Refer Slide Time: 23:42)

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

```

173 margin = 0, nspace = 1)
174 text(modl, splits = T, use.n = F, all = F, minlength = 0,
175      cex = 0.7)
176 prp(modl, varlen = 0, cex = 0.7, extra = 0, compress = T,
177      Margin = 0, digits = 0)
178
179 # First four levels of full-grown tree:
180 prp(modl, varlen = 0, cex = 0.7, extra = 0, compress = T,
181      Margin = 0, digits = 0, vncf.mn.cex = 0.6)
182
183 toss1 = as.integer(row.names(modl$frame))
184 toss2 = sort(toss1)
185 toss3 = toss2[which(toss2 == 16):length(toss2)]
186 modlsub = snip.rpart(modl, toss = toss3)
187 prp(modlsub, varlen = 0, cex = 0.7, extra = 0, compress = T,
188      Margin = 0, digits = 0)
189

```

The console output shows the following:

```

Warning message:
'labs do not fit even at cex 0.15, there may be some overplotting'
> toss1 = as.integer(row.names(modl$frame))
> toss1
 [1] 1 2 4 8 16 17 34 68 69 138 139 35 9 18 36 37
[17] 74 148 149 298 299 598 599 75 150 300 301 602 603 151 302 604
[33] 605 303 19 5 10 20 40 41 82 164 165 330 331 83 166 167
[49] 334 668 669 1338 1339 335 21 11 22 23 3 6 12 24 25 50
[65] 51 13 26 52 104 105 210 211 53 27 7 14 28 56 112 224
[81] 225 450 451 113 226 452 453 906 907 1814 1815 227 454 455 57 114
[97] 115 29 58 59 118 236 237 119 15 30 60 61 31
> toss2 = sort(toss1)
>

```

The Environment pane on the right shows the following objects:

- modlsub2: list of 14
- partidx: int [1:2100] 2891 1182 484
- partidx1: int [1:1100] 1176 963 3128
- pinnames: chr [1:96] "110001" "11000"
- toss1: int [1:108] 1 2 4 8 16 17
- toss2: int [1:108] 1 2 3 4 5 6 7
- cpkn: "table" int [1:96(16)] 14

So, let us create another variable sorted values. So, once these values have been sorted we would like to identify the nodes which we want to get rid off. So, how we can do this? So, let us again zoom into the full grown tree and with node numbers. So, for example, first four levels, so level 1, 2, and 3 and 4. So, after these four levels, we would like to get rid of the remaining part of the remaining part of the tree, that means, the nodes is starting from node number 16. So, let us go back to the code, you can see. Same thing I mentioned here the toss to this is once you know so 16; and up to the last node right length of the tossed. So, from these node numbers starting at 16 to the last node, I would like to get rid off this node, so that we just see the first four levels.

(Refer Slide Time: 24:42)

```

173 margin = 0, digits = 0, mnt, mn cex = 0.6)
174 text(modl, splits = T, use.n = F, all = F, minlength = 0,
175      cex = 0.7)
176 prp(modl, varlen = 0, cex = 0.7, extra = 0, compress = T,
177      Margin = 0, digits = 0)
178
179 # First four levels of full-grown tree
180 prp(modl, varlen = 0, cex = 0.7, extra = 0, compress = T,
181      margin = 0, digits = 0, mnt, mn cex = 0.6)
182
183 toss1 = as.integer(row.names(modl$frame))
184 toss2 = sort(toss1)
185 toss3 = toss2[which(toss2 == 16):length(toss2)]
186 modlsub = snip.rpart(modl, toss = toss3)
187 prp(modlsub, varlen = 0, cex = 0.7, extra = 0, compress = T,
188      Margin = 0, digits = 0)
189
190
191 > toss1 = as.integer(row.names(modl$frame))
192 > toss1
193      1      2      4      8     16     17     34     68     69    138    139    35     9     18     36     37
194 [17]  74  148  149  298  299  598  599   75  150  300  301  602  603  151  302  604
195 [33] 605 303  19   5  10  20  40  41  82  164  165 330 331  83  166  167
196 [49] 334 668 669 1338 1339 335 21  11  22  23   3   6  12  24  25  50
197 [65]  51  13  26  52  104 105 210 211  53  27   7  14  28  56 112 224
198 [81] 225 450 451 113 226 452 453 906 907 1814 1815 227 454 455 57 114
199 [97] 115 29 58 59 118 236 237 119  15  30  60  61  31
200
201 > toss2 = sort(toss1)
202 > toss3 = toss2[which(toss2 == 16):length(toss2)]
203

```

The Environment pane shows the following objects:

- modlsub2: list of 14
- partidx: int [1:2500] 2991 1192 464
- partidx1: int [1:1500] 1176 965 3528
- pinnames: chr [1:96] "110001" "11000
- toss1: int [1:108] 1 2 4 8 16 17
- toss2: int [1:108] 1 2 3 4 5 6 7
- toss3: int [1:94] 16 17 18 19 20

Let us compute this. This is done. Now, as you can see I am using snip dot r part function here. So, mode one and we would like to snip it using the toss tree argument containing the nodes to get rid off. So, this is done.

(Refer Slide Time: 24:56)

```

181 margin = 0, digits = 0, mnt, mn cex = 0.6)
182
183 toss1 = as.integer(row.names(modl$frame))
184 toss2 = sort(toss1)
185 toss3 = toss2[which(toss2 == 16):length(toss2)]
186 modlsub = snip.rpart(modl, toss = toss3)
187 prp(modlsub, varlen = 0, cex = 0.7, extra = 0, compress = T,
188      Margin = 0, digits = 0)
189
190 # Description of each splitting step of the full-grown tree
191 # No. of Decision Nodes
192 nrow(modl$split)
193 # No. of Terminal Nodes
194 nrow(modl$frame) - nrow(modl$split)
195
196 # j is counter for split variables
197 # i is counter for split values
198
199 > prp(modlsub, varlen = 0, cex = 0.7, extra = 0, compress = T,
200      Margin = 0, digits = 0)
201

```

The Environment pane shows the following objects:

- modlsub: list of 15
- modlsub: list of 14
- modlsub2: list of 14
- partidx: int [1:2500] 2991 1192 464
- partidx1: int [1:1500] 1176 965 3528
- pinnames: chr [1:96] "110001" "11000
- toss1: int [1:108] 1 2 4 8 16 17

Now, we can use p r p function to print the tree. Now, you would see just four levels of the tree. Now, this is quite clear easy to understand what is going on here. So, you can see first split is income less than 101.5 and then we have split using spending and this split using education at the right part. The left part is spending then further

spending, income, family size, income. So, these are some of the common variables they would see income and spending they remain the two important variables here, income spending, family size is also there and the pin code is also visible here in this part you can see pin code is also there. So, income, spending, education, family size, pin code are the important variables; however, the income and spending seems to be occurring more often.

So, now since the whole full grown tree is developed to understand more about this particular full tree. We can look at few more things for example, number of decision nodes that are there. So, again `r` part object this particular split attribute that is there it will give us these splits contains the information about the variables used for a split. So, we can also a length off this particular this particular very attribute will give us the number of additional nodes. So, 54 decision nodes have been used. And we look at the terminal nodes. So, a total number of nodes would be can we find out by the this frame as we risk as we looked at the health section that frame contains a unique row number for each of the node.

So, therefore, it can give us the total number of nodes that are there. You can see 109 which we already know. And then once we subtract the number of decision nodes will get the number of terminal nodes 55. So, as you can see number of decision nodes 54 and number of terminal nodes you know 55, which is much more one more than the number of decision nodes. This is property of binary trees. In binary trees the number of terminal nodes or number of leaves are one more than the decision nodes.

Now, if we are interested in having a table where we have the information about the variables which have been used for splitting and the split values. So, the predictor value combination, if we want to look at look at that list, so by default the output that we get out of summary function, it is more descriptive right. So, we would like to have a tabular, tabular output then we would have to write the code for the same. So, here we are essentially trying to do this. So, we are trying to capture the split variable information and a split value information. So, particularly split value for each of the splitter split variable, because as we saw that the in `r` part object split function has the information on the variables used for split. So, for all those variables can we compute the can we extract the split values from the model.

(Refer Slide Time: 28:20)

```
# NO. OF SPLITTING NODES
191 nrow(mod1$plits)
192 # NO. OF Terminal Nodes
193 nrow(mod1$frame) - nrow(mod1$plits)
194 # i is counter for split variables
195 # j is counter for split values
196 splitvalue=NULL; j=1; i=1
197 for(x in mod1$framevar) {
198   if (is.character(x) && "leaf"==x) {
199     if (is.factor(df[,as.character(x)])) {
200       splitvalue[i]=mod1$plits[j,"index"]
201     }
202   } else {
203     # split variable is a factor
204     # k = (1, largest number of levels in the factors)
205   }
206 }
207
```

Environment

- Global Environment
- mod1sub List of 15
- modsub List of 14
- modsub2 List of 14
- partidx int [1:2500] 2991 1192 464.
- partidx1 int [1:1500] 1176 965 3528.
- Pinnares chr [1:96] "110001" "11000.
- toss1 int [1:109] 1 2 4 8 16 17

Files Plots Packages Help Viewer

Frame

data frame with one row for each node in the tree. The row.names of frame contain the (unique) node numbers that follow a binary ordering indexed by node depth. Columns of frame include: var, a factor giving the names of the variables used in the split at each node (leaf nodes are denoted by the level "leaf"); n, the number of observations reaching the node; var, the sum of case weights for observations

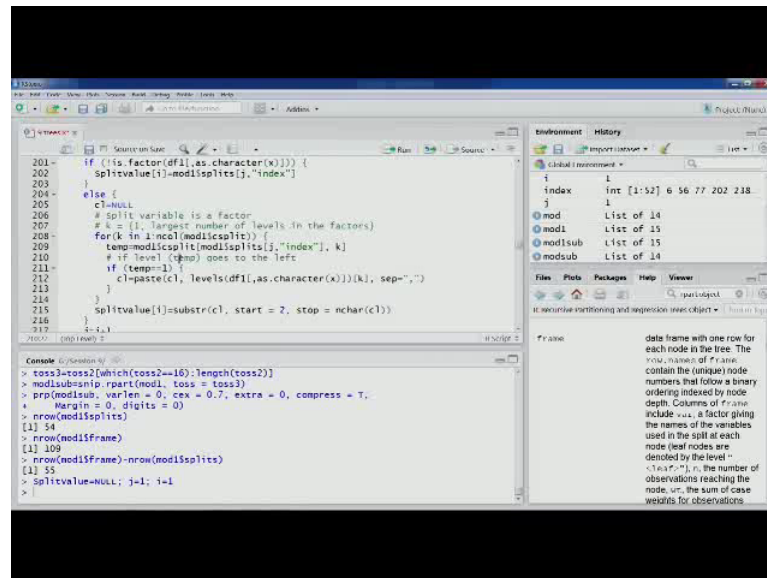
Console

```
> toss2=sort(toss1)
> toss3=toss2[which(toss2==16):length(toss2)]
> mod1sub=snip.rpart(mod1, toss = toss3)
> prp(mod1sub, varlen = 0, cex = 0.7, extra = 0, compress = T,
+ margin = 0, digits = 0)
> nrow(mod1$plits)
[1] 34
> nrow(mod1$frame)
[1] 109
> nrow(mod1$frame)-nrow(mod1$plits)
[1] 55
>
```

So, let us compute the split values. So, split value lets compute. So, we have to counter j is the counter file split variable i is counter for split values. So, then a split value is the this variable where we are going to record all the split values. So, let us initialize this one on also the counters. Then in the loop you can see x in mode one frame var. So, for all the variables that are there. So, frame has all the information on all the variables. So, for we run a loop for all the variables, and then we place a check there if as dot character and leaf.

So, if the particular variable is the leaf node right if the particular you know that variable is leaf node then we would like to skip that you know if it is not leaf node will like to continue. If it is a leaf node, we would like to skip and go to the else part. So, within this we see that if the variable is not factor right, there is the split variable could be factor or numeric. So, again we do a check if the split variable is not factor that means, numeric variable then simply the value of that split value can be found find out using this particular code where the splits attribute that we discussed. And the index column that is there the index particular column is actually contains the split value. And j is anyway counter for a split variable. So, for that split variable and index column will have this split value will be immediately recorded or captured here; if we go to the else part which will essentially deal with the factor variable.

(Refer Slide Time: 30:11)



```
201- if (!is.factor(df1[,as.character(x)])) {
202-   splitvalue[i]=mod$split[j,"index"]
203- }
204- else {
205-   c1=NULL
206-   # split variable is a factor
207-   # k = [1, largest number of levels in the factors]
208-   for(k in 1:ncol(mod$split)) {
209-     temp=mod$split[j,"index",k]
210-     # if level (temp) goes to the left
211-     if (temp=="1") {
212-       c1=paste(c1, levels(df1[,as.character(x)])(k), sep=",")
213-     }
214-   }
215-   splitvalue[i]=substr(c1, start = 2, stop = nchar(c1))
216- }
217- }
218- }

Console: R Session R/
> toss3=toss2[which(toss2==10),length(toss2)]
> mod$sub=splitpart(mod, toss = toss3)
> prp(mod$sub, varlen = 0, cex = 0.7, extra = 0, compress = T,
+ Margin = 0, digits = 0)
> nrow(mod$split)
[1] 54
> nrow(mod$frame)
[1] 109
> nrow(mod$frame)-nrow(mod$split)
[1] 55
> splitvalue=NULL; j=1; i=1
>
```

Environment: Global Environment, Index, mod, mod1, mod\$sub, mod\$split

Files, Plots, Packages, Help, Viewer

data frame with one row for each node in the tree. The row.names of frame contain the (unique) node numbers that follow a binary ordering indexed by node depth. Columns of frame include: 1. a factor giving the names of the variables used in the split at each node (leaf nodes are denoted by the level = "leaf", n, the number of observations reaching the node, var, the sum of case weights for observations

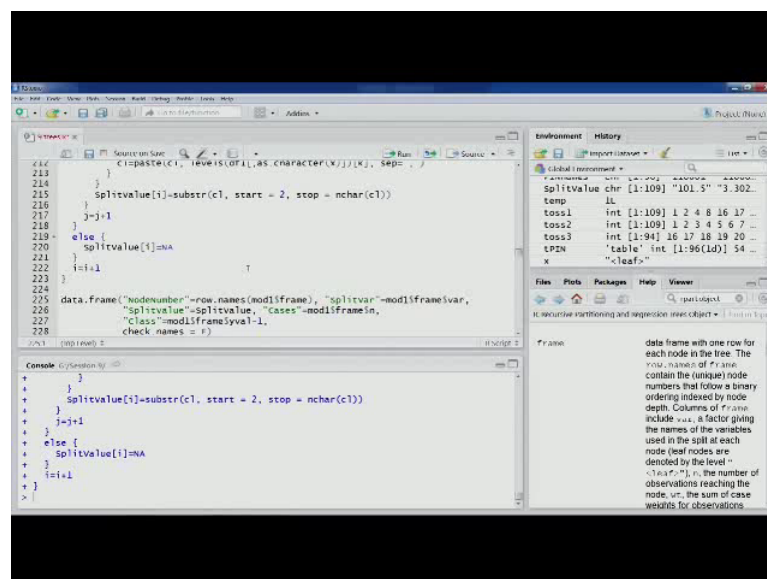
So, here if we go to this part, we have another variable `c1` as null initialized. So, split if this spirit variable is factor. So, `k` is another counter that we are starting which is ranging from 1 to largest number of levels in the factors. So, the different factor variables that we have the pin code had the largest number of categories right. So, it had thirteen categories. So, `k` will we you know lie between 1 and 13. So, we are running a loop for the those number of categories. So, you can see `k` in one, two number of column that are then and `c1` is split this is particular attribute actually contains information about factor variables. So, it will have that information. And once it has that information, so we can run the loop here. And within this we can look at we can again we are getting this temp variable the we are recording this information of index.

So, this index and particular level, so for a particular variable and its levels, so there are going to be different categories for factor variable. So, we get further information so that information would be whether the that particular whether the level of so level of that particular variable is going to be recorded in temp. So, if that goes into the left child, so for a particular for particular factor variable or categorical variable, if it has four categories. So, once the tree is being constructed, and the categorical variable is the split variable, we have to check which category is going to the left side and which categories are going to the right side right. Whether `a` is going here, and `b`, `c`, `d` are going away to the right side.

So, the same information as the same thing is being captured in this code. So, once we have recorded whether a particular you know whether particular level right that is represented by k right. So, k is running for all the levels right; maximum of levels that will be run for all the you know it can be used for all the variables and the all the levels. So, if it is goes for left goes into the that particular level is goes to the left child, left branch, then that is being recorded here right that is being recorded in c l that c l variable that we had created class. So, this we are recording here. And then you can see the next line here, and the else part itself. Again for all the levels, so this loop will run and for all the levels we would end up recording this, we would end up recording all the levels right.

So, those levels are actually nothing but the values specific values for the categorical variables right; just like numeric variable you know the specific value which was used for split. So, the these level different levels which level has gone to the left part, which level has gone to the right part, they actually represent the they actually represent the split value. So, you can see spirit value that variable that we had initialized. So, numerical variables are being stored in the if part; and in else part we are storing the categorical variable. So, here will keep on storing using this particular function, in this particular code we will keep on storing the value for categorical.

(Refer Slide Time: 33:54)



Now, the second else part that we have the else section that we have so, this was for the leaf node. So, if we come in arrive at a leaf node then we assign a spirit value as NA, because leaf node is not the ricean node and it will not have any split value, and then we continue with our counter. So, this is the code. So, this is how we can go about capturing extracting the split values.

So, we will continue our discussion. We will stop here will continue our discussion in the next lecture from the same point

Thank you.