Dealing with Materials Data: Collection, Analysis and Interpretation

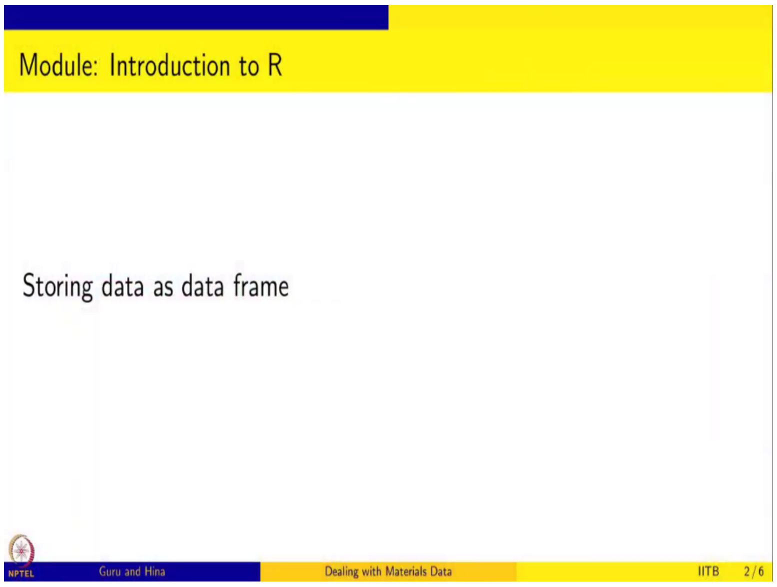
Professor M.P. Gururajan
Professor Hina A. Gokhale
Department of Metallurgical Engineering and Materials Science
Indian Institute of Technology Bombay

Lecture No. 14

Data frame in R: Properties of elements

Welcome to dealing with materials data. In this course, we are going to look at collection, analysis and interpretation of materials data. We are in the first module. This is an module 2 introduction to R and we are looking at specific data set and we have been working with the data set, manipulating it and plotting it and so on. So, in this session we are going to learn about storing data as a data frame.

(Refer Slide Time: 0:43)



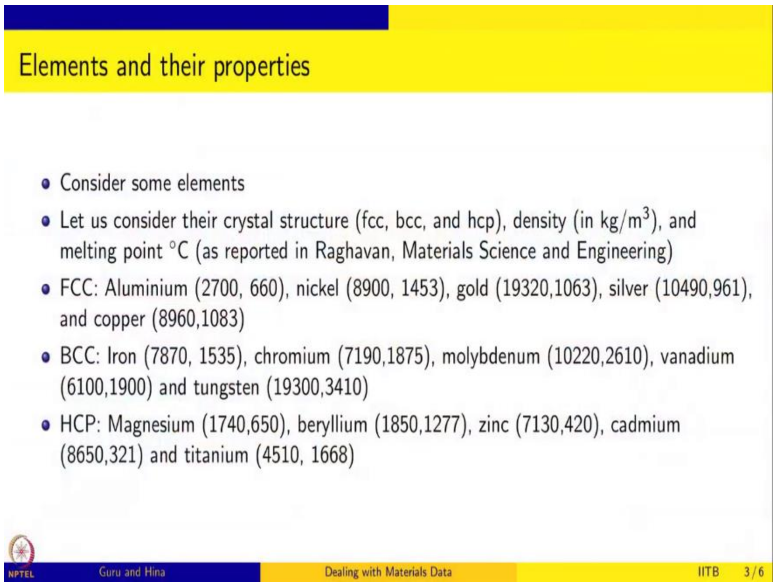
Module: Introduction to R

Storing data as data frame

Guru and Hina | Dealing with Materials Data | IITB 2/6

So, we have already looked at storing data as a table and as a data table. So, this is for storing the data as a data frame.

(Refer Slide Time: 0:55)



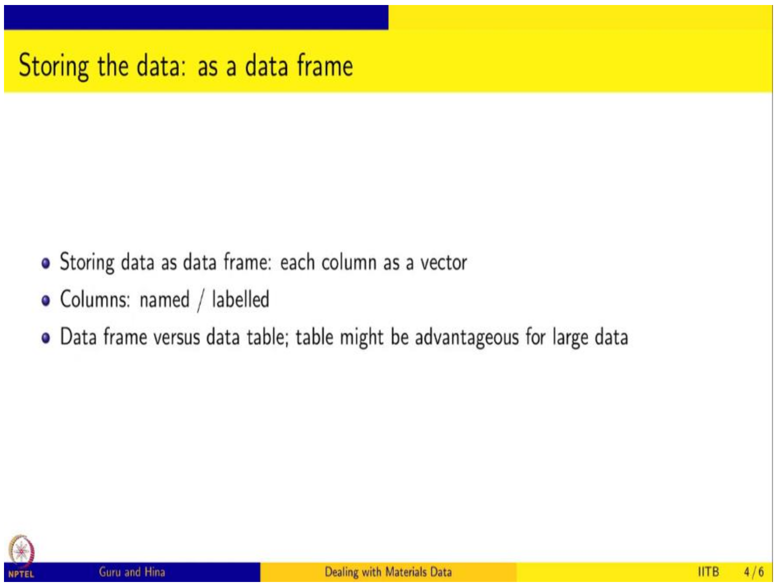
Elements and their properties

- Consider some elements
- Let us consider their crystal structure (fcc, bcc, and hcp), density (in kg/m^3), and melting point °C (as reported in Raghavan, Materials Science and Engineering)
- FCC: Aluminium (2700, 660), nickel (8900, 1453), gold (19320,1063), silver (10490,961), and copper (8960,1083)
- BCC: Iron (7870, 1535), chromium (7190,1875), molybdenum (10220,2610), vanadium (6100,1900) and tungsten (19300,3410)
- HCP: Magnesium (1740,650), beryllium (1850,1277), zinc (7130,420), cadmium (8650,321) and titanium (4510, 1668)

NPTEL Guru and Hina Dealing with Materials Data IITB 3/6

And in this session, we are going to again consider the same data. So, it is a data on some elements. It consists of their crystal structure, density and melting point. And we have about 15 elements and 5 fcc, 5 bcc and 5 hcp. And this is the data set that we have been working for the past few sessions. And this is what we are going to use.

(Refer Slide Time: 1:18)



Storing the data: as a data frame

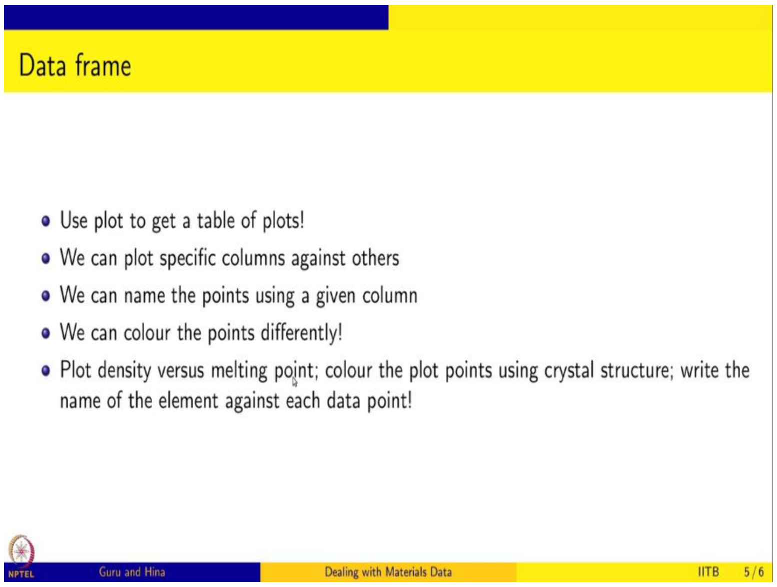
- Storing data as data frame: each column as a vector
- Columns: named / labelled
- Data frame versus data table; table might be advantageous for large data

NPTEL Guru and Hina Dealing with Materials Data IITB 4/6

And in this session, so we are going to store data as a data frame. As you will see, the advantage with storing data as a data frame is that you can input it very easily. Each column is input as a vector. And you can of course, name or label the columns. And so like I mentioned earlier data frame versus data table, so it might be advantageous for large data to store data as the table.

But we are not going to use too many large data in this course, so we will mostly use data frame but you already know how to deal with the data tables and work with them in any case.

(Refer Slide Time: 2:04)



Data frame

- Use plot to get a table of plots!
- We can plot specific columns against others
- We can name the points using a given column
- We can colour the points differently!
- Plot density versus melting point; colour the plot points using crystal structure; write the name of the element against each data point!

NPTEL Guru and Hina Dealing with Materials Data IITB 5/6

So, what is it that we are going to do in this session? So, we are going to plot the complete data as a table of plots. So, and then we are going to plot specific columns against specific columns. Okay, so these are individual plots, and we want to name the points of the scatterplot using a given column.

And we want to color the points differently. Specifically, what we want to do is the following, we want to plot density versus melting point, we want to color the plot points using crystal structure. So, fcc, bcc, hcp should get different colored points. And next to the element, their names should be written. And all this data is already available in the data in the data frame that we have. So, we must be able to take it and do this so, this is going to be the exercise.

(Refer Slide Time: 3:01)

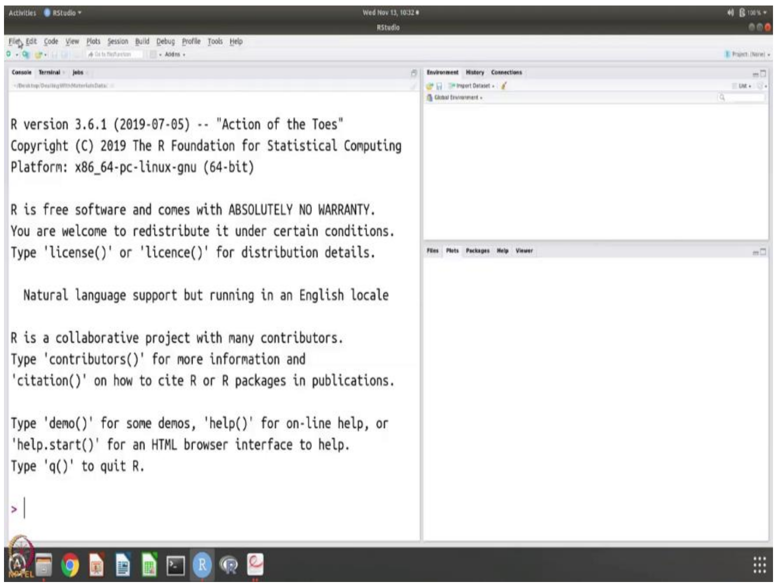
Other libraries

ggplot2 for plotting

Guru and Hina   Dealing with Materials Data   IITB   6 / 6

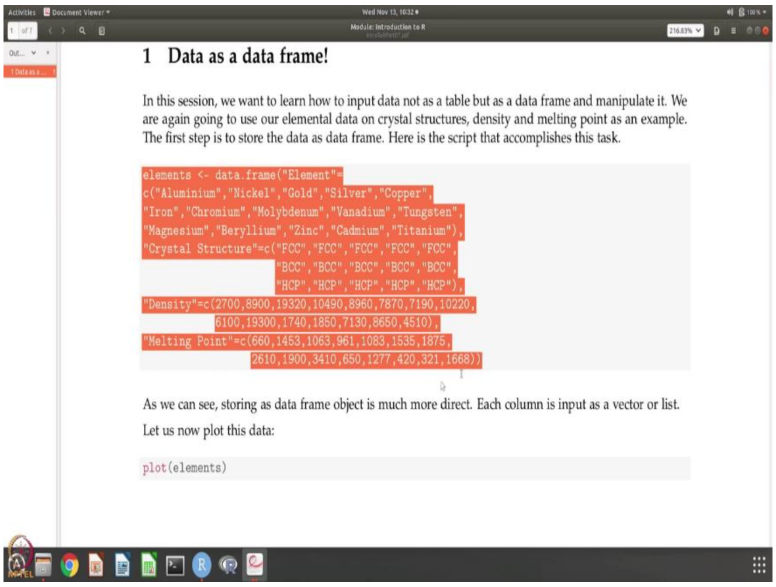
And we are also going to learn about other libraries that are available for plotting. So specifically, we are going to learn about ggplot2 for plotting. So that is going to be the, the aim of this session. So to learn how to take the data frame and do all this and learn about ggplot2.

(Refer Slide Time: 3:26)



File Edit Code View Plots Session Build Debug Profile Tools Help
Console Terminal Jobs
Environment History Connections
Global Environment
Files Plots Packages Help Viewer

```
R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```



1 Data as a data frame!

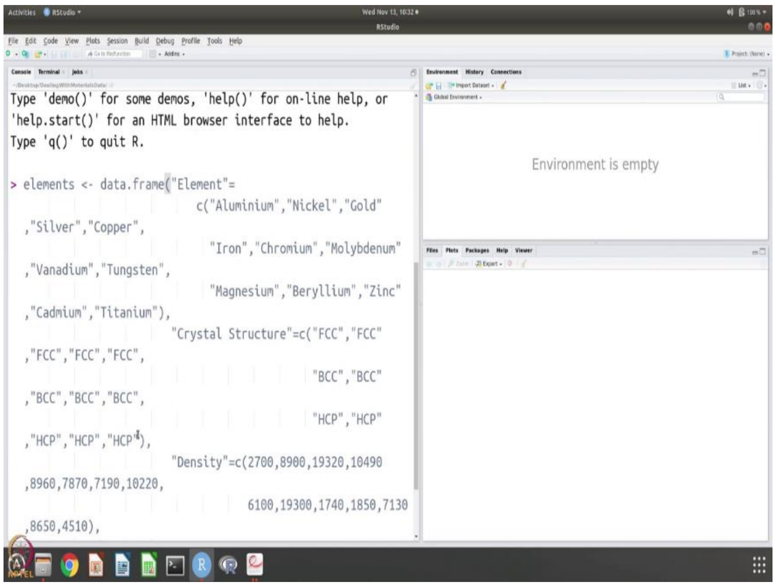
In this session, we want to learn how to input data not as a table but as a data frame and manipulate it. We are again going to use our elemental data on crystal structures, density and melting point as an example. The first step is to store the data as data frame. Here is the script that accomplishes this task.

```
elements <- data.frame("Element"=
c("Aluminium","Nickel","Gold","Silver","Copper",
"Iron","Chromium","Molybdenum","Vanadium","Tungsten",
"Magnesium","Beryllium","Zinc","Cadmium","Titanium"),
"Crystal Structure"=c("FCC","FCC","FCC","FCC","FCC",
                      "BCC","BCC","BCC","BCC","BCC",
                      "HCP","HCP","HCP","HCP","HCP"),
"Density"=c(2700,8900,19320,10490,8960,7870,7190,10220,
            6100,19300,1740,1850,7130,8650,4510),
"Melting Point"=c(660,1453,1063,961,1083,1535,1875,
                  2610,1900,3410,650,1277,420,321,1668))
```

As we can see, storing as data frame object is much more direct. Each column is input as a vector or list.

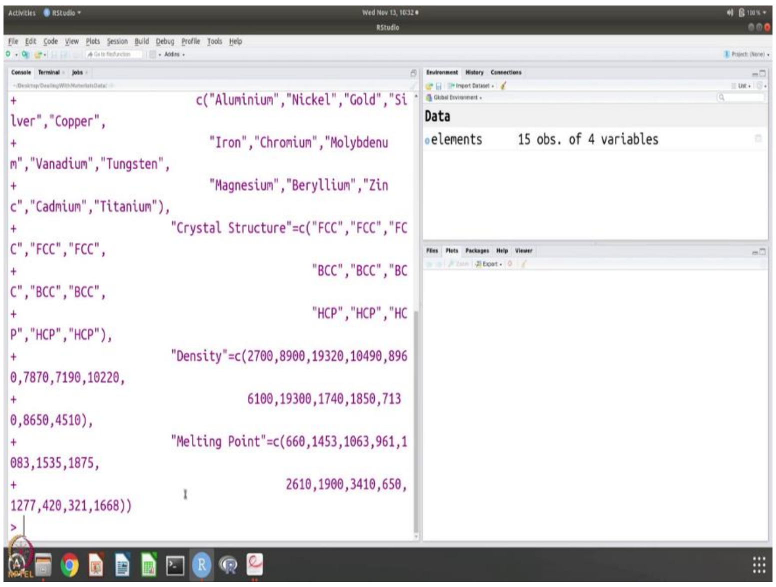
Let us now plot this data:

```
plot(elements)
```



```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> elements <- data.frame("Element"=
                          c("Aluminium","Nickel","Gold"
 ,"Silver","Copper",
                            "Iron","Chromium","Molybdenum"
 ,"Vanadium","Tungsten",
                            "Magnesium","Beryllium","Zinc"
 ,"Cadmium","Titanium"),
                       "Crystal Structure"=c("FCC","FCC"
 ,"FCC","FCC","FCC",
                                             "BCC","BCC"
 ,"BCC","BCC","BCC",
                                             "HCP","HCP"
 ,"HCP","HCP","HCP"),
                       "Density"=c(2700,8900,19320,10490
 ,8960,7870,7190,10220,
                                   6100,19300,1740,1850,7130
 ,8650,4510),
```

Environment is empty



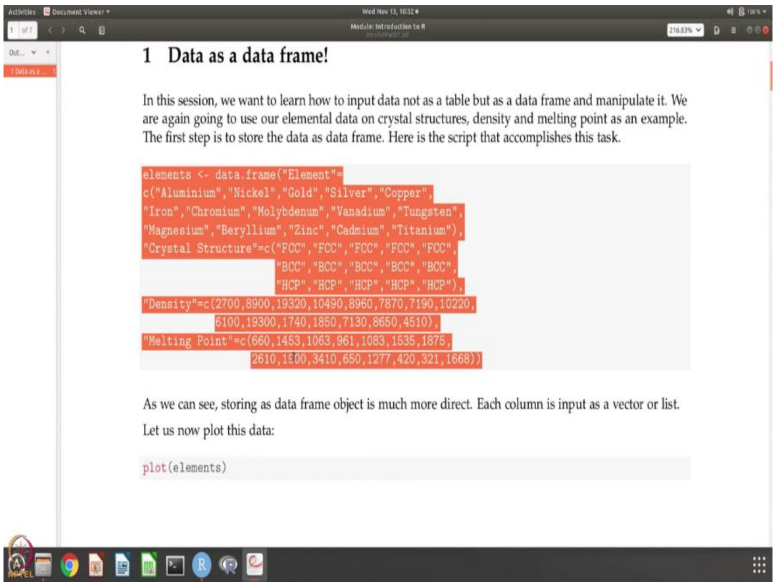
```
+                         c("Aluminium","Nickel","Gold","Si
lver","Copper",
+                           "Iron","Chromium","Molybdenu
m","Vanadium","Tungsten",
+                           "Magnesium","Beryllium","Zin
c","Cadmium","Titanium"),
+                      "Crystal Structure"=c("FCC","FCC","FC
C","FCC","FCC",
+                                            "BCC","BCC","BC
C","BCC","BCC",
+                                            "HCP","HCP","HC
P","HCP","HCP"),
+                      "Density"=c(2700,8900,19320,10490,896
0,7870,7190,10220,
+                                  6100,19300,1740,1850,713
0,8650,4510),
+                      "Melting Point"=c(660,1453,1063,961,1
083,1535,1875,
+                                        2610,1900,3410,650,
1277,420,321,1668))
>
```

Data
elements   15 obs. of 4 variables

So, let us do that. For doing that I am going to invoke R and so let us first look at how to give the data. So, this is similar to what we have been doing. So, let me take this command and put it here. So, what does it do? So it says the elements is a data frame and it has several columns. So, this is one column, for example, element is the first column.

So, that has aluminum, nickel, gold, silver, copper, iron, chromium, molybdenum, vanadium, etcetera up to titanium and the next column is their crystal structure. So, first five are fcc, next five are bcc, next five are hcp and so on. And next is the density, so that is the third column and melting point is the 4th column

(Refer Slide Time: 4:22)


Module: Introduction to R

1 Data as a data frame!

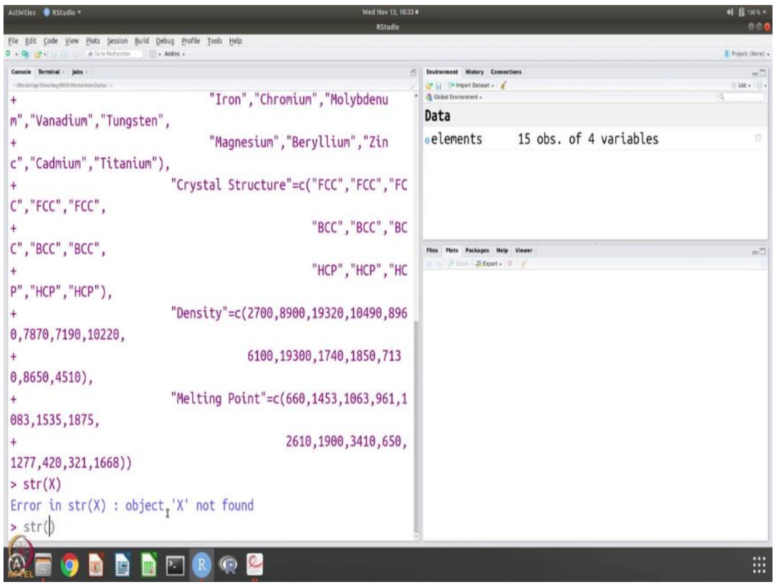
In this session, we want to learn how to input data not as a table but as a data frame and manipulate it. We are again going to use our elemental data on crystal structures, density and melting point as an example. The first step is to store the data as data frame. Here is the script that accomplishes this task.

```
elements <- data.frame("Element"=
c("Aluminium","Nickel","Gold","Silver","Copper",
"Iron","Chromium","Molybdenum","Vanadium","Tungsten",
"Magnesium","Beryllium","Zinc","Cadmium","Titanium"),
"Crystal Structure"=c("FCC","FCC","FCC","FCC","FCC",
                      "BCC","BCC","BCC","BCC","BCC",
                      "HCP","HCP","HCP","HCP","HCP"),
"Density"=c(2700,8900,19320,10490,8960,7870,7190,10220,
            6100,19300,1740,1850,7130,8650,4510),
"Melting Point"=c(660,1453,1063,961,1083,1535,1875,
                  2610,1900,3410,650,1277,420,321,1668))
```

As we can see, storing as data frame object is much more direct. Each column is input as a vector or list.

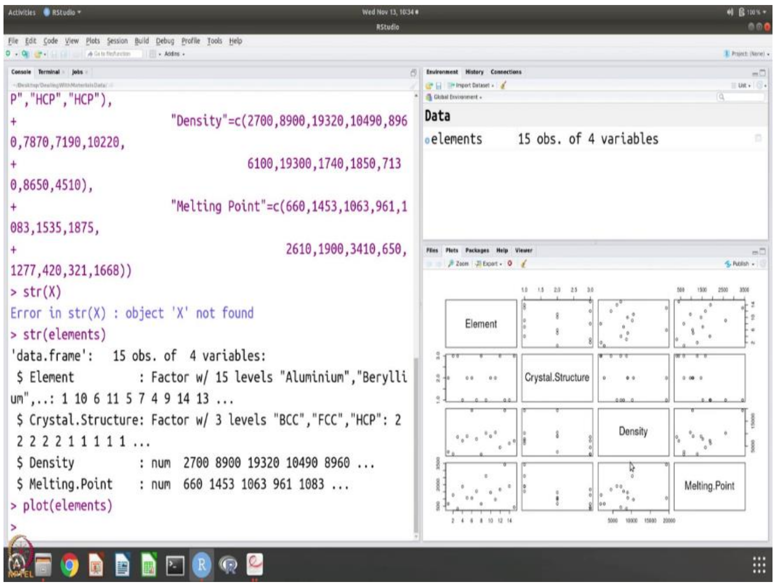
Let us now plot this data:

```
plot(elements)
```



```
+                    "Iron","Chromium","Molybdenu
m","Vanadium","Tungsten",
+                    "Magnesium","Beryllium","Zin
c","Cadmium","Titanium"),
+                "Crystal Structure"=c("FCC","FCC","FC
C","FCC","FCC",
+                                   "BCC","BCC","BC
C","BCC","BCC",
+                                   "HCP","HCP","HC
P","HCP","HCP"),
+                "Density"=c(2700,8900,19320,10490,896
0,7870,7190,10220,
+                          6100,19300,1740,1850,713
0,8650,4510),
+                "Melting Point"=c(660,1453,1063,961,1
083,1535,1875,
+                                   2610,1900,3410,650,
1277,420,321,1668))
> str(X)
Error in str(X) : object 'X' not found
> str()
```

```
Data
elements        15 obs. of 4 variables
```



```
P","HCP","HCP"),
+                    "Density"=c(2700,8900,19320,10490,896
0,7870,7190,10220,
+                              6100,19300,1740,1850,713
0,8650,4510),
+                    "Melting Point"=c(660,1453,1063,961,1
083,1535,1875,
+                                   2610,1900,3410,650,
1277,420,321,1668))
> str(X)
Error in str(X) : object 'X' not found
> str(elements)
'data.frame':	15 obs. of  4 variables:
 $ Element          : Factor w/ 15 levels "Aluminium","Berylli
um",..: 1 10 6 11 5 7 4 9 14 13 ...
 $ Crystal.Structure: Factor w/ 3 levels "BCC","FCC","HCP": 2
 2 2 2 2 1 1 1 1 1 ...
 $ Density          : num  2700 8900 19320 10490 8960 ...
 $ Melting.Point    : num  660 1453 1063 961 1083 ...
> plot(elements)
>
```

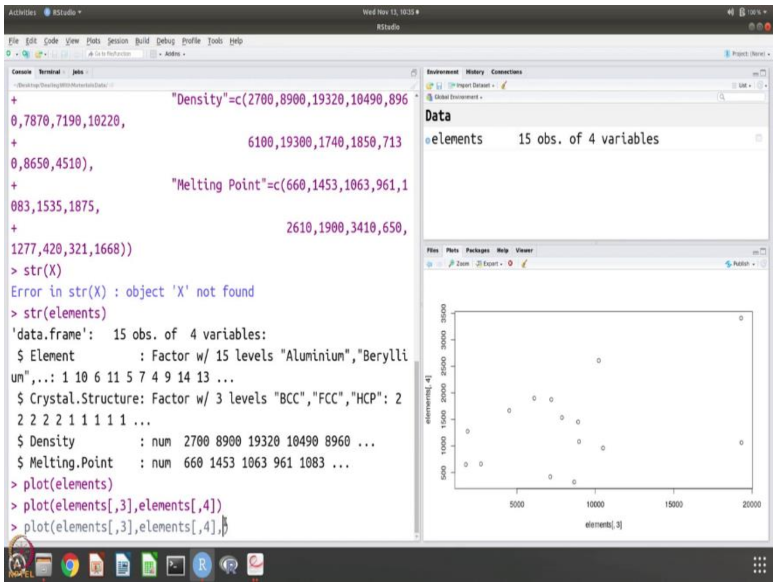
Data
elements	15 obs. of 4 variables

So, we have put the. So, now, as you can see in unlike the other case storing data frame is just this one command, we do not have to say data as matrix and then store as table or type convert, none of that is required, okay. So, this is the data frame and of course, you can, you can already see that elements is 15 observations of 4 variables the information is there. You can also use the structure, sorry structure of this. So, you can see it is a data frame it has 15 observations and 4 variables and the first column is element and it has 15 levels aluminum, beryllium, etc. And the next one is crystal structure. So, it has three levels bcc, fcc, hcp etc.

And then the density, so, it has all these densities and melting points. So, that is what this is. And you can directly say plot elements, when you do you can see that there are 4 variables, all the 4 variables and against each of these 4 variables. For example, element against crystal structure, element against density, element against melting point. And crystal structure, so crystal structure against element, against density, against melting point and density against element, crystal structure and melting point.

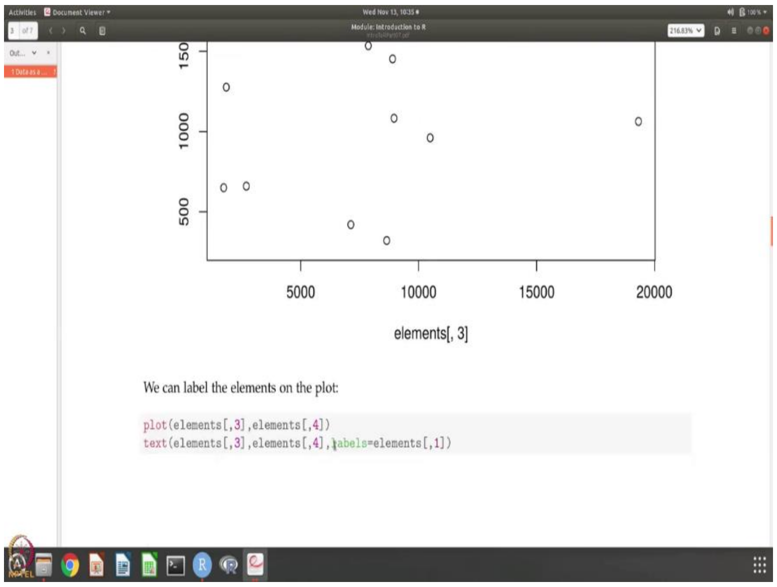
But the interest for us is basically this density versus melting point right. So, so we are going to plot this plot, you can do that you can get the individual columns picked up and plotted. So, but this gives you so if you have a data frame, you can generate a table of plots, which plots each of these variables against each of the other variables. So, if you have 4 levels, there are 16 plots here. So it is a nice way of getting a complete picture of the data in one go.

(Refer Slide Time: 6:25)



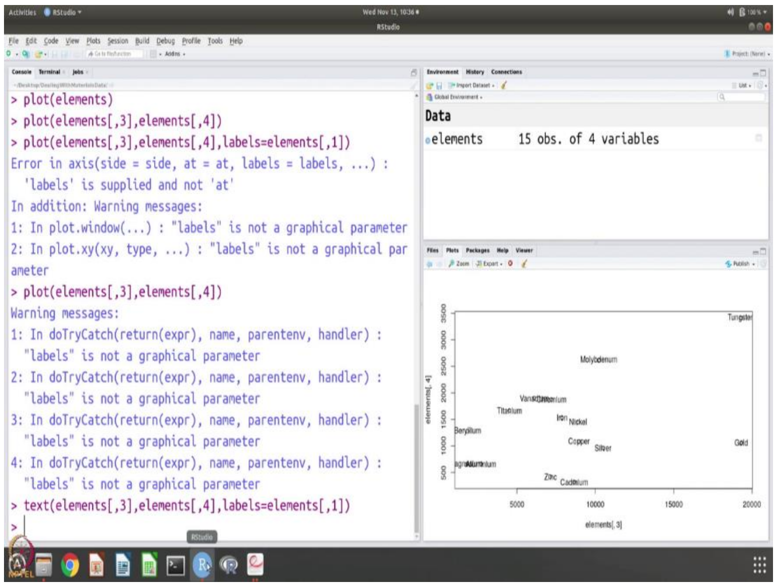
```
+                          "Density"=c(2700,8900,19320,10490,896
0,7870,7190,10220,
+                                      6100,19300,1740,1850,713
0,8650,4510),
+                          "Melting Point"=c(660,1453,1063,961,1
083,1535,1875,
+                                            2610,1900,3410,650,
1277,420,321,1668))
> str(X)
Error in str(X) : object 'X' not found
> str(elements)
'data.frame':   15 obs. of  4 variables:
 $ Element          : Factor w/ 15 levels "Aluminium","Berylli
um",..: 1 10 6 11 5 7 4 9 14 13 ...
 $ Crystal.Structure: Factor w/ 3 levels "BCC","FCC","HCP": 2
 2 2 2 2 1 1 1 1 1 ...
 $ Density          : num  2700 8900 19320 10490 8960 ...
 $ Melting.Point    : num  660 1453 1063 961 1083 ...
> plot(elements)
> plot(elements[,3],elements[,4])
> plot(elements[,3],elements[,4],
```

Data
elements   15 obs. of 4 variables



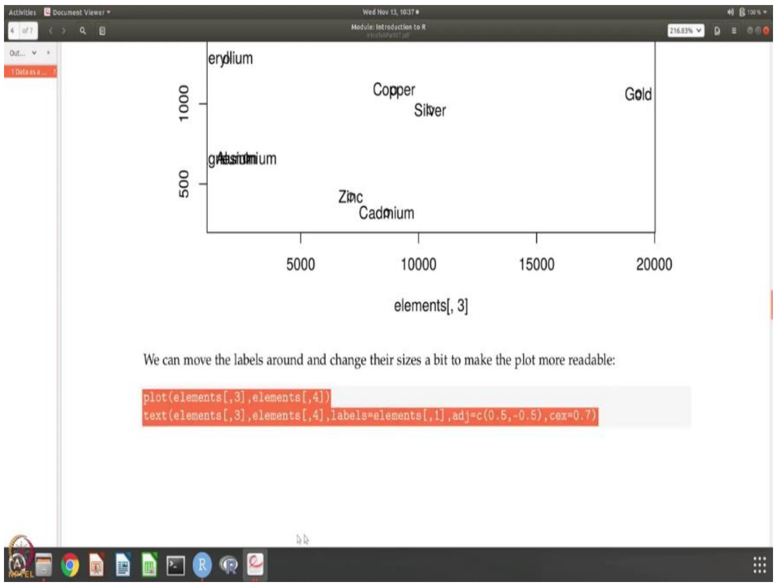
We can label the elements on the plot:

```
plot(elements[,3],elements[,4])
text(elements[,3],elements[,4],labels=elements[,1])
```



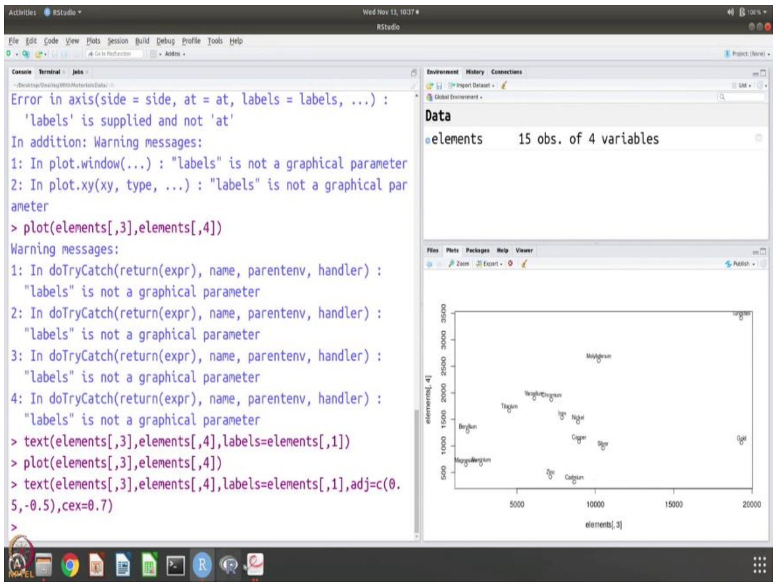
```
> plot(elements)
> plot(elements[,3],elements[,4])
> plot(elements[,3],elements[,4],labels=elements[,1])
Error in axis(side = side, at = at, labels = labels, ...) :
  'labels' is supplied and not 'at'
In addition: Warning messages:
1: In plot.window(...) : "labels" is not a graphical parameter
2: In plot.xy(xy, type, ...) : "labels" is not a graphical par
ameter
> plot(elements[,3],elements[,4])
Warning messages:
1: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
2: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
3: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
4: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
> text(elements[,3],elements[,4],labels=elements[,1])
>
```

Data
elements   15 obs. of 4 variables



We can move the labels around and change their sizes a bit to make the plot more readable:

```
plot(elements[,3],elements[,4])
text(elements[,3],elements[,4],labels=elements[,1],adj=c(0.5,-0.5),cex=0.7)
```



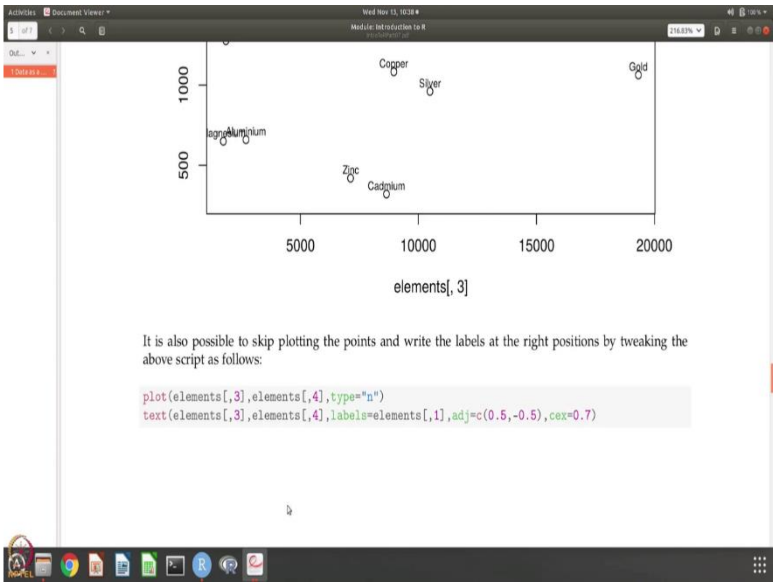
```
Error in axis(side = side, at = at, labels = labels, ...) :
  'labels' is supplied and not 'at'
In addition: Warning messages:
1: In plot.window(...) : "labels" is not a graphical parameter
2: In plot.xy(xy, type, ...) : "labels" is not a graphical par
ameter
> plot(elements[,3],elements[,4])
Warning messages:
1: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
2: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
3: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
4: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
> text(elements[,3],elements[,4],labels=elements[,1])
> plot(elements[,3],elements[,4])
> text(elements[,3],elements[,4],labels=elements[,1],adj=c(0.
5,-0.5),cex=0.7)
>
```

Once you have it. Of course, we want to go do more individual plotting. So, let us say I want to plot the density versus melting point, so now you have elements 3 and 4. So like I said, we want to name this and we want to label them. So, how do we do that? And that is what is shown here. So, the way to do is to say that labels, so these points should be labeled and they should be labeled according to the name given in the column 1 of the data, then the data frame the first column actually has the names aluminum, magnesium, etc.

So, we want that to be put next to these points, I get an error message, sorry, I have to plot and then I had to say this is the text that it has to right next to, so the text is to be written at these points. So, it chooses the points and then it labels them. So, now you can see that zinc, cadmium, silver, copper, nickel, iron, etc., it has written, but there is a problem because the data point is there and it is writing on top of it. So, you can shift the data points a little bit and that is what is done in the next one. So, we can say adjacent and you can give a vector which gives the position. And so, so let us do that. Let us take these two lines and so that is the command I am going to execute next.

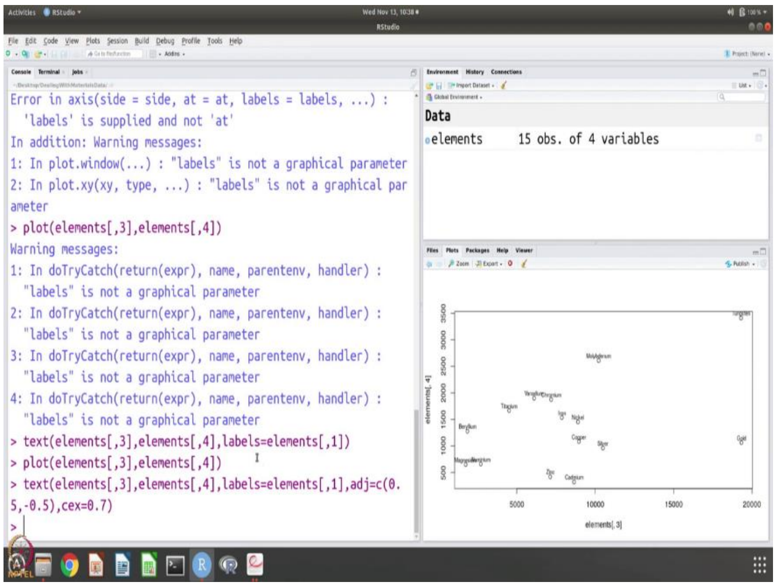
So, you can see the plot is the same 3 verses 4 and the text is at 3, 4. We are going to label and label according to element 1. But the position of the label and the size of the label is what is given here. So, as you can see, now it has shifted, so you can now read Magnesium, aluminum, beryllium, titanium, etc. And you can also see where the data points are.

(Refer Slide Time: 9:07)

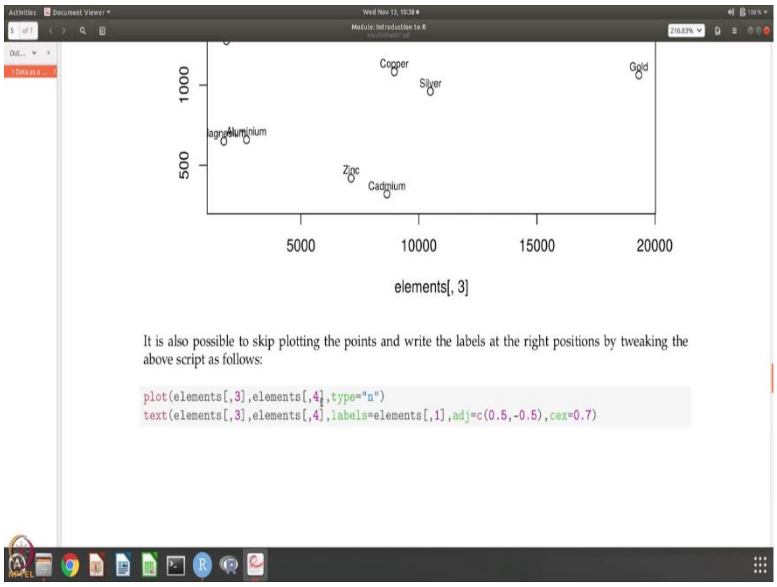


It is also possible to skip plotting the points and write the labels at the right positions by tweaking the above script as follows:

```
plot(elements[,3],elements[,4],type="n")
text(elements[,3],elements[,4],labels=elements[,1],adj=c(0.5,-0.5),cex=0.7)
```

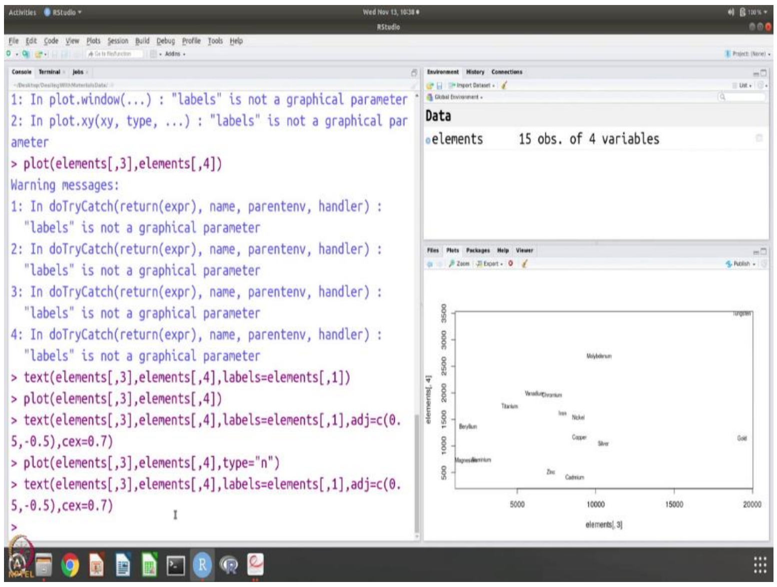


```
Error in axis(side = side, at = at, labels = labels, ...) :
  'labels' is supplied and not 'at'
In addition: Warning messages:
1: In plot.window(...) : "labels" is not a graphical parameter
2: In plot.xy(xy, type, ...) : "labels" is not a graphical par
ameter
> plot(elements[,3],elements[,4])
Warning messages:
1: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
2: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
3: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
4: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
> text(elements[,3],elements[,4],labels=elements[,1])
> plot(elements[,3],elements[,4])
> text(elements[,3],elements[,4],labels=elements[,1],adj=c(0.
5,-0.5),cex=0.7)
>
```

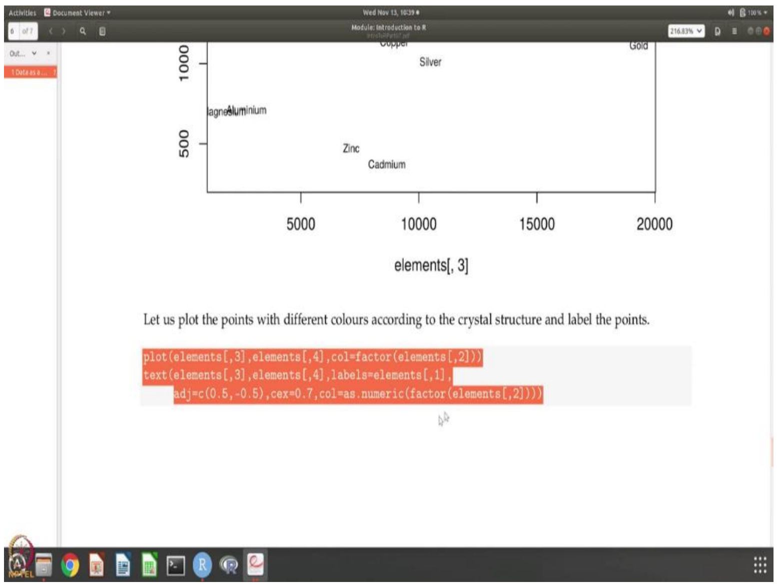


It is also possible to skip plotting the points and write the labels at the right positions by tweaking the above script as follows:

```
plot(elements[,3],elements[,4],type="n")
text(elements[,3],elements[,4],labels=elements[,1],adj=c(0.5,-0.5),cex=0.7)
```



```
1: In plot.window(...) : "labels" is not a graphical parameter
2: In plot.xy(xy, type, ...) : "labels" is not a graphical par
ameter
> plot(elements[,3],elements[,4])
Warning messages:
1: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
2: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
3: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
4: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
> text(elements[,3],elements[,4],labels=elements[,1])
> plot(elements[,3],elements[,4])
> text(elements[,3],elements[,4],labels=elements[,1],adj=c(0.
5,-0.5),cex=0.7)
> plot(elements[,3],elements[,4],type="n")
> text(elements[,3],elements[,4],labels=elements[,1],adj=c(0.
5,-0.5),cex=0.7)
>
```



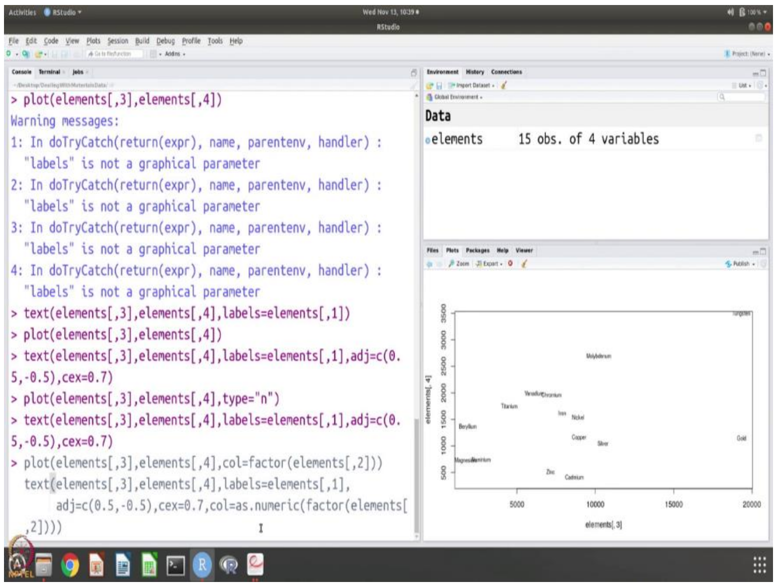
Let us plot the points with different colours according to the crystal structure and label the points.

```
plot(elements[,3],elements[,4],col=factor(elements[,2]))
text(elements[,3],elements[,4],labels=elements[,1],
     adj=c(0.5,-0.5),cex=0.7,col=as.numeric(factor(elements[,2])))
```

So, so this changes the size of the elements a little bit. And so, so you can also do other changes. For example, instead of this, so now let us go to the next step. In this next step what we are doing is slightly different. So, we have the data points and names. It is also possible for you to remove the data points and just write the names. So that is done, if you say that you do not want to put any points here, but just want to write names.

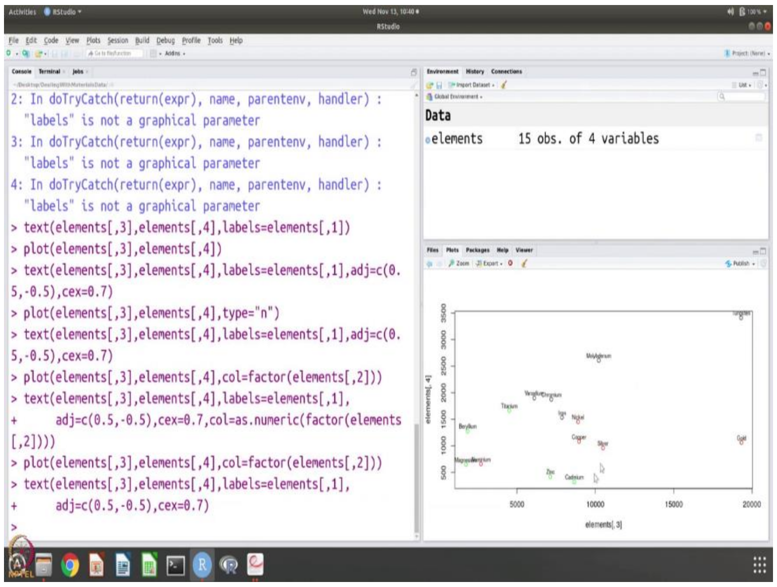
So that is also possible. And so you see just the names are written, the points are not given. So, it is possible to do either way. So, now, let us go to the next problem that we wanted to solve namely that we wanted to put them in different colors. And here is where that is done. So, let us go and see.

(Refer Slide Time: 10:28)



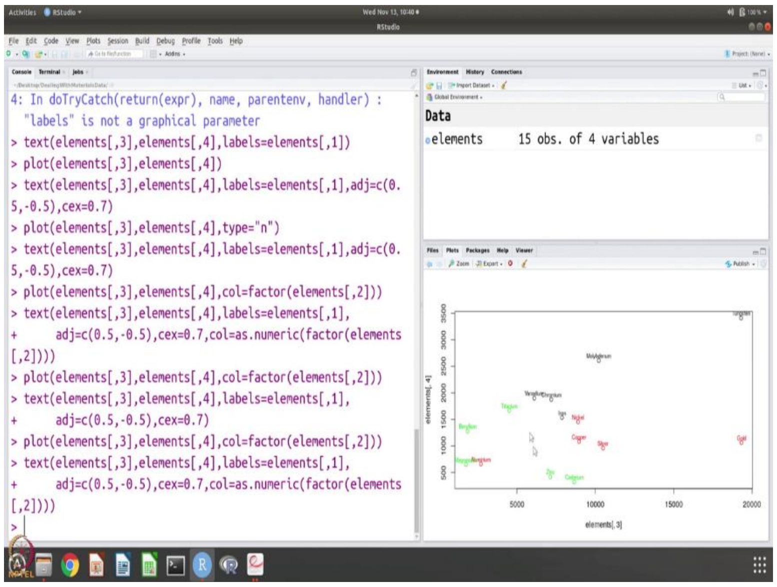
```
> plot(elements[,3],elements[,4])
Warning messages:
1: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
2: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
3: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
4: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
> text(elements[,3],elements[,4],labels=elements[,1])
> plot(elements[,3],elements[,4])
> text(elements[,3],elements[,4],labels=elements[,1],adj=c(0.
5,-0.5),cex=0.7)
> plot(elements[,3],elements[,4],type="n")
> text(elements[,3],elements[,4],labels=elements[,1],adj=c(0.
5,-0.5),cex=0.7)
> plot(elements[,3],elements[,4],col=factor(elements[,2]))
  text(elements[,3],elements[,4],labels=elements[,1],
       adj=c(0.5,-0.5),cex=0.7,col=as.numeric(factor(elements[
  ,2])))
```

Data
elements 15 obs. of 4 variables



```
2: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
3: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
4: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
> text(elements[,3],elements[,4],labels=elements[,1])
> plot(elements[,3],elements[,4])
> text(elements[,3],elements[,4],labels=elements[,1],adj=c(0.
5,-0.5),cex=0.7)
> plot(elements[,3],elements[,4],type="n")
> text(elements[,3],elements[,4],labels=elements[,1],adj=c(0.
5,-0.5),cex=0.7)
> plot(elements[,3],elements[,4],col=factor(elements[,2]))
> text(elements[,3],elements[,4],labels=elements[,1],
+       adj=c(0.5,-0.5),cex=0.7,col=as.numeric(factor(elements
[,2])))
> plot(elements[,3],elements[,4],col=factor(elements[,2]))
> text(elements[,3],elements[,4],labels=elements[,1],
+       adj=c(0.5,-0.5),cex=0.7)
>
```

Data
elements 15 obs. of 4 variables



```
4: In doTryCatch(return(expr), name, parentenv, handler) :
  "labels" is not a graphical parameter
> text(elements[,3],elements[,4],labels=elements[,1])
> plot(elements[,3],elements[,4])
> text(elements[,3],elements[,4],labels=elements[,1],adj=c(0.
5,-0.5),cex=0.7)
> plot(elements[,3],elements[,4],type="n")
> text(elements[,3],elements[,4],labels=elements[,1],adj=c(0.
5,-0.5),cex=0.7)
> plot(elements[,3],elements[,4],col=factor(elements[,2]))
> text(elements[,3],elements[,4],labels=elements[,1],
+       adj=c(0.5,-0.5),cex=0.7,col=as.numeric(factor(elements
[,2])))
> plot(elements[,3],elements[,4],col=factor(elements[,2]))
> text(elements[,3],elements[,4],labels=elements[,1],
+       adj=c(0.5,-0.5),cex=0.7)
> plot(elements[,3],elements[,4],col=factor(elements[,2]))
> text(elements[,3],elements[,4],labels=elements[,1],
+       adj=c(0.5,-0.5),cex=0.7,col=as.numeric(factor(elements
[,2])))
>
```

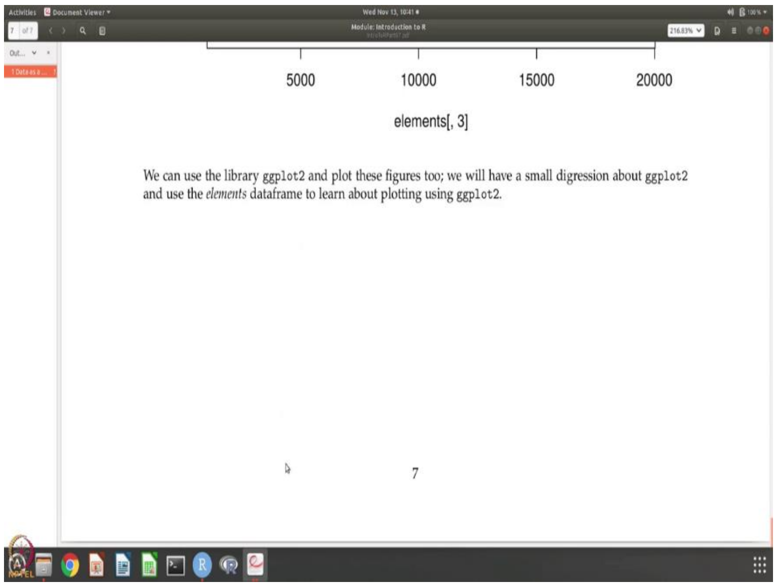
```
Data
elements      15 obs. of 4 variables
```

So, this is plotting element 3 versus element 4 and like we did earlier we are going to color and the color is according to what is there in column two, if it is bcc, hcp, fcc, etc, they will get different colors. And then of course, you can write a text and the text is the element name and that is going to be put. And the text is also to be colored according to the same the fcc, bcc, hcp, etc.

So, if you do not give this color, then you will get them all written black like this, but if you give this color that will also be colored according to the crystal structure. So, the plot looks nice. So we will do both. So, let us do this first. And so you can see, now the all fcc are in red the points as well as the names. And all hcp is in green. And all bcc is in black. If you do not do the second coloring if you do not do this, if you do not do this, let us say do not do this.

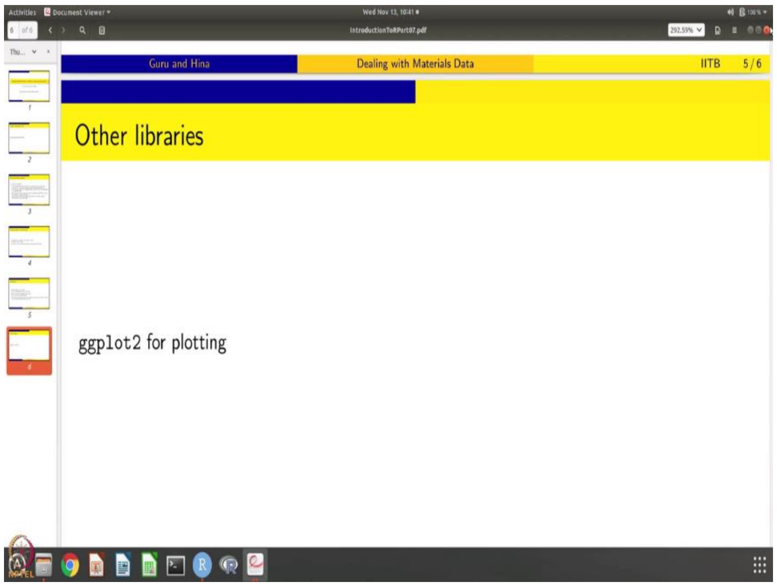
And then if you enter, then the colors the names are not written in the corresponding color. So, to make it colored, it looks nice. So, we are going to say that the color should again be according to the factor elements. And so in the text, also you can get the color so you get this.

(Refer Slide Time: 12:18)



We can use the library ggplot2 and plot these figures too; we will have a small digression about ggplot2 and use the *elements* dataframe to learn about plotting using ggplot2.

7



Other libraries

ggplot2 for plotting

So, now, the next step is to do this. So, and we want to use a ggplot to do the same thing. So I want to take a small digression here to tell about the plotting library called ggplot. And then we will use ggplot to do the same thing. Namely to generate density versus melting point, color the points differently and write the names of the elements and so on and so forth. So that is what we are going to do next. After we take a small digression into ggplot2. Thank you.