

### Accessing Column from data frame:

Accessing Column from data frame is quite easy. Once you write any data frame and after that if you put “dollar sign (\$)” then you would see all the columns whatever it contains and you can see whatever is there in that column.

Now, there is another way to do it, if you use “third bracket[]” and if you use row, column as I discussed earlier.

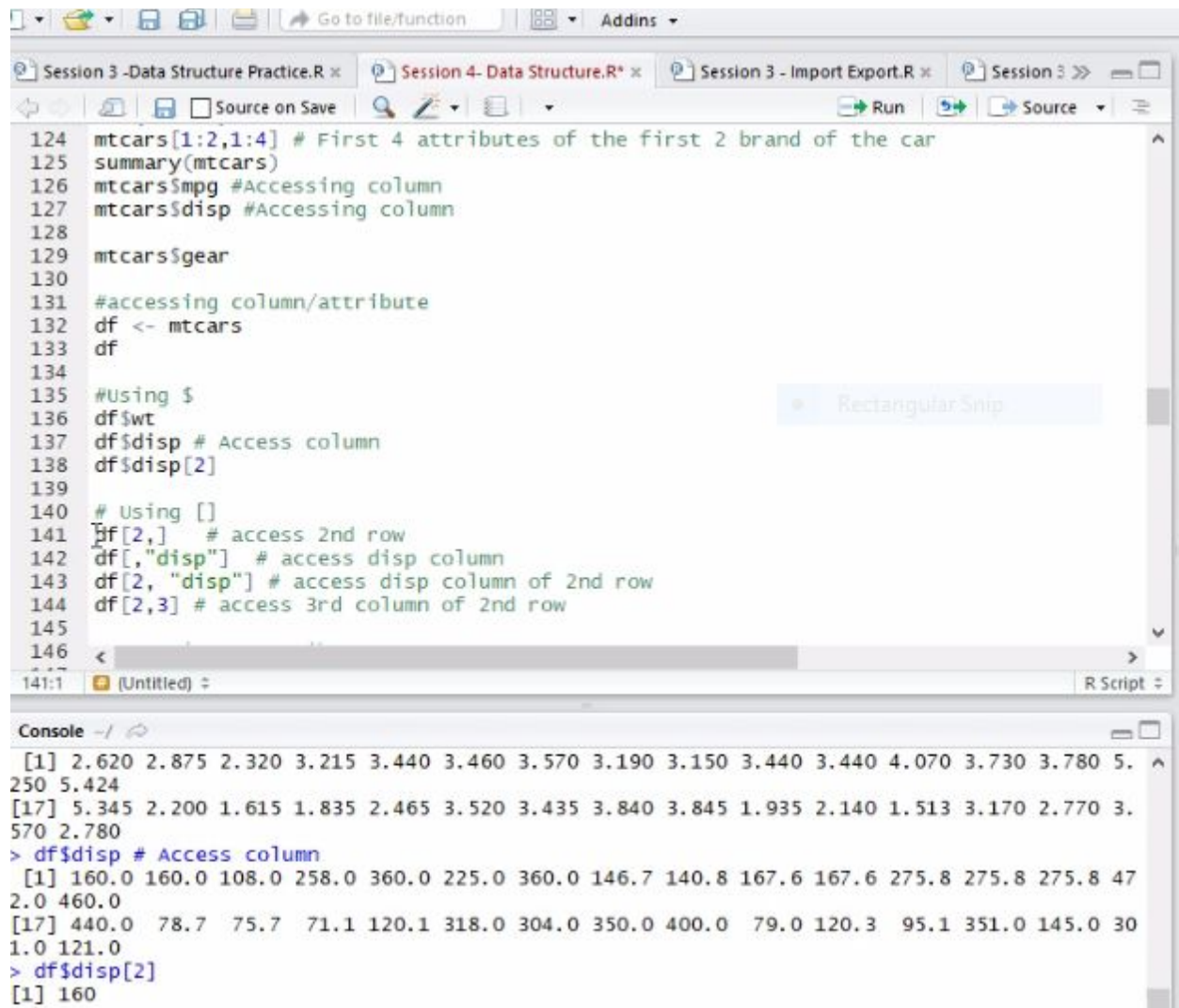
Similarly, you can name the column and you can see the column as well. But it has a problem if you have more than one column then how do you do that? In that case, you use combine function and give the number of all those columns and you can see those columns as well.

### Accessing Row:

Similarly, Accessing Row is also easy. As said use third bracket and between third bracket row, column.

For an example: `df[2,]`

Here row value is 2 and all column value is NULL. The function is used for accessing the second row.



```
124 mtcars[1:2,1:4] # First 4 attributes of the first 2 brand of the car
125 summary(mtcars)
126 mtcars$mpg #Accessing column
127 mtcars$disp #Accessing column
128
129 mtcars$gear
130
131 #accessing column/attribute
132 df <- mtcars
133 df
134
135 #Using $
136 df$wt
137 df$disp # Access column
138 df$disp[2]
139
140 # Using []
141 df[2,] # access 2nd row
142 df[, "disp"] # access disp column
143 df[2, "disp"] # access disp column of 2nd row
144 df[2,3] # access 3rd column of 2nd row
145
146
```

Console

```
[1] 2.620 2.875 2.320 3.215 3.440 3.460 3.570 3.190 3.150 3.440 3.440 4.070 3.730 3.780 5.
250 5.424
[17] 5.345 2.200 1.615 1.835 2.465 3.520 3.435 3.840 3.845 1.935 2.140 1.513 3.170 2.770 3.
570 2.780
> df$disp # Access column
[1] 160.0 160.0 108.0 258.0 360.0 225.0 360.0 146.7 140.8 167.6 167.6 275.8 275.8 275.8 47
2.0 460.0
[17] 440.0 78.7 75.7 71.1 120.1 318.0 304.0 350.0 400.0 79.0 120.3 95.1 351.0 145.0 30
1.0 121.0
> df$disp[2]
[1] 160
```

If you want to drop a column then you don't want the column to be included in your data frame. You simply put (-sign) before that number. If you say that you don't want to see drop third column which is actually displacement column.

```
df[,-c(2,3)]
```

Similarly, if you drop second and third then it would actually drop displacement and cylinder.

The fourth one is a subset. So, what if you don't want to see all the rows of that column. You want to see only those observation or those car brands where the cylinder value is more than 6 or the horsepower is more than 50.

```
car 2<-subset (df,hp>50)
```

Similarly, for horsepower also you can see the result.

If you have two data frame, so you have data frame one and data frame two now you have combined them by row. Suppose, in the first data frame you have twenty car brands and second data frame you have twelve car brands. Now you want to combine this two `rbind ()` function.

## Data Frame

### **rbind()**

```
> df1 <- df[1:20,]  
> str(df1)  
> df2 <- df[21:32,]  
> str(df2)  
  
> df_full <- rbind(df1,df2)  
> str(df_full)
```

Combine dataframe  
by Row

### **cbind()**

```
> df3 <- df$mpg  
> str(df3)  
> df4 <- df$cyl  
> str(df4)  
> df_full <- cbind(df3,df4)  
> str(df_full)
```

Combine dataframe  
by Column

Likewise, if you have two columns. Suppose all the 32 car brands you have the mileage which is a column and you have another one may be a cylinder. So you want to combine this column that also you can do using cbind() function.

### **Factor:**

In a data frame, character vectors are automatically converted into factors, and the number of levels can be determined as the number of different values in such a vector. You can create a data frame using more than one data types. It can be a mixture of numeric, with a character with logical all those things.

## Factor

In a data frame, character vectors are automatically converted into factors, and the number of levels can be determined as the number of different values in such a vector.

# By default char is changed to factor in data frame

```
name <- c("joe", "john", "nancy")
sex <- c("M", "M", "F")
age <- c(27, 26, 26)
```

```
df <- data.frame(name, sex, age)
df
class(df)
class(df$name)
class(df$sex)
```

```
> class(name)
[1] "character"
> df
  name sex age
1  joe   M  27
2 john   M  26
3 nancy  F  26
> class(df)
[1] "data.frame"
> class(df$name)
[1] "factor"
> class(df$sex)
[1] "factor"
> class(df$age)
[1] "numeric"
```

# Change numeric to factor class

```
mtcars$cyl = as.factor(mtcars$cyl)
mtcars$am = as.factor(mtcars$am)
```

# Change the name of the factors using level()

```
levels(factor_gender_vector) <- c("F", "M")
```



Activate Windows

Similarly, if you see the showing data frame which has three vectors like name, age and gender and you have created a new data frame using data.frame() function. If you see the class of it, you would get it as data.frame. But surprisingly if you see the class of name this data frame dollar could give all those names. If you see the class of it you would be surprised this is now not a character variable rather it's now a factor. So it is another new data type.