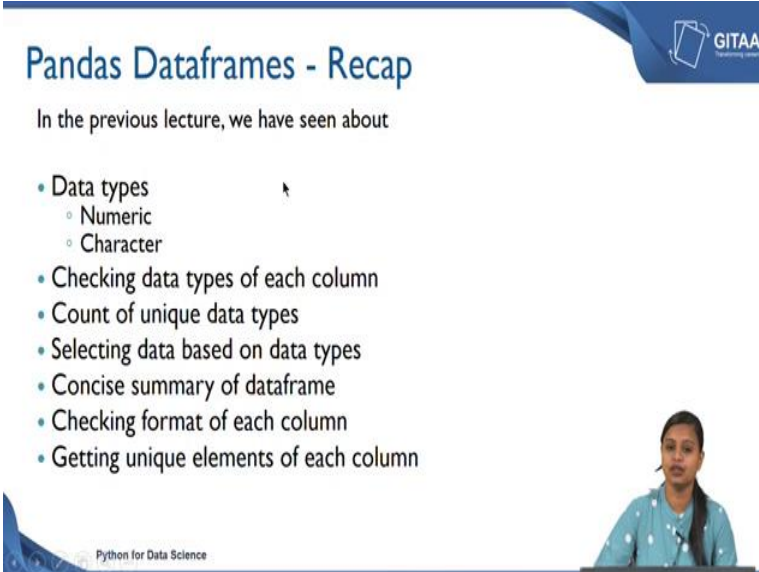


Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 16
Pandas Dataframes Part-III

Hello all, welcome to the lecture on Pandas Dataframes.

(Refer Slide Time: 00:19)



The slide is titled "Pandas Dataframes - Recap" in a blue font. Below the title, it says "In the previous lecture, we have seen about". A bulleted list follows, detailing the topics covered: Data types (with sub-points for Numeric and Character), Checking data types of each column, Count of unique data types, Selecting data based on data types, Concise summary of dataframe, Checking format of each column, and Getting unique elements of each column. In the bottom right corner, there is a small video inset showing a woman in a blue patterned shirt speaking. The slide also features a GITAA logo in the top right and "Python for Data Science" in the bottom left.

- Data types
 - Numeric
 - Character
- Checking data types of each column
- Count of unique data types
- Selecting data based on data types
- Concise summary of dataframe
- Checking format of each column
- Getting unique elements of each column

So, we have been looking at the pandas Dataframes in the previous lectures. So, let us have a quick recap on what we have done in the previous lecture. So, in the previous lecture we have seen about the data types the two broad main data types that are numeric in character which we will be working with often and what they represents and where we can use numeric and character data types.

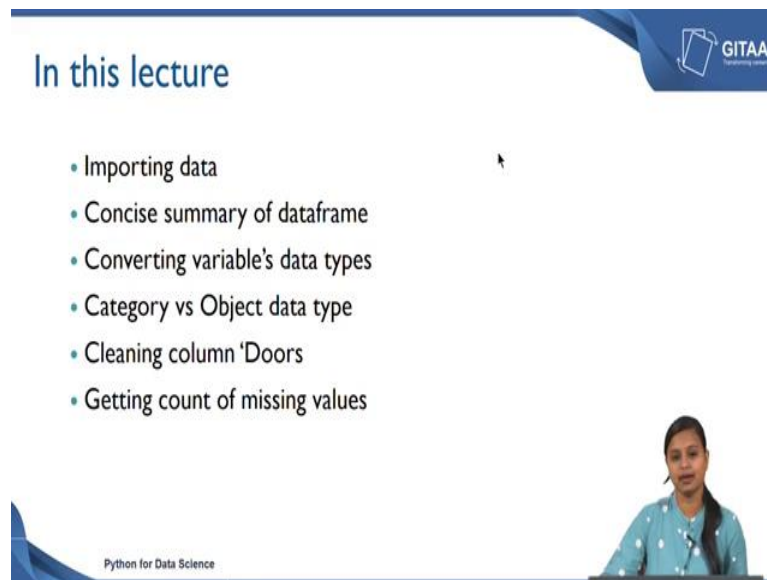
We have also checked the data types of each column to generally know about what are the data types we are going to work with for the analysis and we have also seen how to get the count of unique data types to get an overall idea about how many variables are there with different data types and we have also seen how to select the data based on data types.

For example, if you want to perform any operations only on a specific data type we can also do that by selecting the data based on the particular data types. And we have also looked at the concise summary of Dataframe where we have got some information about

the Dataframe with respect to what is the data type of each variables and how many non-null values are there under each column and which also gave us the memory used by the Dataframe and then we checked the format of each column.

And we found out that there are few columns that has not been read properly. So, we have gone back and looked at the unique elements of each column to get an idea about what is the reason behind variables being read with different data types which are not expected. So, these are the things we have seen in the previous lectures.

(Refer Slide Time: 01:57)



The slide is titled "In this lecture" in a blue font. It features a list of six topics, each preceded by a blue bullet point. The topics are: "Importing data", "Concise summary of dataframe", "Converting variable's data types", "Category vs Object data type", "Cleaning column 'Doors'", and "Getting count of missing values". In the bottom right corner, there is a small video inset showing a woman with dark hair, wearing a blue patterned top, speaking. The slide has a blue header bar with the "GITAA" logo on the right and a blue footer bar with the text "Python for Data Science" on the left.

- Importing data
- Concise summary of dataframe
- Converting variable's data types
- Category vs Object data type
- Cleaning column 'Doors'
- Getting count of missing values

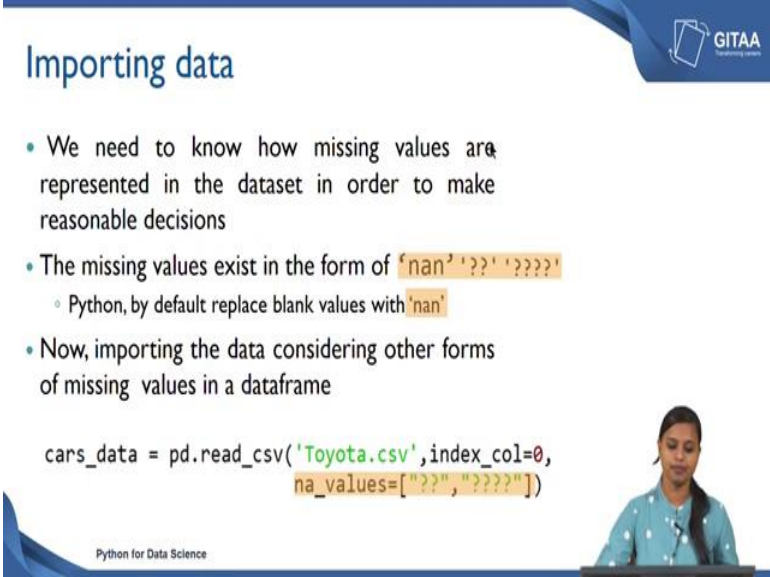
Python for Data Science

And in this lecture we are going to see how to import the data. In the previous lectures we found out that there were question marks and other special characters that are not being considered as blank while you. So, now, in this lecture we are going to import the data by considering all of those after that we are going to again look at the concise summary of data to cross verify whether there is any difference after importing the data by considering all those special characters or not.

After that we are going to convert the variable's data types because in the previous lectures we have encountered that some variables are of not expected data type. So, in this lesson we are going to convert variables to the expected data type and we are also going to look at the category which is object data type what is the effect being the category data type and what is the effect being a object data type.

Next we are going to clean the column door since it has few strings to it like 5, 4 and 3. So, we are going to clean that to make the Doors columns perfect for the future analysis after that we are going to get the count of missing values to basically arrive at a solution on how we are going to go ahead with the missing values. So, now, let us try to import the data by considering the all forms of missing values.

(Refer Slide Time: 03:20)



Importing data

- We need to know how missing values are represented in the dataset in order to make reasonable decisions
- The missing values exist in the form of 'nan' '??' '????'
 - Python, by default replace blank values with 'nan'
- Now, importing the data considering other forms of missing values in a dataframe

```
cars_data = pd.read_csv('Toyota.csv', index_col=0,  
                        na_values=["??", "????"])
```

Python for Data Science

So, now, basically before importing we need to know how missing values are being represented in the data set in order to make any reasonable decisions because you need to be really sure how the missing values are presented in your dataframe, whether it is a blank value or zeros or any question marks or with any special characters.

So, but we have encountered in our data frame the missing values exist in the form of nan which represents the blank values. Double question mark and four question mark represents the missing values wherever they do not have any records for, but we do not have to worry about the blank values because python by default replace blank values with nan. So, now, we know what are the other forms of missing values we have in our dataframe.

So, now, I am going to import the data considering the other forms of missing values in a dataframe. So, I am using the same command as earlier that is read_csv from the pandas library and pd being the alias. So, the input being Toyota csv file and the index I have set

at to 0. So, that the first column will be treated as the index column and I have added another argument called `na_values`.

So, in the argument I have given double question mark and four question marks to basically say consider all these values as nan values. So, now we have seen how to import the data by considering all forms of missing values in your dataframe.

(Refer Slide Time: 04:52)

Concise summary of dataframe

Summary - before replacing special characters with nan	Summary - after replacing special characters with nan
<pre>cars_data.info()</pre>	<pre>cars_data.info()</pre>
<pre><class 'pandas.core.frame.DataFrame'> Int64Index: 1436 entries, 0 to 1435 Data columns (total 10 columns): Price 1436 non-null int64 Age 1336 non-null float64 KM 1436 non-null object FuelType 1336 non-null object HP 1436 non-null object MetColor 1286 non-null float64 Automatic 1436 non-null int64 CC 1436 non-null int64 Doors 1436 non-null object Weight 1436 non-null int64 dtypes: float64(2), int64(4), object(4) memory usage: 163.4+ KB</pre>	<pre><class 'pandas.core.frame.DataFrame'> Int64Index: 1436 entries, 0 to 1435 Data columns (total 10 columns): Price 1436 non-null int64 Age 1336 non-null float64 KM 1421 non-null float64 FuelType 1336 non-null object HP 1430 non-null float64 MetColor 1286 non-null float64 Automatic 1436 non-null int64 CC 1436 non-null int64 Doors 1436 non-null object Weight 1436 non-null int64 dtypes: float64(4), int64(4), object(2) memory usage: 123.4+ KB</pre>

Python for Data Science

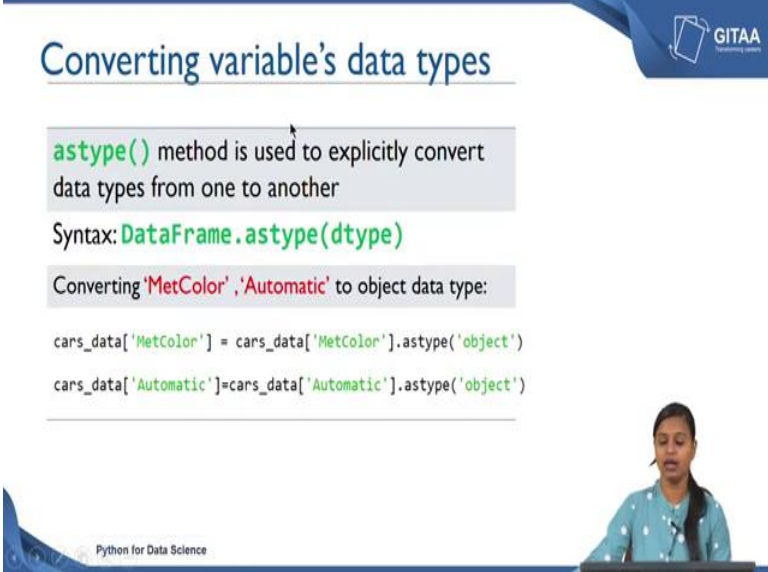
Now let us move ahead and get the concise summary of dataframe because I really want to see the difference after changing the question marks with nan values because ideally there are missing values. So, let us check that. So, this is the output that we got before replacing any special characters with nan.

If you see I have highlighted kilometre and the horsepower column. So, kilometre basically had no missing values 1436, but after replacing all the special characters with nan the kilometres non null values have been decreased from 1436 to 1421 because all rest of the values actually being question marks, but that has not been treated as blank values rather it has been treated as just categories.

So, that is why you see a drop in values; that is as 1421 and after that there is also the similar case with horsepower initially there were no missing values all the 1436 has non null values, but when we consider all the special characters with as nan it says 1430 non null values are there other 6 observations has missing values under horsepower column.

So, this step is really important whenever you are performing any analysis if there are any other forms of missing values that is being encountered in your DataFrame you can make sure that you are replacing all those special characters with the default value. So, now, we have sorted out the issues where we replaced all the special characters with nan. We have to solve the other issue by converting variables data type to the expected data type.

(Refer Slide Time: 06:35)



Converting variable's data types

`astype()` method is used to explicitly convert data types from one to another

Syntax: `DataFrame.astype(dtype)`

Converting 'MetColor', 'Automatic' to object data type:

```
cars_data['MetColor'] = cars_data['MetColor'].astype('object')
cars_data['Automatic'] = cars_data['Automatic'].astype('object')
```

Python for Data Science

Because in the previous lecture we have seen that some of the variables have not been read properly with the correct data type and we have encountered the reasons as well now it is time to change or convert the variables data type to the expected data type. And `astype` is the function which is used to explicitly convert data types from one to another and the syntax being `astype` of data type you can basically specify to which data type you want to convert it to and you use have to use your DataFrame name.

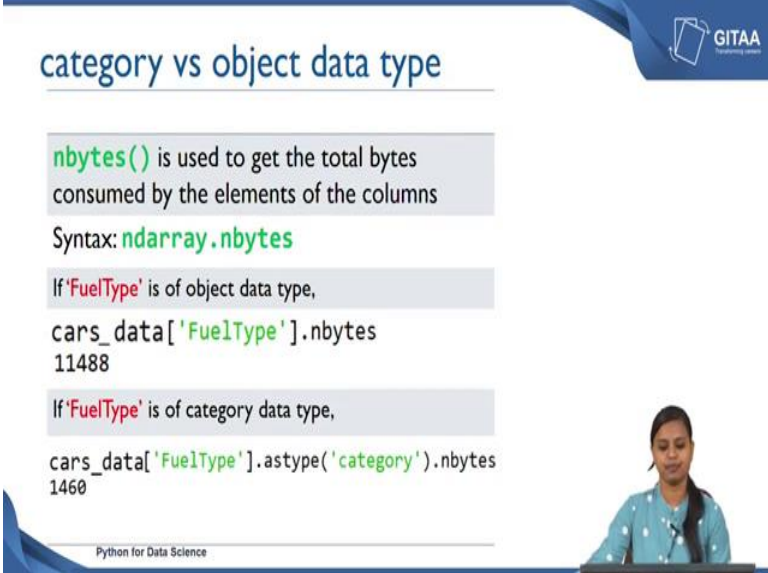
So, here I am interested in converting the MetColor on the Automatic variable to object data type because those values represents just the categories to it, I am going to convert them to object data type. So, how I am doing here is I am accessing the MetColor from the `cars_data` and using the `astype` function, I am converting the MetColor to object by giving object inside the function.

So, whatever I am doing here that should be reflected back to original variable that is why I have equated that to `cars_data` of MetColor. So, now, MetColor has been

converted to object similarly you can try that for Automatic. So, I have done it here I have converted the Automatic variables also to object data type and I have reflected the changes to the original variable as well.

Now, we have converted the MetColor in Automatic to object data type, but we have already seen there are two data types that the python can handle with respect to character data type. One is being category and another one is being object. We have already changed everything to object there is also another data type called category right. So, let us see what is the impact when we have these MetColor or any categorical variable as category or as object.

(Refer Slide Time: 08:35)



category vs object data type

`nbytes()` is used to get the total bytes consumed by the elements of the columns
Syntax: `ndarray.nbytes`

If `'FuelType'` is of object data type,
`cars_data['FuelType'].nbytes`
11488

If `'FuelType'` is of category data type,
`cars_data['FuelType'].astype('category').nbytes`
1460

Python for Data Science

So, here is a slide which will illustrate the impact of variables being category and the impact of variables being object data type. So, nbytes is the command that is used to get the total bytes consumed by the elements of columns, we are going to look FuelType as object data type and FuelType as category data type.

So, I am going to get the total bytes consumed by the elements with respect to object data type as well as category data type, for that nbytes can be used and the syntax would be n dimensional array.nbytes. So, it can be n dimensional array and you can use nbytes from it. So, I am using like I am accessing the fuel type from the cars_data frame and I have used nbytes to get the total bytes consumed by the elements where the FuelType is

being object. So, when the FuelType is being object it has consumed 11488 bytes, when it is of category.

So, here if you check I have converted FuelType to category and then I am checking the total bytes consumed by the element of the column, but it has drastically dropped down to 1460 bytes. So, this is the advantage of keeping categorical variables as category data type instead of keeping them as objects. So, this will be really useful when you are dealing with a huge amount of data and you want to reduce the space that is being allocated to each and every variable or for each and every cell.

So, in that case you would go for category data type for all the string values that you have in your dataframe, but in our case we are just dealing with a less amount of data. So, there is no need to explicitly convert the variable to category, but if you see we are going to go ahead and give the FuelType as object and the slide is just for the illustration purpose.

(Refer Slide Time: 10:46)

Re-checking the data type of variables

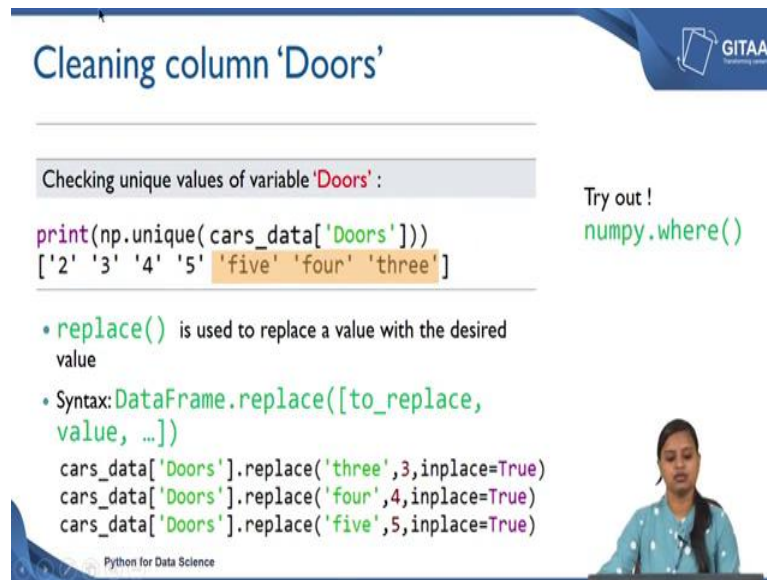
Re-checking the data type of variables after all the conversions

```
cars_data.info() <class 'pandas.core.frame.DataFrame'>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
Price      1436 non-null int64
Age        1336 non-null float64
KM         1421 non-null float64
FuelType   1336 non-null object
HP         1430 non-null float64
MetColor   1286 non-null object
Automatic  1436 non-null object
CC         1436 non-null int64
Doors      1436 non-null object
Weight     1436 non-null int64
dtypes: float64(3), int64(3), object(4)
memory usage: 123.4+ KB
```

Python for Data Science

And we have changed the data type for few variables, now I want to recheck the data type of variables after all the conversions that I have made. So, I am again checking the concise summary of the DataFrame cars data using the.info command, if you see now all the variables are of expected data type. For example, kilometre is of float MetColor is of object Automatic color is of object. So, now we have to clean the column doors.

(Refer Slide Time: 11:13)



The slide is titled "Cleaning column 'Doors'" and features a GITAA logo in the top right corner. It contains a code block for checking unique values of the 'Doors' column, which outputs a list of unique values: ['2', '3', '4', '5', 'five', 'four', 'three']. Below this, it explains the use of the `replace()` function to replace string values with numerical ones. The syntax is given as `DataFrame.replace([to_replace, value, ...])`. Three lines of code are shown: `cars_data['Doors'].replace('three', 3, inplace=True)`, `cars_data['Doors'].replace('four', 4, inplace=True)`, and `cars_data['Doors'].replace('five', 5, inplace=True)`. A small inset image of a woman is visible in the bottom right corner of the slide.

Cleaning column 'Doors'

Checking unique values of variable 'Doors' :

```
print(np.unique(cars_data['Doors']))  
['2' '3' '4' '5' 'five' 'four' 'three']
```

Try out !
`numpy.where()`

- `replace()` is used to replace a value with the desired value
- Syntax: `DataFrame.replace([to_replace, value, ...])`

```
cars_data['Doors'].replace('three', 3, inplace=True)  
cars_data['Doors'].replace('four', 4, inplace=True)  
cars_data['Doors'].replace('five', 5, inplace=True)
```

Python for Data Science

Because in the previous lecture we have seen that under the Doors column it has some string values to it. So, I just want to recap what we have seen in the Doors column, we have checked the unique values of variable Doors it turned out to have some unique values which are in numbers as well as in strings. So, those basically represents five, four, three in numbers. So, there might be cases where they have wrongly typed the numbers in terms of strings.

So, now if you want to have a consistent values like if you want to only have numerical values like 2, 3, 4 and 5 you can always go back and change all the string values to numerical values by using a command in python. So, the command is `replace` that is used to replace a value with the desired value you can replace any numerical to string as well as string to numericals. So, the syntax would be `DataFrame.replace`.

Inside the function I have used the value to be replaced and after that the second argument would be what is the value that you are going to replace it with. So, here I have problem in three strings. So, using a `replace` command I can replace all string values to numbers. So, let us take the string three first.

So, I am accessing the Doors column from the DataFrame `cars_data` and by using `replace` command. The first argument that I have used is the value that should be replaced that is 3 and using which value that is the number 3 and I have used `inplace` is equal to `true`. So,

the modifications done in the DataFrame will be reflected back in the DataFrame that you are working on. Similarly I have done it for all the other strings like four and five.

So, now, I have replaced all the string values to numbers we do not want multiple values which represents the same meaning. Now, I have a consistent values like 2, 3, 4 and 5. So, now, this is not the only way where you can replace any value with the desired value there are several function that does the same one is using where. So, you can try out where function from numpy, but you can also use the where function on pandas libraries as well. So, you can try that out to replace all the string values with the numbers.

(Refer Slide Time: 13:39)

Converting 'Doors' data type

Converting 'Doors' to int64:

```
cars_data['Doors']=cars_data['Doors'].astype('int64')
```

Python for Data Science

Now, it is time to convert the Doors to int65 because whatever levels you have added, whatever values you have replaced it with it will create it as a new set of values because python cannot differentiate between the existing set of values and the newly added values. So, it will basically create new set of categories for example, 2 3 4 5 again it will have 3 4 5 because that is the categories that we have added newly.

So, in that case let us convert them back to integer64. So, that 2, 3, 4 and 5 become the integer data type that represents the number of Doors being 2, 3, 4 and 5. So, as we know that I can convert them back to integer 64 by using the command astype and I have reflected the changes to the original variable doors. So, this is to basically have the existing levels as it is if we replace any values that will get added as a new level rather

than updating it in the original level. So, just to avoid that confusion I am reconverting them back to the integer data type.

(Refer Slide Time: 14:56)

To detect missing values

To check the count of missing values present in each column `Dataframe.isnull.sum()` is used

```
cars_data.isnull().sum()
```

```
Out[108]:
Price      0
Age       100
KM         15
FuelType   100
HP          6
MetColor   150
Automatic   0
CC          0
Doors       0
Weight      0
dtype: int64
```

Python for Data Science

So, now we are done with cleaning the Doors column. So, now, we have to check whether there are any missing values in your Dataframe or not. So, to check the count of missing values present in each column `Dataframe.isnull.sum` will be used basically `isnull` is a function which gives you an output in terms of Boolean values that is true or false.

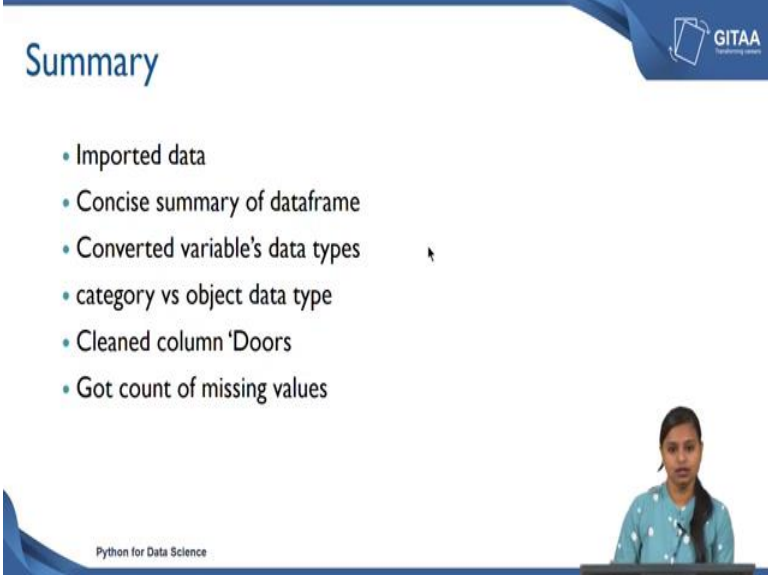
So, it will consider the whole Dataframe because you are applying the `isnull` on the Dataframe wherever you have missing values it will be represented as true, wherever the cell does not have any missing values it will be represented as false and I have used `.sum` to sum up all the true values because I am interested in getting the count of missing values present in each column. So, that is why I have used `.sum` after `isnull`.

So, this is the input that I have given `cars_data.isnull.sum()` that will give me the missing values present in each column. So, under the price column I do not have any missing values, but if you see under age we have 100 missing values present under the age column. So, here is the output for the command `isnull.sum`.

So, under the price variable we do not have any missing values, but if you see the age variable has 100 missing values and the kilometre variable has 15 missing values, similarly FuelType has 100, horsepower has 6 and MetColor has 150 missing values. So,

now, we know that there are some missing values under each columns and the missing values are not same across all the columns because it is differing with each variables. So, we have to come up with the logic approach to see how we are going to fill those missing values or how we are going to deal with this missing values at all.

(Refer Slide Time: 16:56)



Summary

- Imported data
- Concise summary of dataframe
- Converted variable's data types
- category vs object data type
- Cleaned column 'Doors'
- Got count of missing values

Python for Data Science

So, let us summarize whatever we have seen in this lecture we have imported the data by considering all the other forms of missing values as blank values by default python takes it as null. After imported the data we have got the concise summary of data where we have checked the difference after replacing all the special characters with nan values, we have also converted the variables to the expected data types.

Next, we have seen what is the impact of having a variable with the data type category and having the variable of the data type object. And we have also cleaned that column called Doors where it had few strings to it and we have replaced all the strings to numbers. So, now, we also have an idea about how many missing values are present in each column. So, in the upcoming lectures we will be seeing about how to go ahead and deal with the missing values by imputing those.