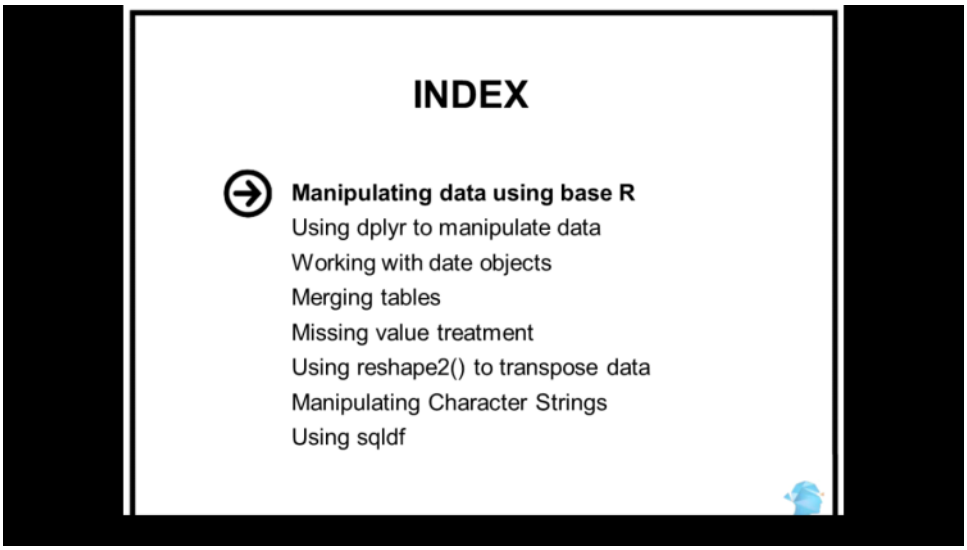
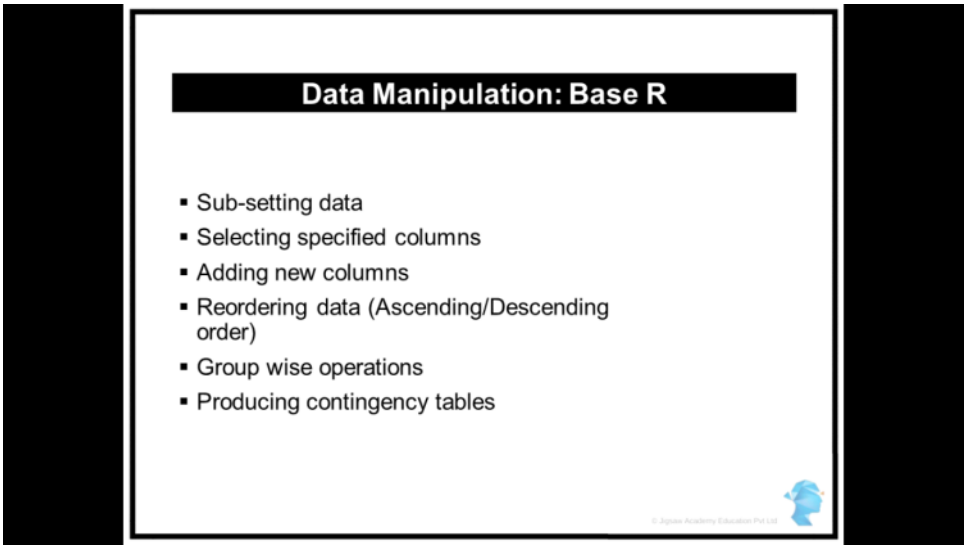
Manipulating Data with R

Tuesday, August 23, 2016 4:41 PM

Outline:





To access one specific cells in a data frame we use the following command --> df[1,3]
To access many specific cells in a data frame we can use the following command --> df[c(1,4,6,198),c(4,5,9)]
We can also access using a specific column name --> df[c(1:5),"brand"]

Lets look at logical subsetting of data i..e
If we want to get a subset of data, using a column filter :
Find all rows of data where band column has value as "Tropicana" --> df[df$Brand == 'Tropicana',]

Hence the condition is in the rows part of the DF in the code.

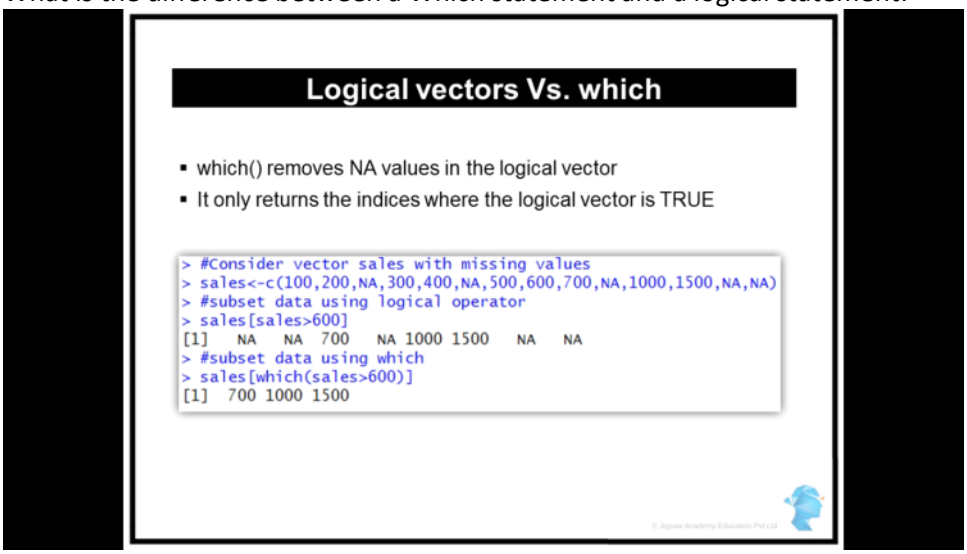
If we want to use multiple filters , we can do that by using the OR operator of R i.e. |
Df[df$brand=='Tropicana'|df$brand=='minutemaid',]

We cana lso combine data output based on AND operators ---> i.e. &
Df[df$brand=='Tropicana' & df$feat==0,]

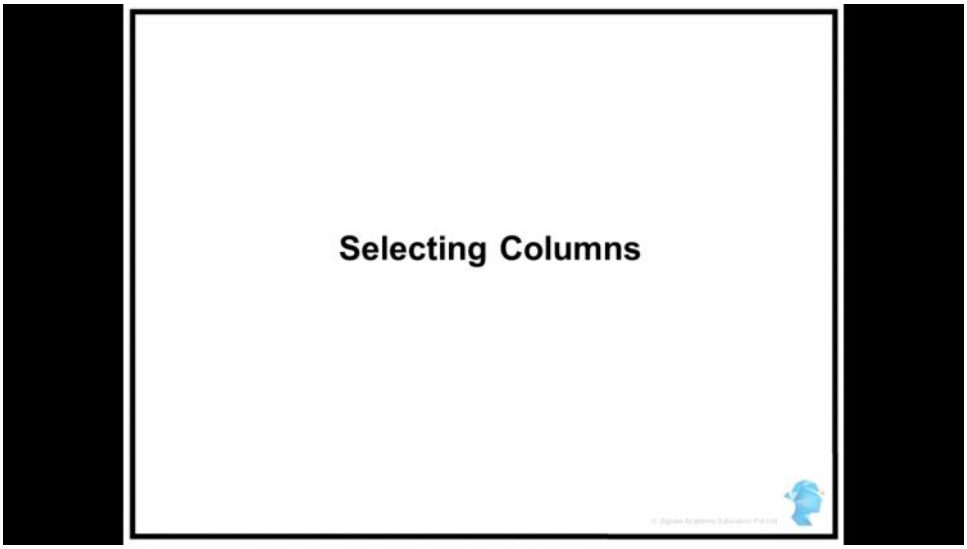
We can use these combinations to subset the data.

Another way to subbset the data is using the which statement: this will give row indices
Index<-which(df$brand=='Tropicana')

What is the difference between a Which statement and a logical statement:

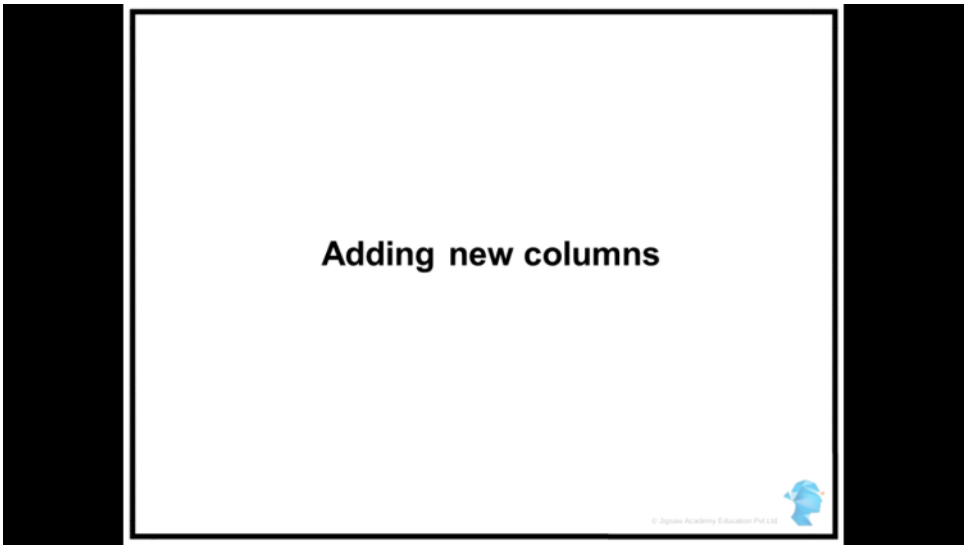


Logical statements will include the "NA" values but which will only return the values that are true for the condition



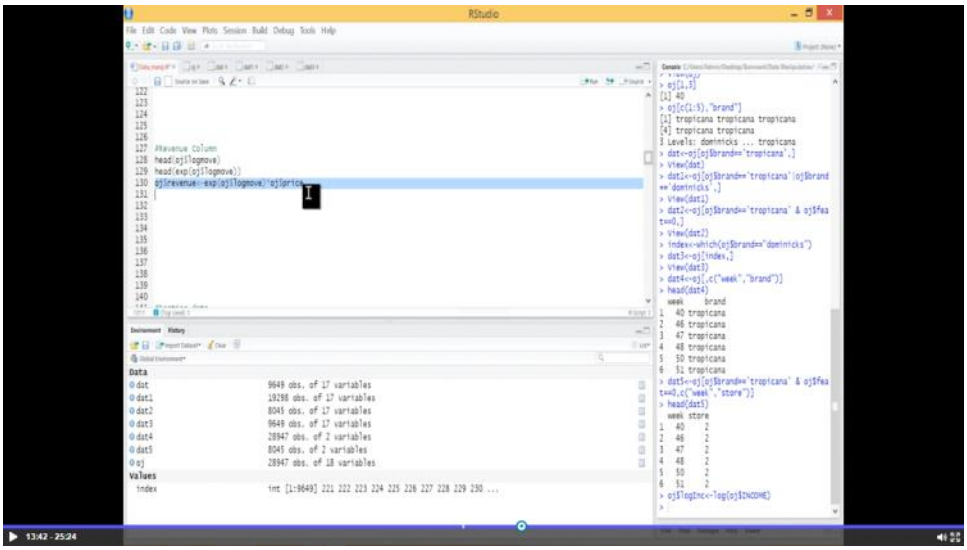
To select specific columns, we can mention either the column index or column name in our query :
Df[, c("brand", "feat")]

To subset based on logical conditions we can do the following :
Df[df$brand=="tropicana" & df$feat==0, c("week","store")]

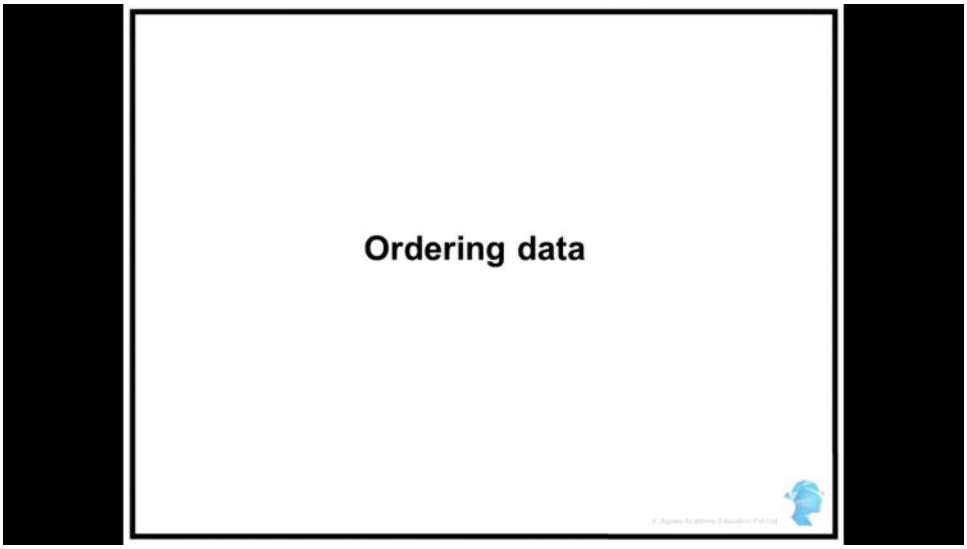


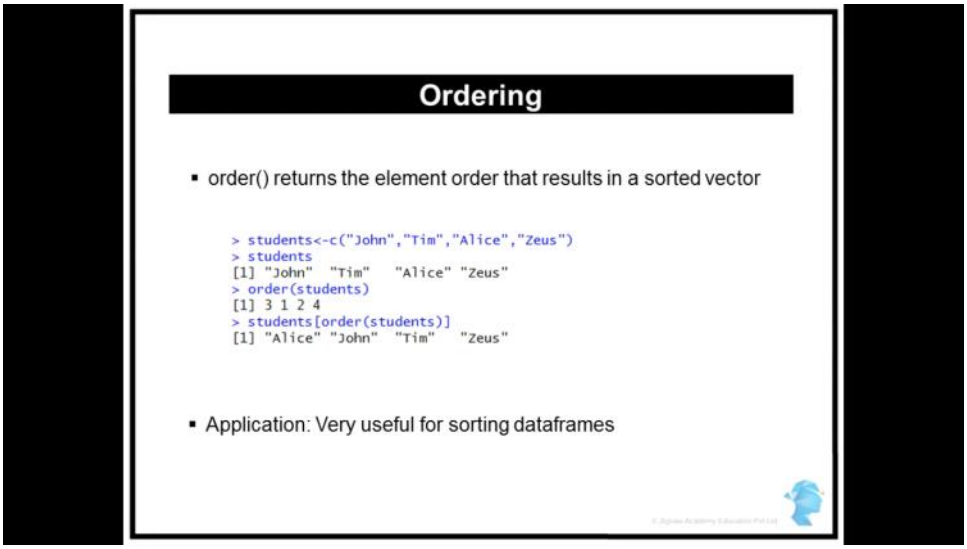
To add a new column to an existing data frame we can use this:
Df$logInc<-log(df$income)

We can store logical operations in new columns as follows:
Df$income<-exp(df$logmove)*df$price

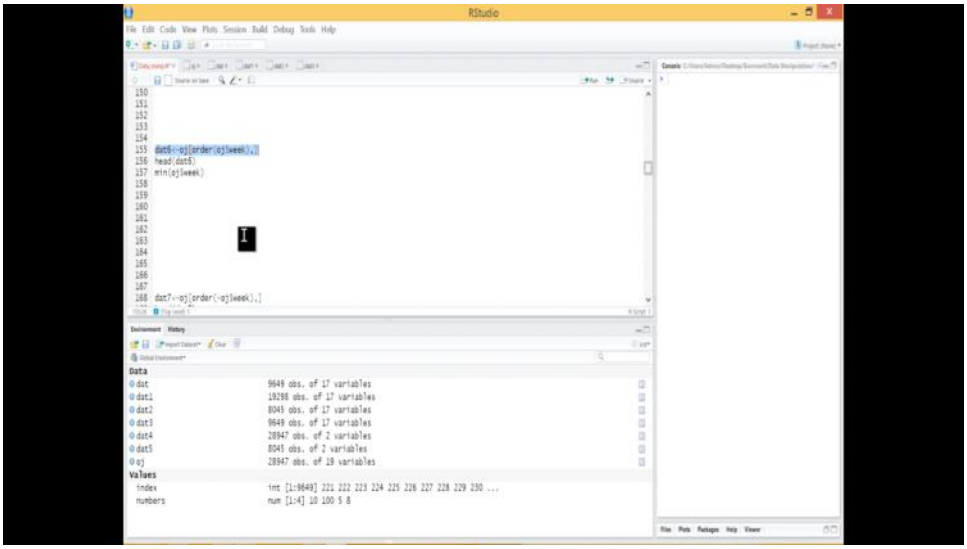


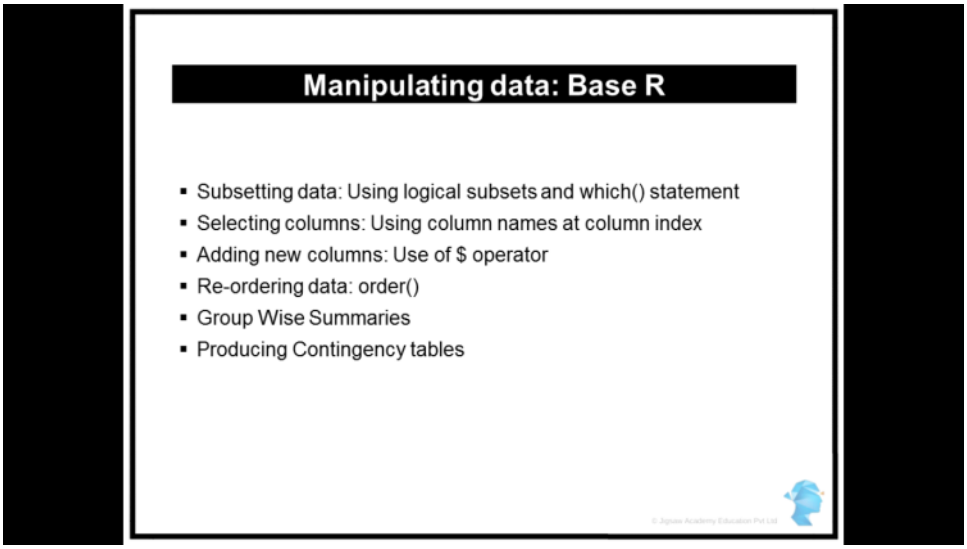
New is , how to sort the data:



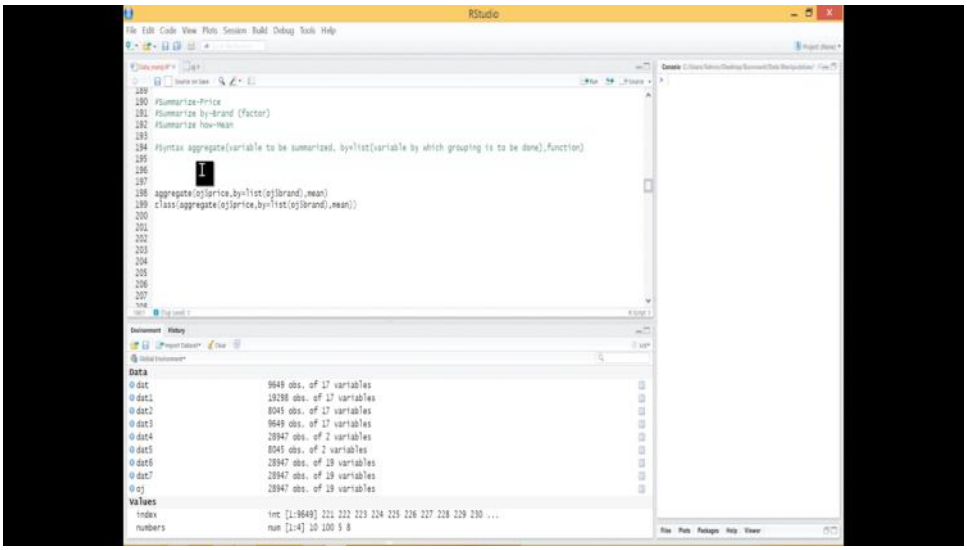


We can use the base R function called : order
Just using order(students) --> this will give the indices
Using --> students[order(students)] --> this will give the dataset
If we want to sort our data frame based on week numbers :
df[order(df$week),]
To reverse order the dataframe --> df[order(-df$week),]





The next function is group by summary : unique
Df[(unique(df$brand)]

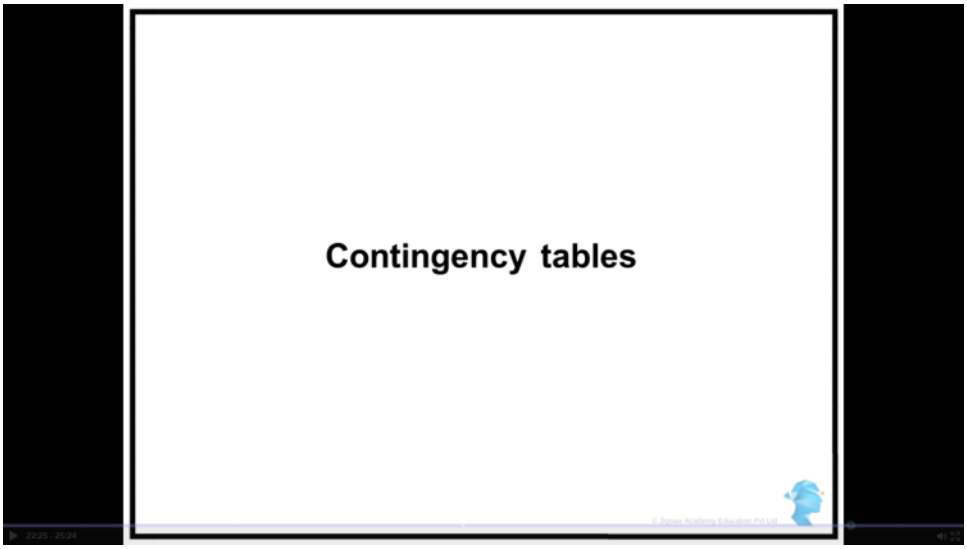


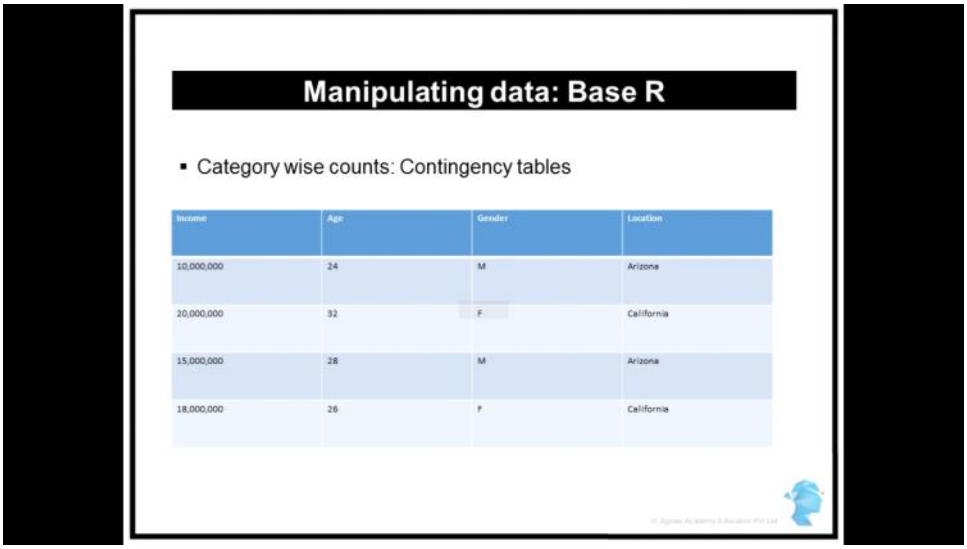
Syntax for aggregation:
Aggregate(variable to be summarized, by=list(variables by which grouping is to be done), function)

Aggregate(df$price, by=list(df$brand), mean)

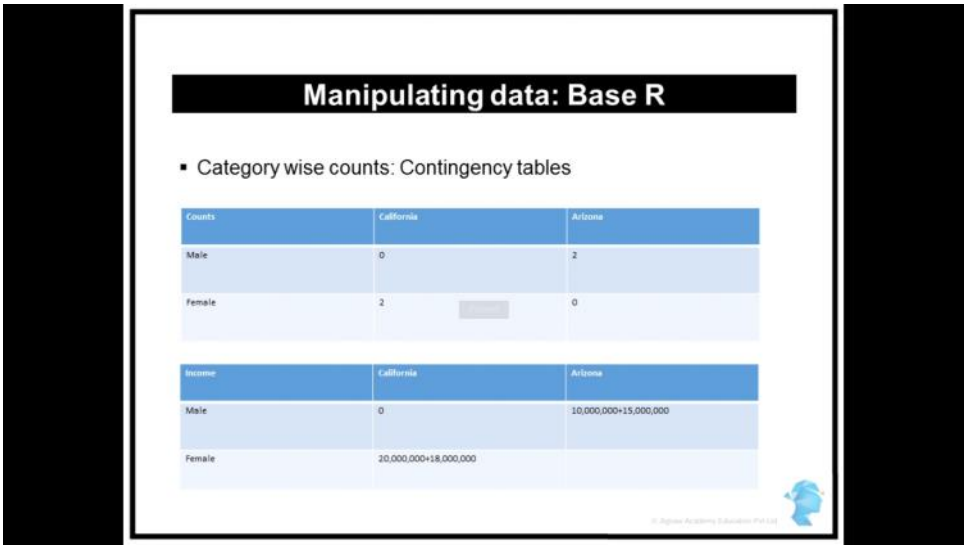
This is always of the class data frame

We can perform the same functionality by using tapply
Tapply(df$price, df$brand, mean)
In this case the class of the object is an array

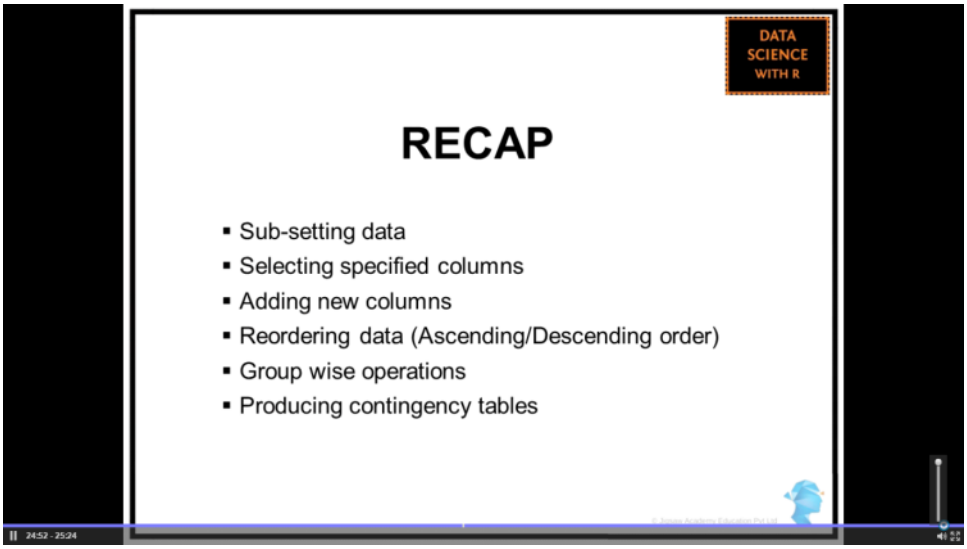


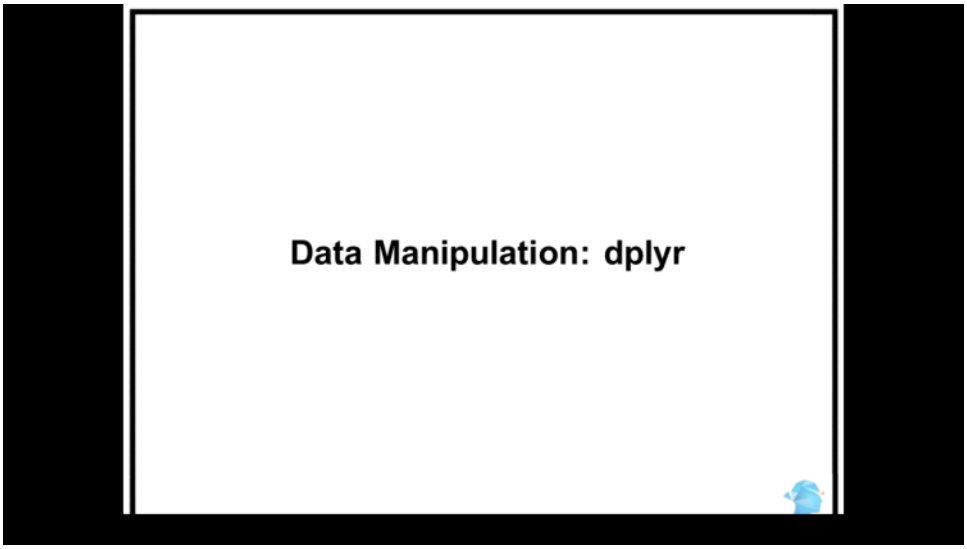


Contingency tables are those that have categorical values



Cross tabulation , use the table command --> table(df$brand, df$feat)





Manipulating data: dplyr

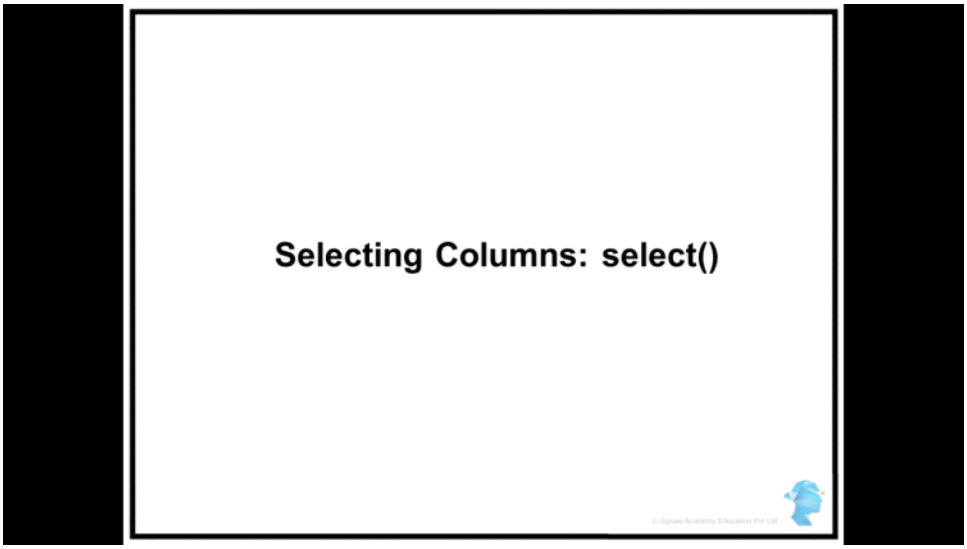
- dplyr: Whats and Whys
- Sub-setting data using filter()
- Selecting columns using select()
- Adding new columns using mutate()
- Ordering data using arrange()
- Summarizing using summarize() and group_by()
- Using functional pipelines to do more than one manipulation task

Manipulating data: dplyr

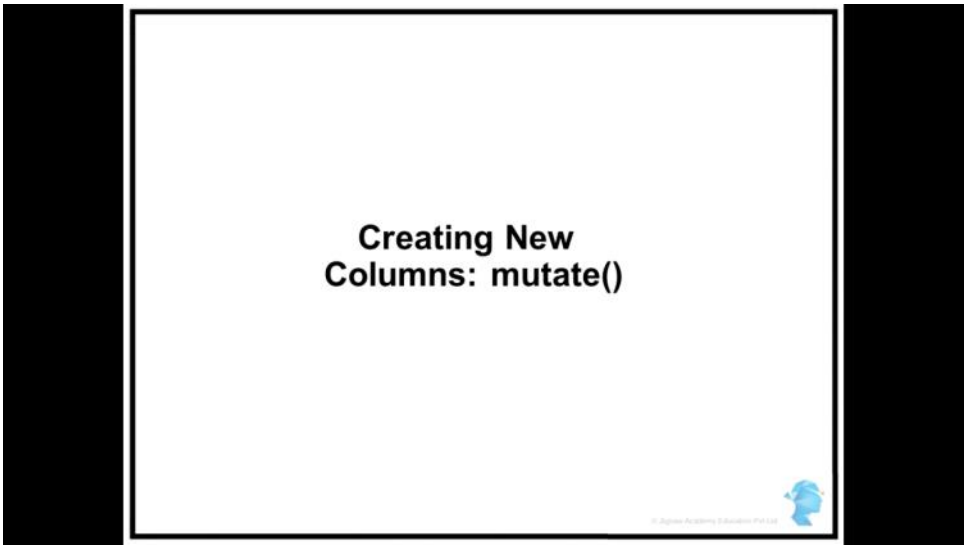
- Base R: Good for Medium sized data sets, Awkward Syntax
- dplyr: Faster and elegant syntax
- dplyr: Dataframes
- install.packages("dplyr")
- library(dplyr)

Sub-setting: filter()

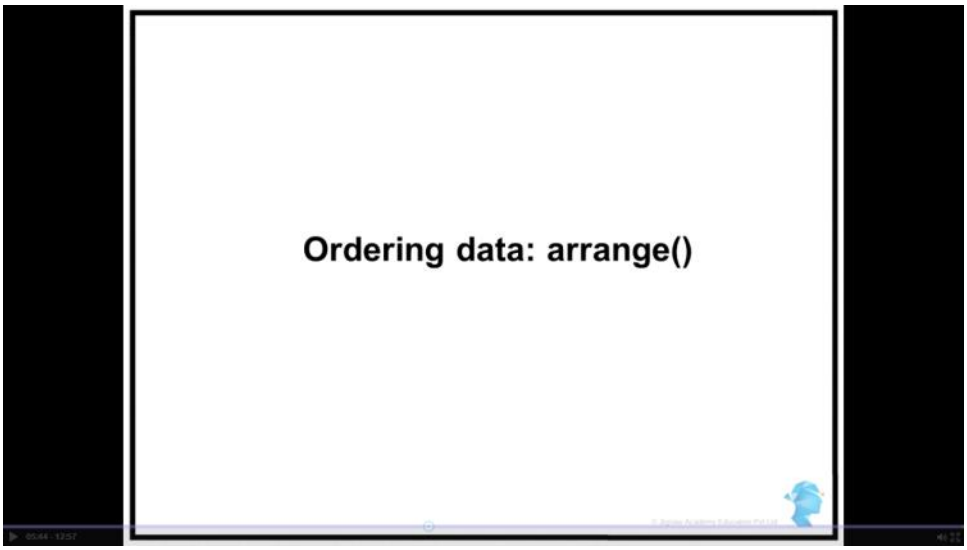

```
Data<-filter(df, brand == "tropicana"|brand=="minutemaid")
```



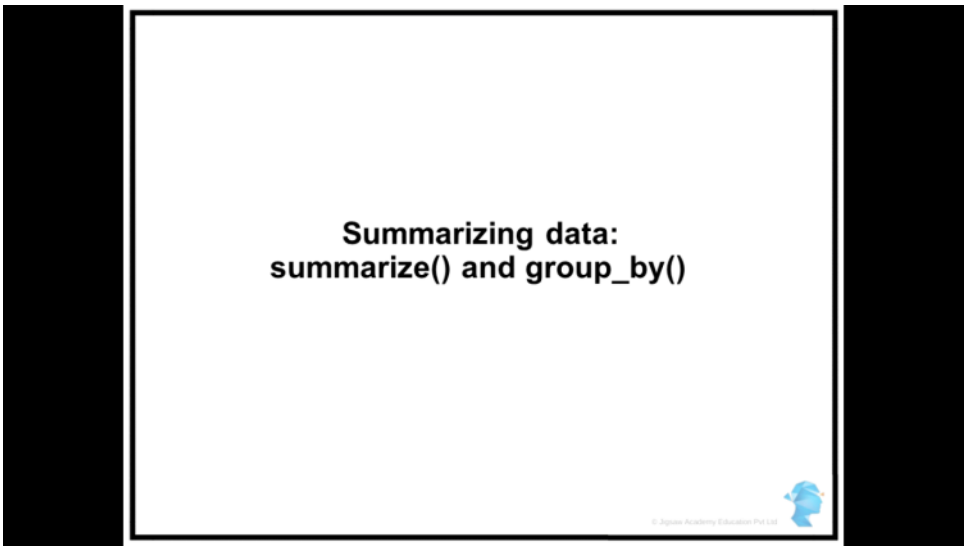
data<-select(df, brand, income, feat)
We can use the -ve mark if we want to exclude culumns from the original set into the new dataset
Data<-select(df, -brand, -feat)



Data<-mutate(df, logincome=log(income))

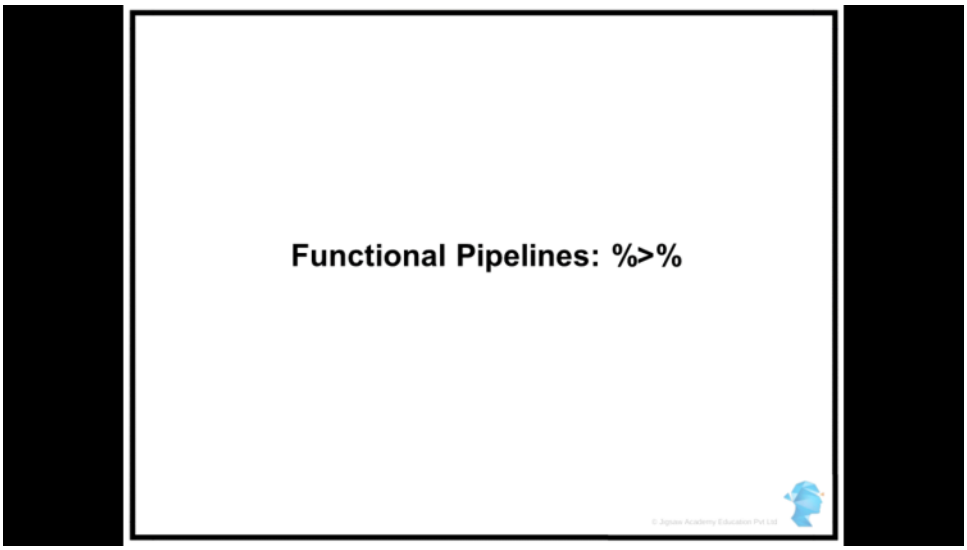


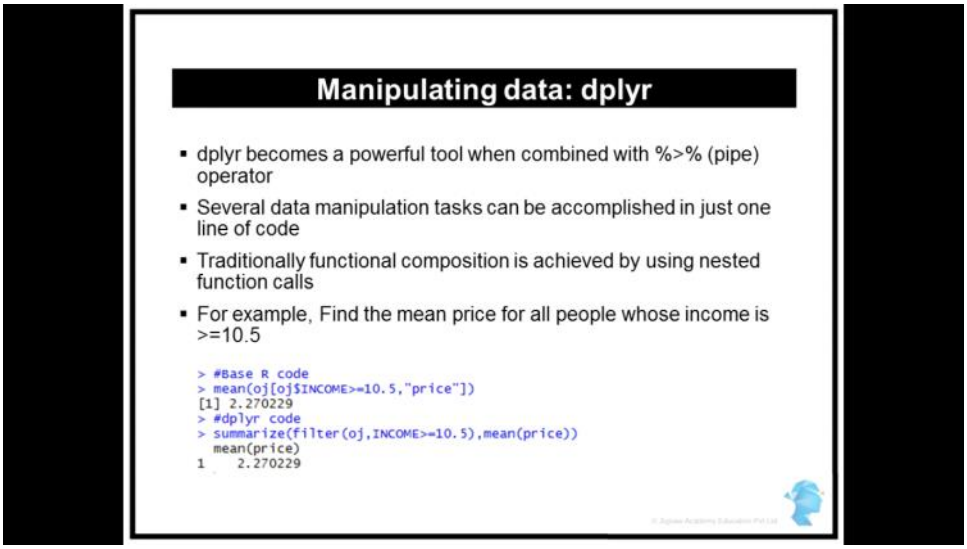
Data<-arrange(df,income)
Data<-arrange(df,desc(income))

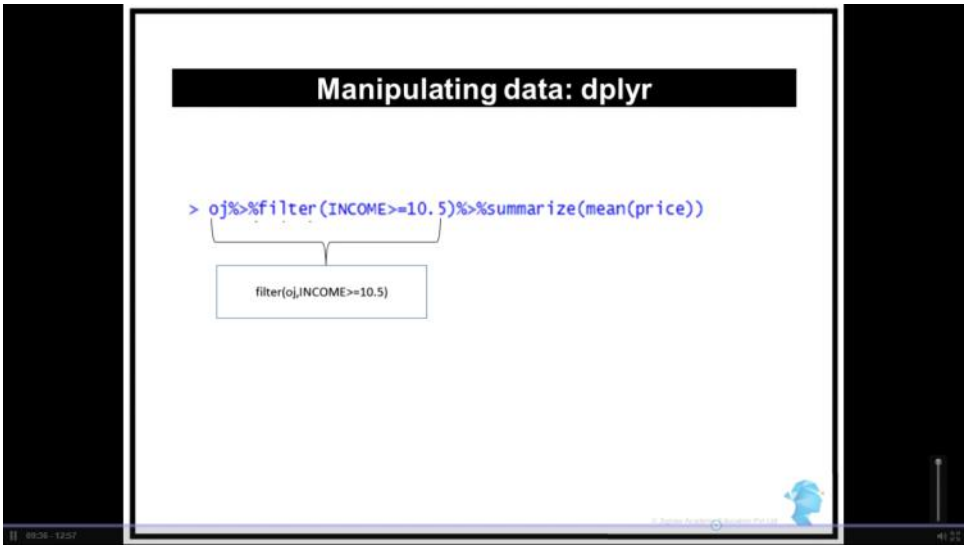


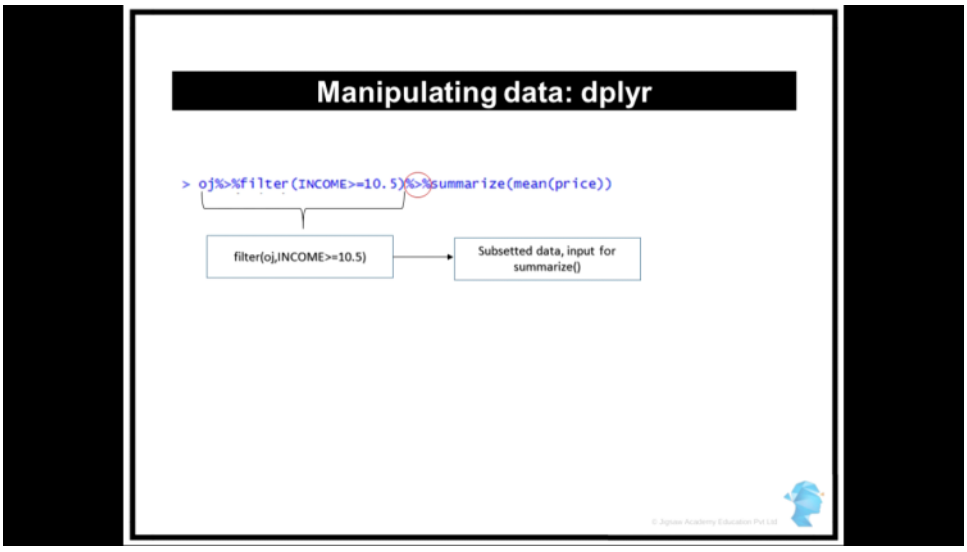
Gr_brnad<-group_by(df,brand)
Summarize<-(gr_band, mean(income), sd(income))

For the summarize function , the first attribute is the input and the rest are the outputs(function that you want to carry on the input).

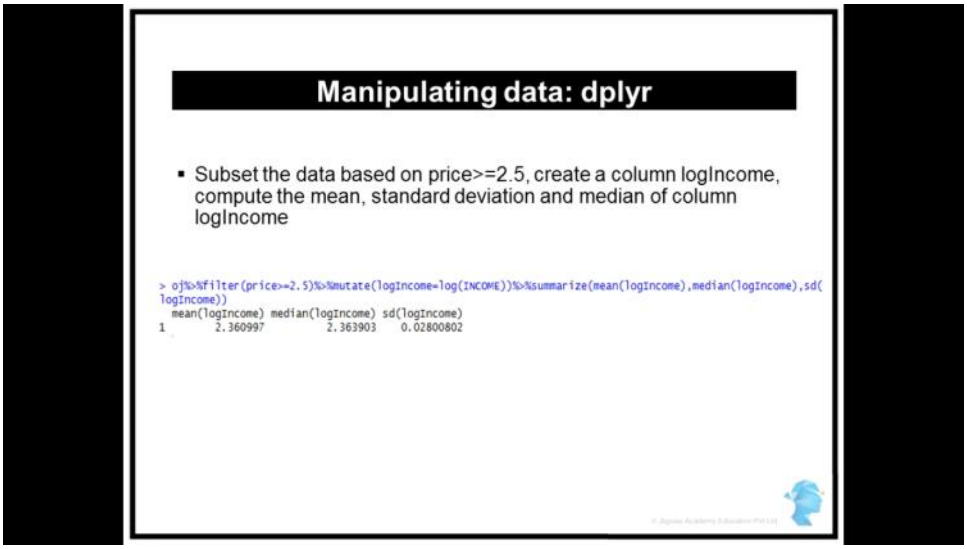


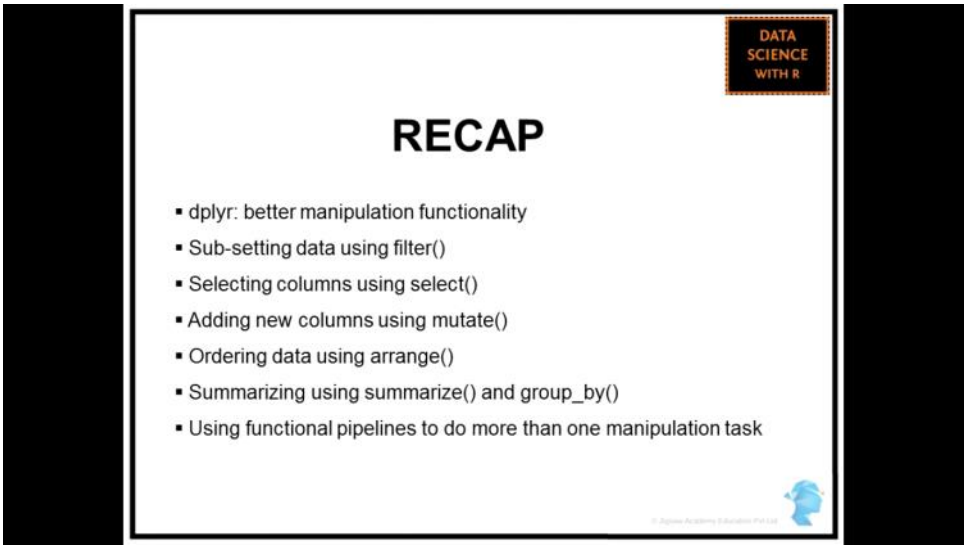




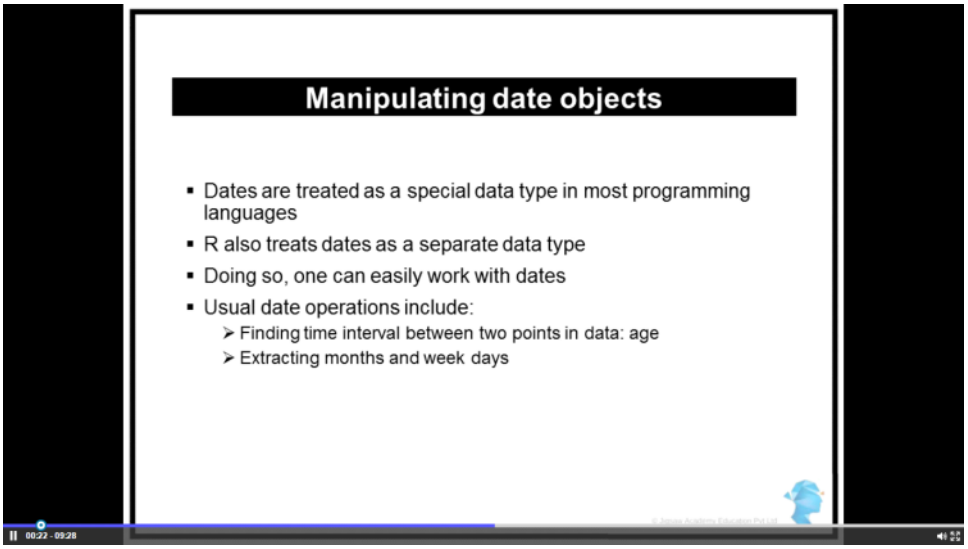


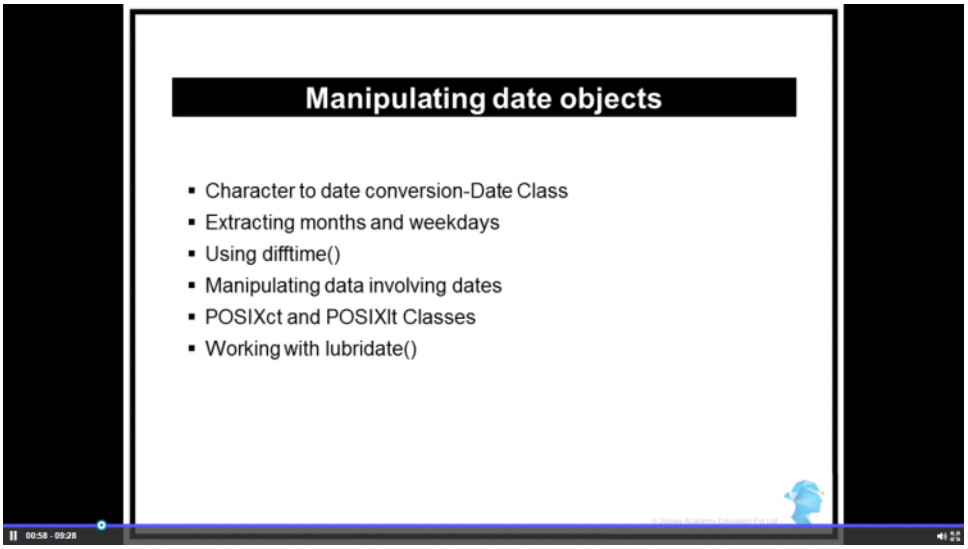
<Make sure that the data type is compatible while using pipes i.e. %>%

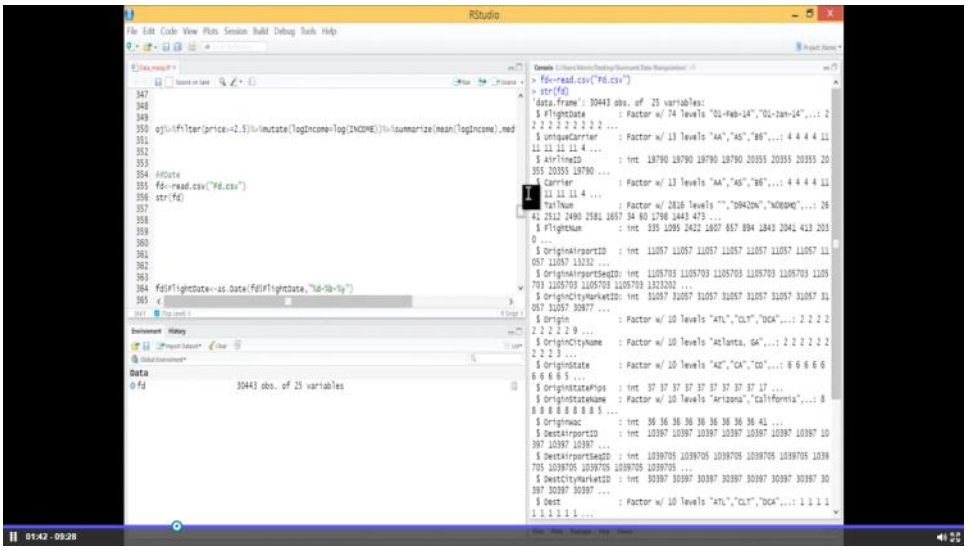




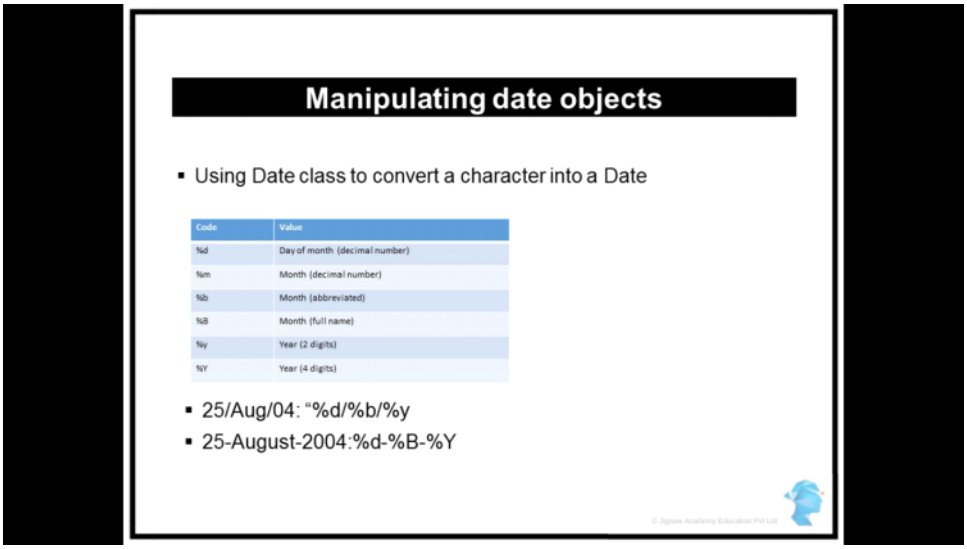
Working with Date Objects:





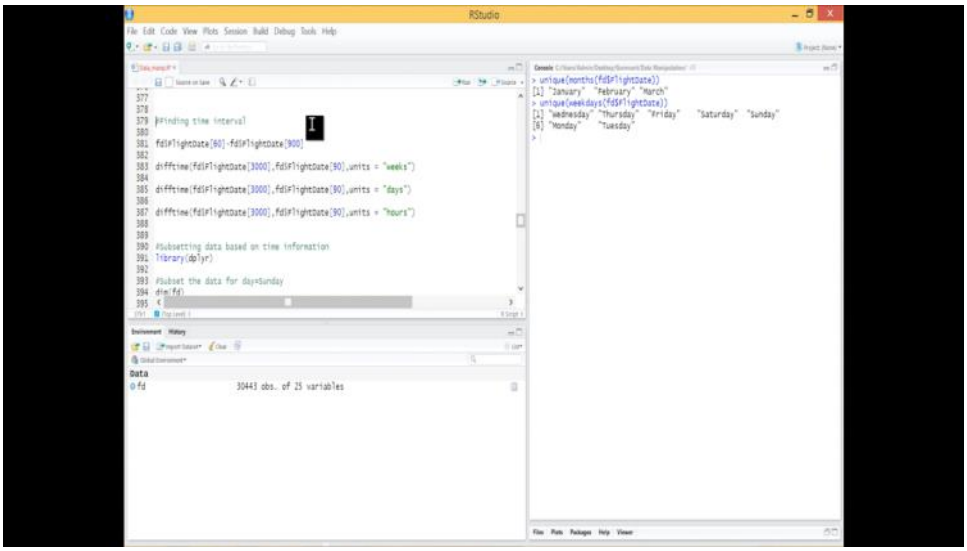


Here are the different data formats for date:



To extract the month from the dates use --> months(fd$flightdate)
To extract unique months --> unique(months(fd$flightdate))
Similarly to get weekdays --> unique(weekdays(fd$flightdate))

To find the time interval between two dates:



The difftime function can be used to find the timedifferences , the units can be in days, minutes, weeks etc

Difftime(fd$Flightdate[3000],fd$flightdate[60], units="weeks")

Manipulating date objects

- Manipulating data involving dates
- Sub-setting data: All rows when the day is Sunday

```
> library(dplyr)
> #Subset the data for day=Sunday
> dim(fd)
[1] 30443    25
> fd_s<-fd%>%filter(weekdays(FlightDate)=="Sunday")
> dim(fd_s)
[1] 4015   25
```

Using DPLYR for the similar functionalities with dates :

Manipulating date objects

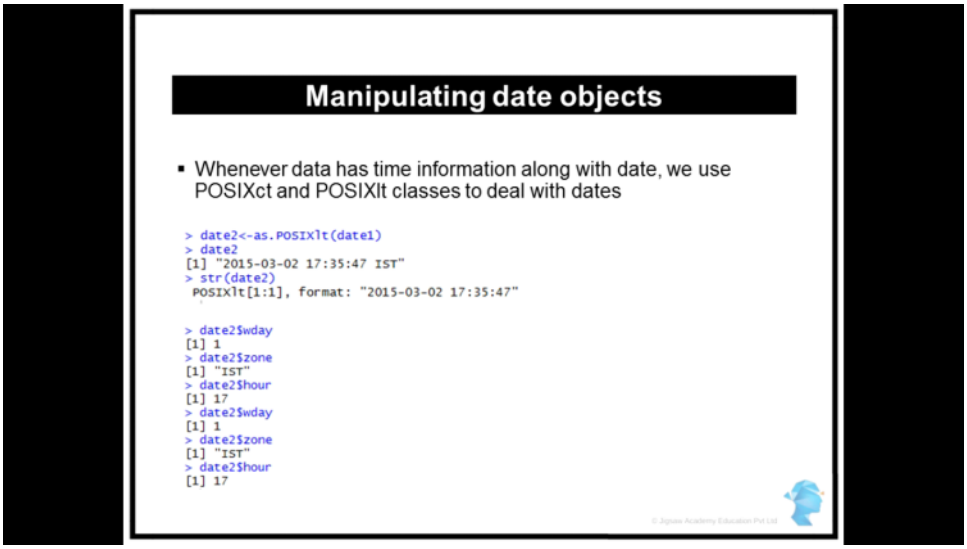
- Manipulating data involving dates
- Find the number of flights on Sundays for destination Atlanta

```
> #Find the number of flights on Sundays for destination Atlanta
> fd%>%filter(weekdays(FlightDate)=="Sunday",DestCityName=="Atlanta, GA")%>%nrow()
[1] 683
```

Manipulating date objects

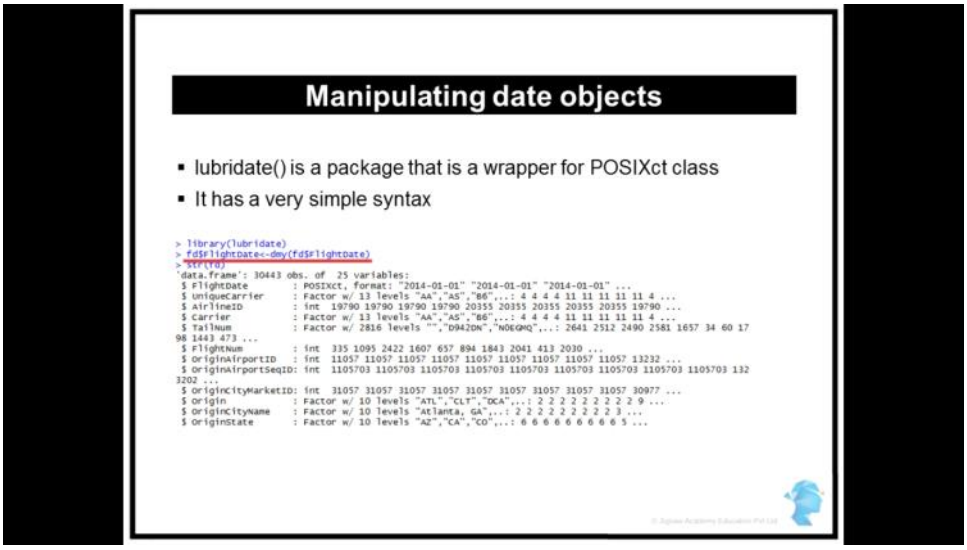
- Whenever data has time information along with date, R uses POSIXct and POSIXlt classes to deal with dates

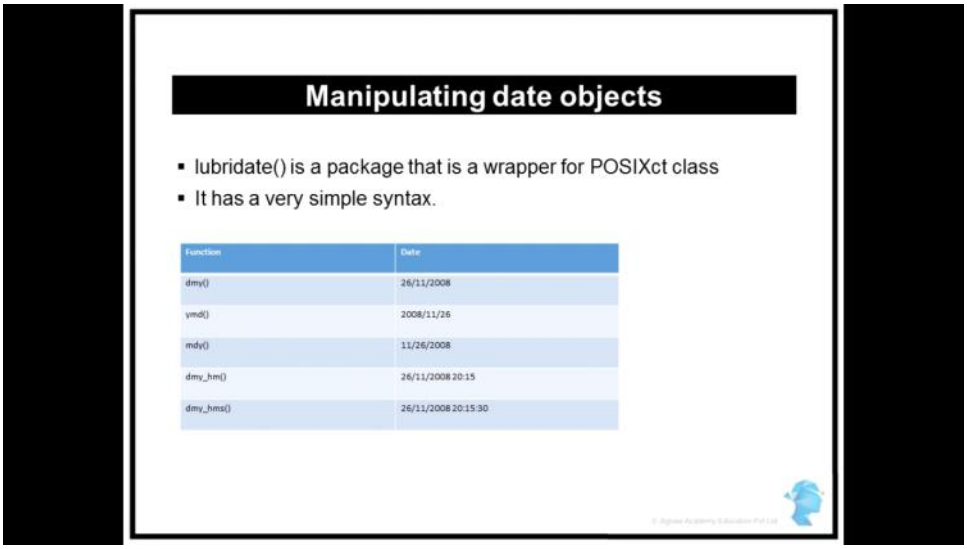
```
> date1<-Sys.time()
> date1
[1] "2015-03-02 17:35:47 IST"
> class(date1)
[1] "POSIXct" "POSIXt"
> weekdays(date1)
[1] "Monday"
> months(date1)
[1] "March"
```

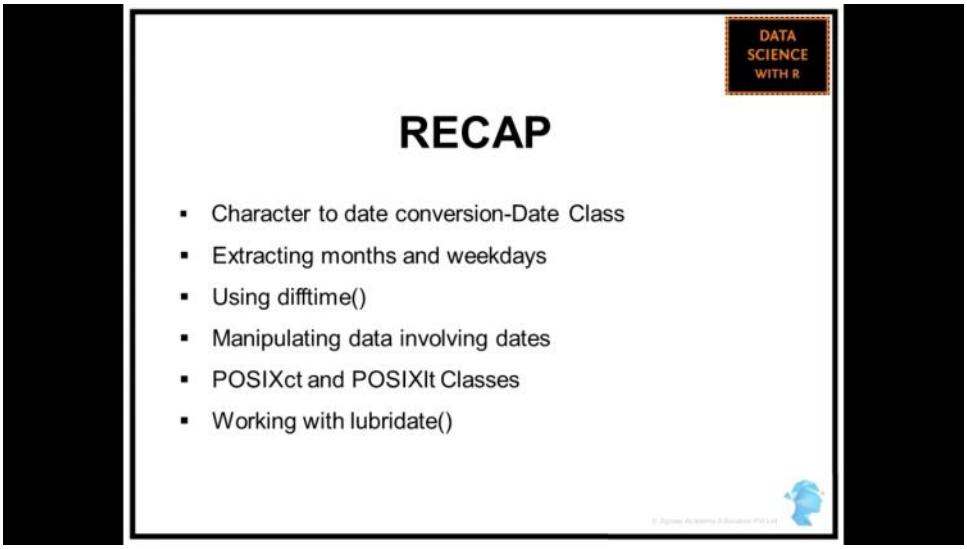


Date2<-as.POSIXlt(date1)
This will store the various date attributes stored as list and makes it easier to exctract this data

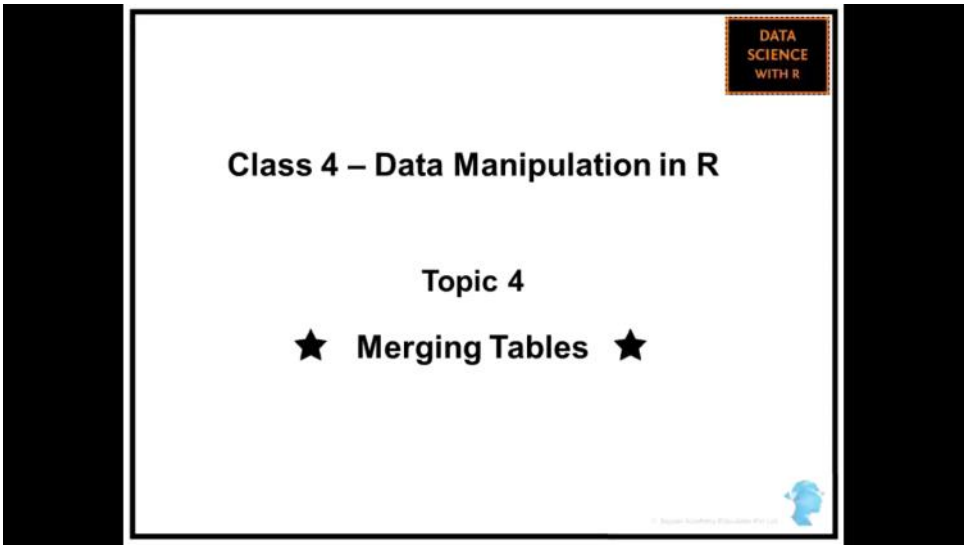
Date2$wday
Date2$zone
Date2$hour







Merging Tables:



Joining dataframes

Joining dataframes

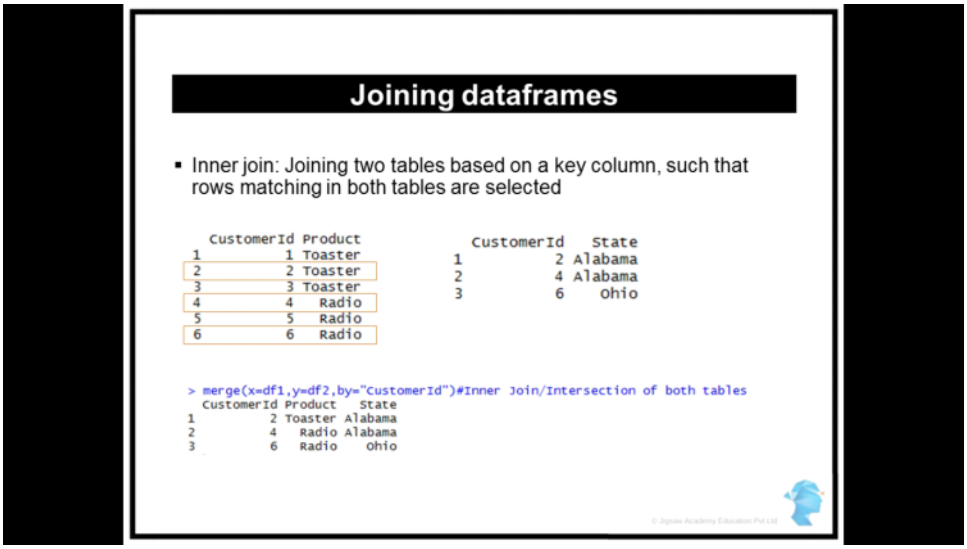
- Just like tables can be joined in sql, we can perform joins on dataframes in R
- Following types of joins can be accomplished
- Inner Join
- Left outer join
- Right outer join
- Full outer join

Joining dataframes

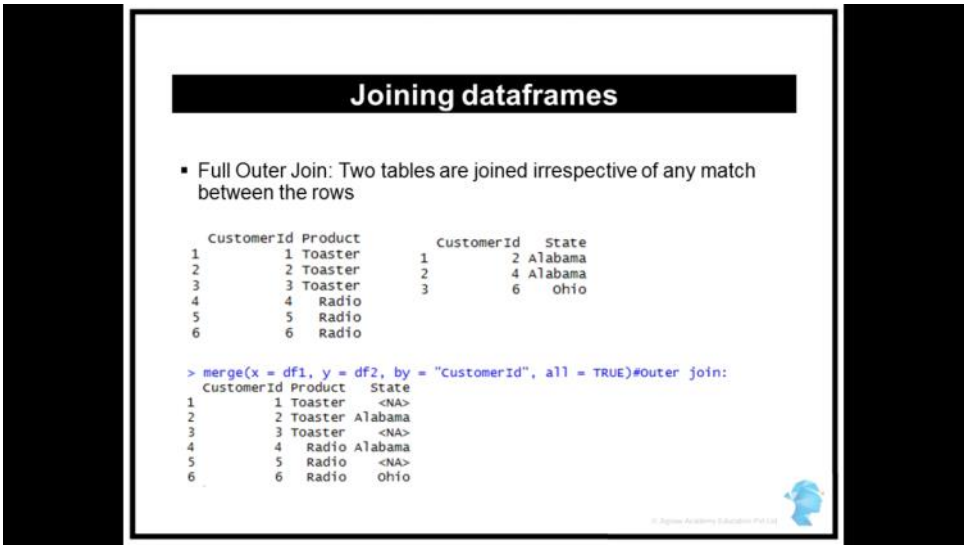
- Inner join: Joining two tables based on a key column, such that rows matching in both tables are selected

	CustomerId	Product
1	1	Toaster
2	2	Toaster
3	3	Toaster
4	4	Radio
5	5	Radio
6	6	Radio

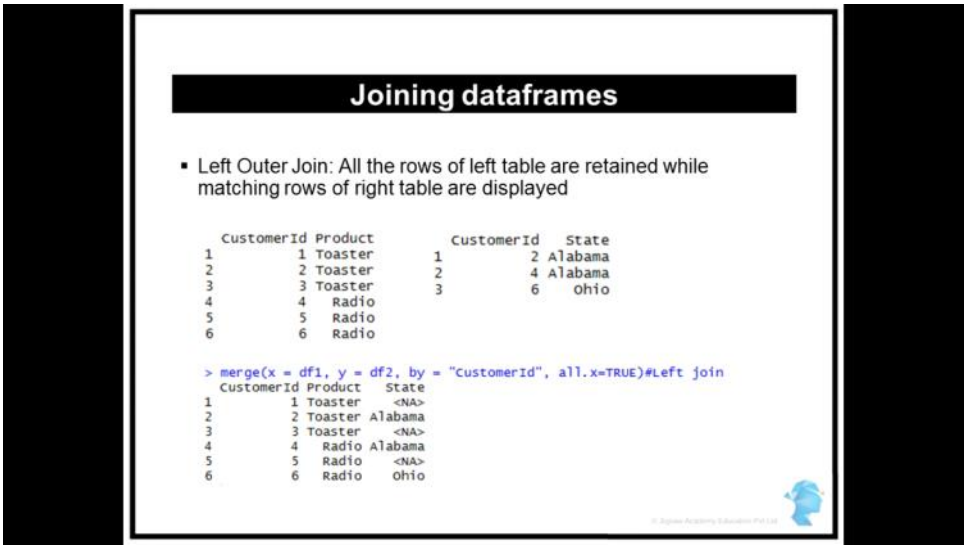
	CustomerId	State
1	2	Alabama
2	4	Alabama
3	6	Ohio



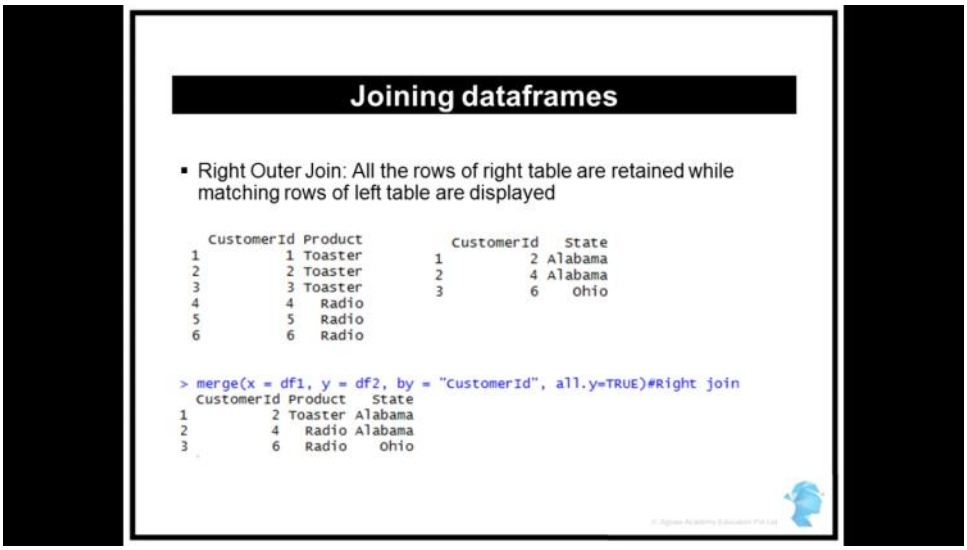
To perform inner join we use the following <- merge(x=df1, y=df2, by="customer id")



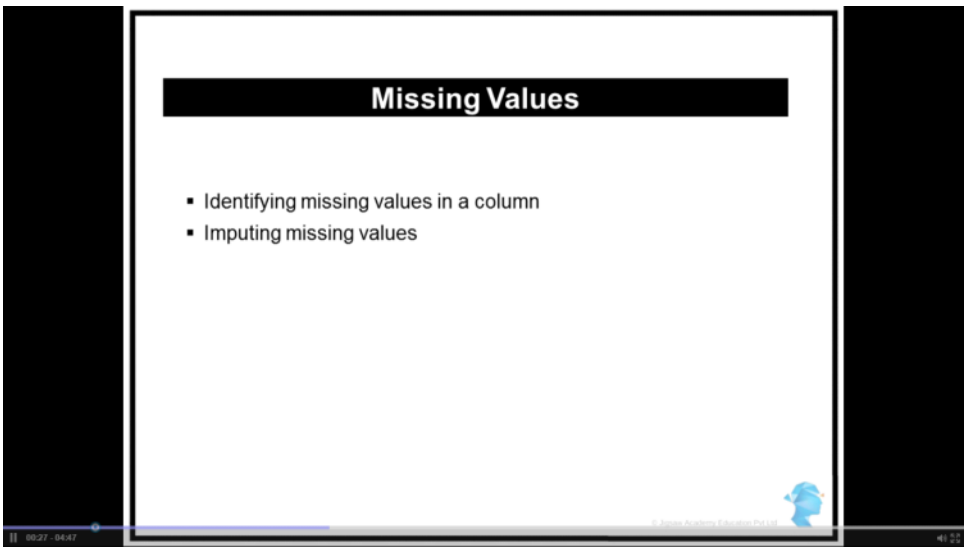
For full outer join <- merge(x=df1, y=df2, by="customer id", all= TRUE)

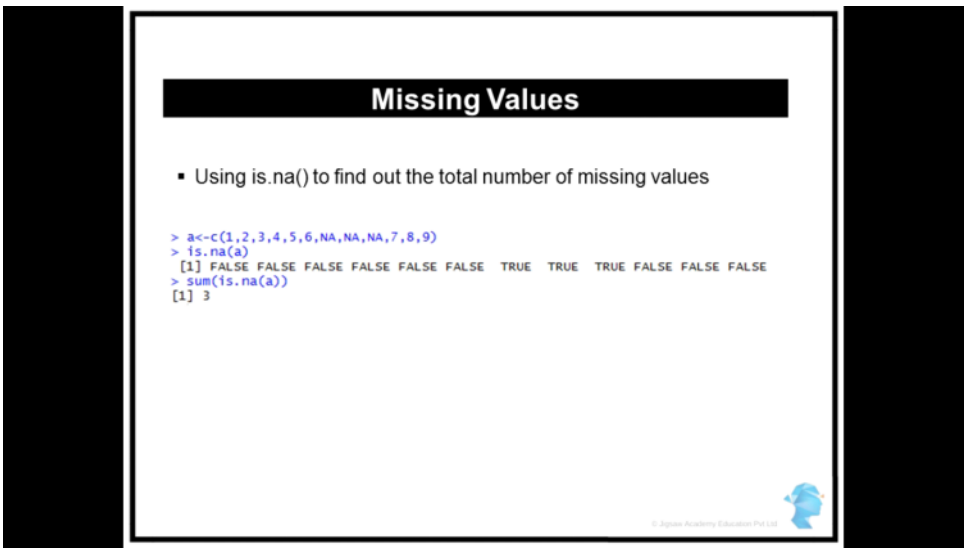


Left outer join <- merge(x=df1, y=df2, by="customer id", all.x= TRUE)
right outer join <- merge(x=df1, y=df2, by="customer id", all.y= TRUE)



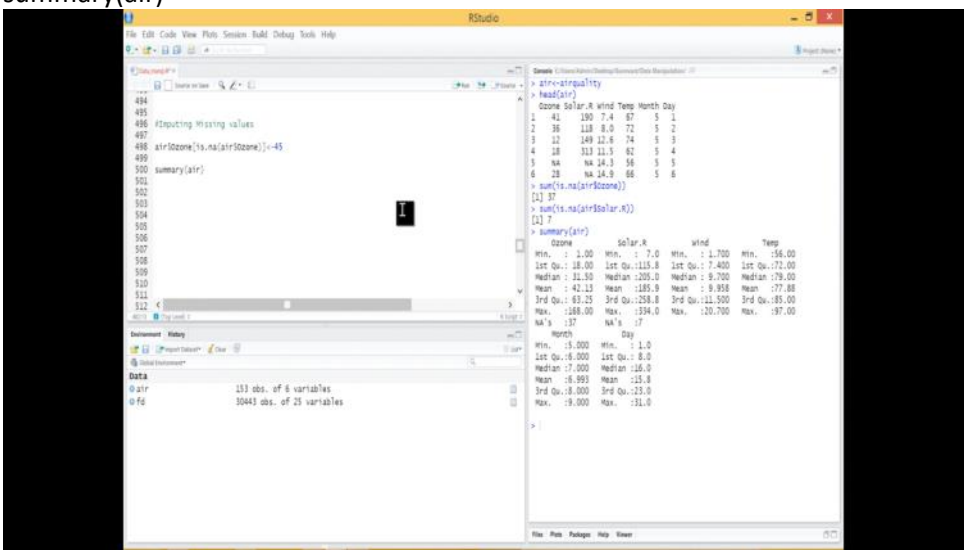
Treating Missing Values:





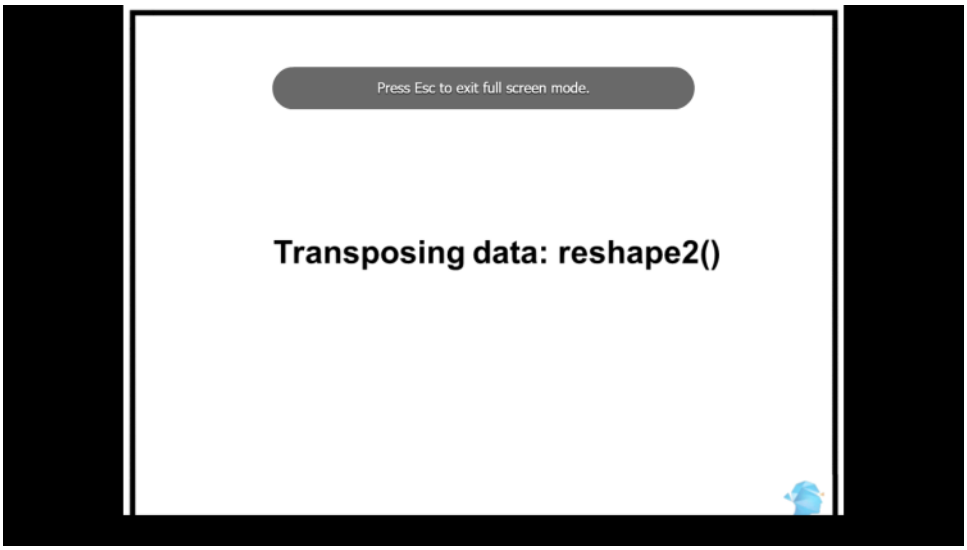
Running a summary command will also give the number of missing values in each column :-

summary(air)



Imputing these missing values :
Air$ozone[is.na(air$ozone)]<-45
Air$Solar.R[is.na(air$Solar.R)]<-mean(air$Solar.R,na.rm=TRUE)

Transposing Data using Reshape2:



Transposing data

- Understanding wide and long data formats
- Converting data in wide format to long
- Converting data in long format to wide

Transposing data

- Understanding wide and long data formats
- Most structured data is in wide format: Variables are columns and Row labels identify observations

Person	Age	Weight
Sankar	26	70
Aiyar	24	60
Singh	25	65

Transposing data

- Understanding wide and long data formats
- The same data can be represented as follows (long format)

Persons	Variable	Value
Sankar	Age	26
Sankar	Weight	70
Aiyar	Age	24
Aiyar	Weight	60
Singh	Age	25
Singh	Weight	65

Use the library --> reshape2

Create a data in wide or long format

Wide <- data.frame(person, age, weight)

To convert this to long format we will use the function called Melt from reshape2

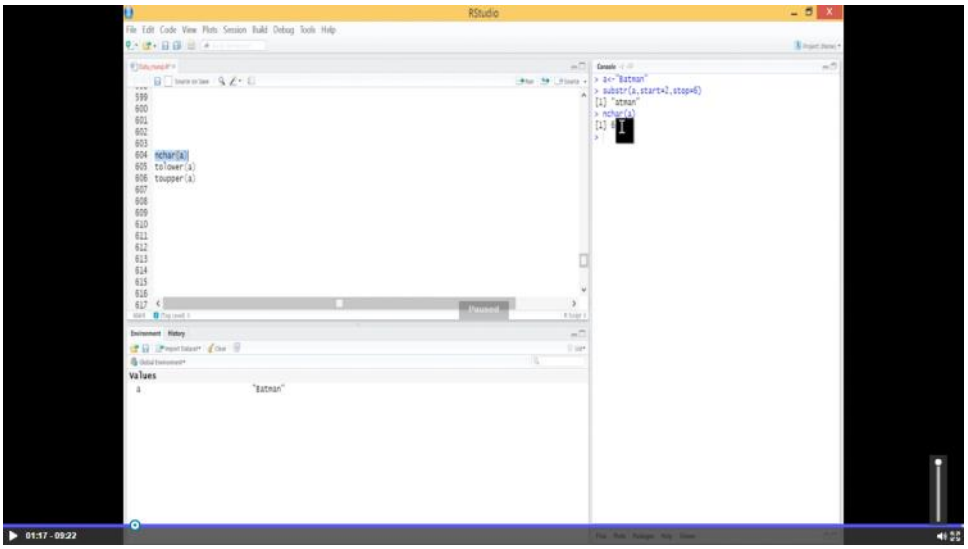
Melt(wide,id.vars="person", value.name="demo_value")
This will generate the long format.

To convert data from long format to wide format we will use dcast

Dcast(melted, person~variable, value.var="demo_value")

Manipulating character strings and using SQLDF

String amnipulation, we can use substr
a<-"Batman"
Susbstr(a,start=2, stop=6)



To find the number of characters in a string --> nchar(a)
To convert all to lower case --> tolower(a)
TO covert all to upper case --> toupper(a)

To split a string, mention the delimiter to split with
Strsplit(a, split="-")

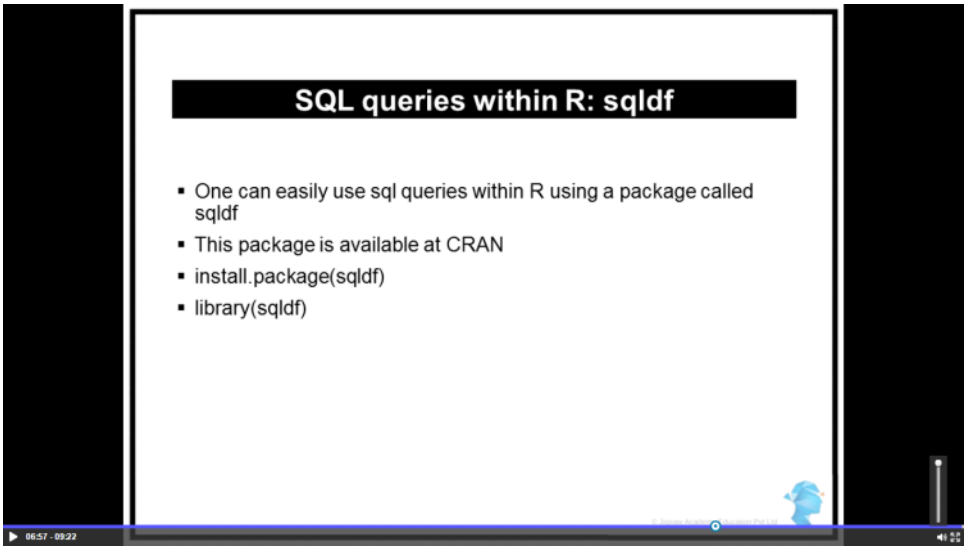
To concatenate two objects together we use paste
Paste(b, split=c)

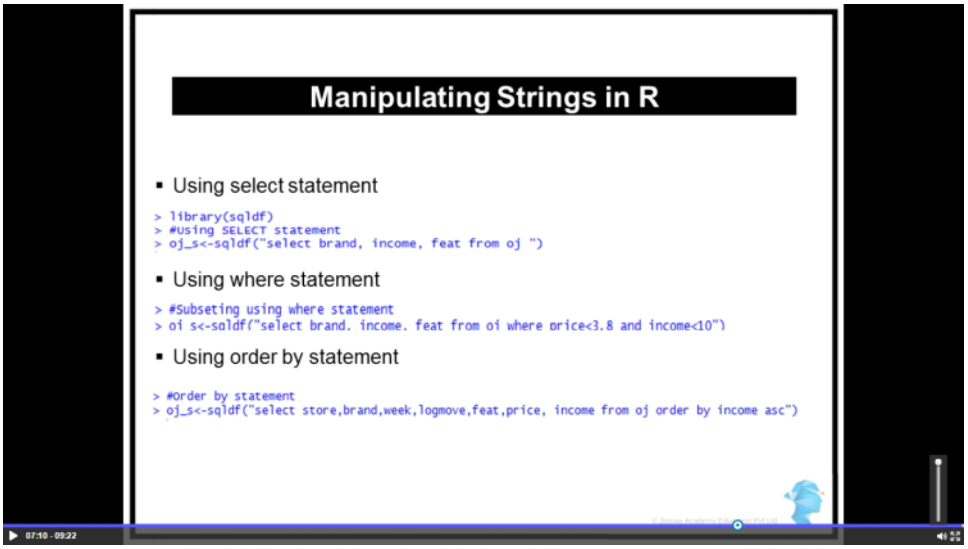
If we want to find a character in the elements of a vector we can use : grep
Grep("-", c(a,b))

We can also use : grepl
This does a logical comparision and returns the results for all the elements based on satisfying the conndition.
Grepl("-",c(a,b))

If we want to substitute a character with another character in an element we use : sub

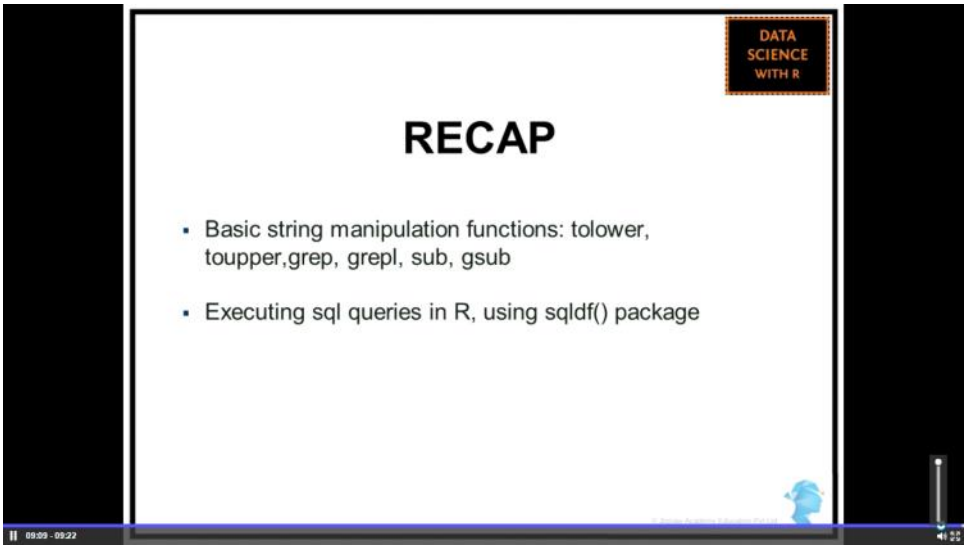
Sub("-","/",a)
But this command only replaces the first occurance of the string. TO replace all the occurances:
Use gsub("-","/",a)





Using the select statement is very easy with this SQLDF library:

Obj_sql<-sqldf("select * from a")
Make sure the object is a table to use with this select statement.


DATA
SCIENCE
WITH R
RECAP
Basic string manipulation functions: tolower, toupper,grep, grepl, sub, gsub
Executing sql queries in R, using sqldf() package