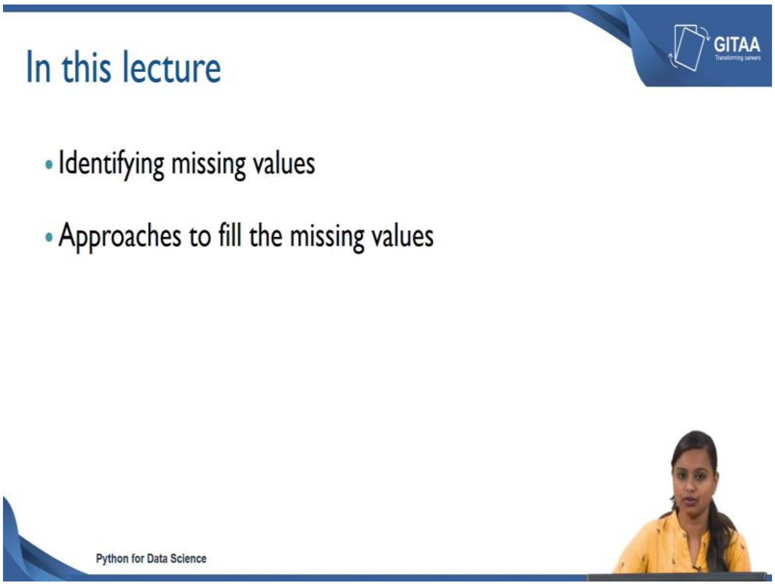
Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 25
Dealing with missing values

Welcome to the lecture on Dealing with missing values.

(Refer Slide Time: 00:19)



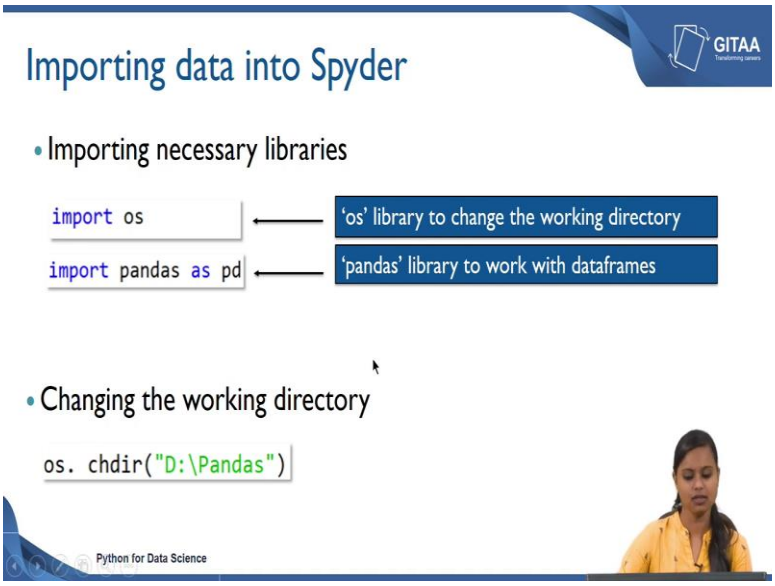
In this lecture

- Identifying missing values
- Approaches to fill the missing values

In the previous lecture we have analyzed the data with complete cases. The complete cases in the sense, we have not considered the missing values in the analysis; wherever they were missing values we have omitted those missing values and then we will proceed with the analysis. That would not be the real case scenario, because in a real time scenario you would be having lot of missing values which you will be dealing with, in that case they need to be your way or an approach on how to deal with those missing values.

So, in this lecture we are going to see how to identify the missing values and some of the approaches to fill in those missing values. So, these are the major concepts that we are going to see in this lecture.

(Refer Slide Time: 01:01)



Importing data into Spyder

- Importing necessary libraries

```
import os
```

'os' library to change the working directory

```
import pandas as pd
```

'pandas' library to work with dataframes

- Changing the working directory

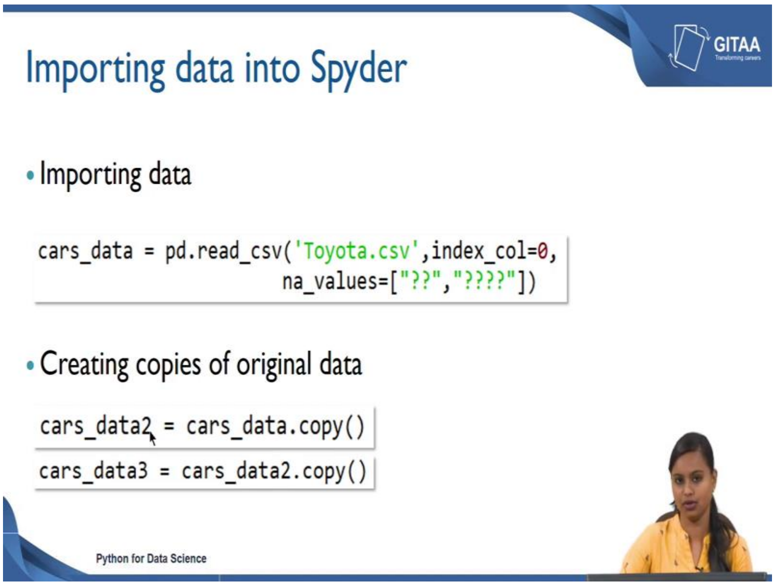
```
os. chdir("D:\Pandas")
```

Python for Data Science

So, prior to importing your data into Spyder, we need to import some of the necessary libraries that are required for the analysis. So, we are going to import the os library that has to change the working directory. So, that you will be able to access the file, then we are going to import the pandas library as pd and this is to work with data frames, so that we can use any functionalities that are from the pandas library.

Next we can now change your working directory by using the chdir command, inside the function you just need to give the file path of it. So, I given the file path; once you set your working directory you will be able to access the file in your current working directory.

(Refer Slide Time: 01:42)



Importing data into Spyder

- Importing data

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

- Creating copies of original data

```
cars_data2 = cars_data.copy()
```

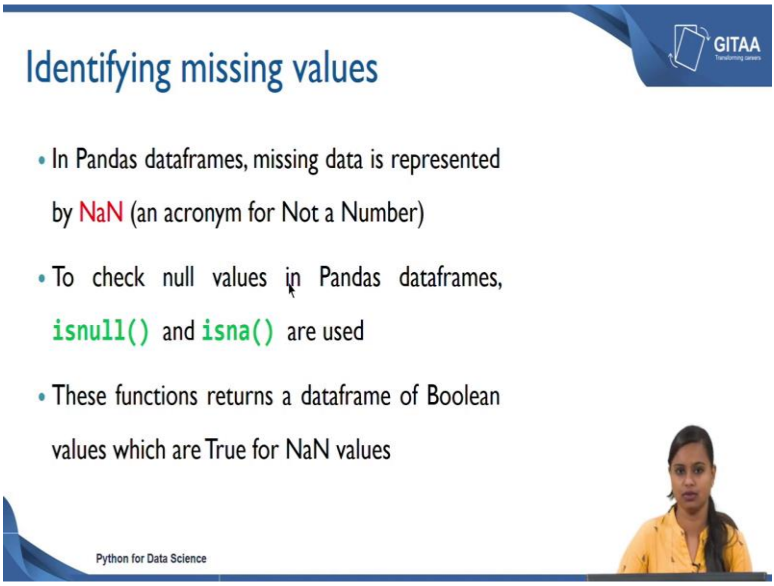
```
cars_data3 = cars_data2.copy()
```

.

Next let us import the data into Spyder, I am importing the Toyota.csv which we were walking on, and I have said the first column as my index column and I have also given some set of string values that needs to be considered as python NaN values; and I have save my output on object called cars_data. Now, cars_data becomes the data frame, whatever I am going to do that should not be reflected in the original data that I have now; rather I can just create multiple copies of data, so that I can work with the duplicated data.

So, let us create copies of original data. So, using the.copy command, so I am creating the copy from the original data cars_data; and saving that as a new data frame called cars_data2. So, this is one copy and I have need also another copy called cars_data3 from the cars_data2 data frames. So, I have now two copies of data from the original one; that is cars_data2, and cars_data3. Now, we are going to see how to identify the missing values.

(Refer Slide Time: 02:48)



Identifying missing values

- In Pandas dataframes, missing data is represented by NaN (an acronym for Not a Number)
- To check null values in Pandas dataframes, `isnull()` and `isna()` are used
- These functions returns a dataframe of Boolean values which are True for NaN values

Python for Data Science

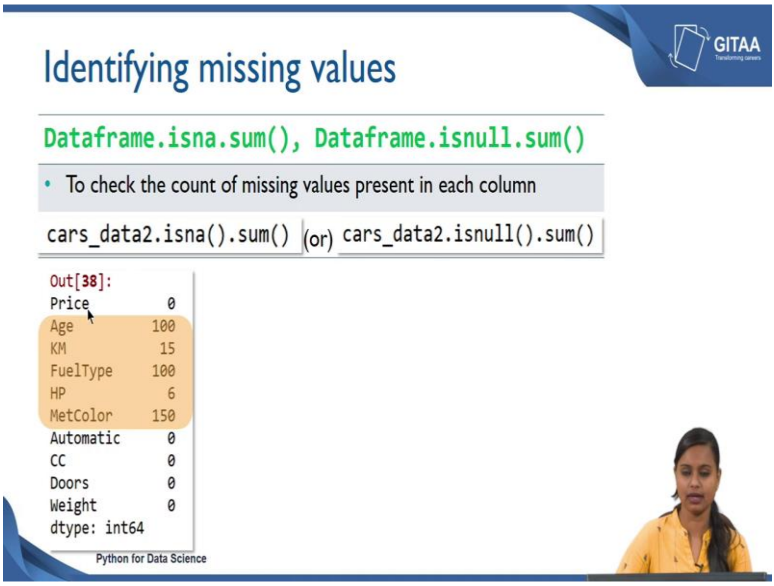
GITAA
Transforming careers

In Pandas data frames, missing data is represented by NaN. And NaN is just an acronym for not a number. So, whenever you have a blank cell in your csv file, the python will automatically read it as NaN value. So, all those NaN values will be treated as missing values in Pandas data frames. And if you want to check that null values in Pandas data frames, there are several functions that are available to do any operations that are related with dealing with missing values. We are going to considered only the two functions that are used to check the null values in Pandas data frames, two of it has been shown here.

The first one is isnull and the second one is isna; both of the functions will account for the python default NaN values. These functions can be used to check the null value city of Pandas data frames. Because before proceeding with how to deal with your missing values, you need to identify whether there are any missing values or not or how many of them are there. In that case these functions would be really helpful to do that; these functions basically returns a data frame of Boolean values which are true for NaN values. And, these functions returns data frame of Boolean values which are true for NaN values.

Basically, you will get an output with only true or false values. So, wherever you have NaN values it will you will see it as true, and wherever you do not have NaN values rather you have other values then you will have the value as false. We will see now how to use these two functions.

(Refer Slide Time: 04:20)



Identifying missing values

Dataframe.isna.sum(), Dataframe.isnull.sum()

- To check the count of missing values present in each column

```
cars_data2.isna().sum() (or) cars_data2.isnull().sum()
```

```
Out[38]:
Price          0
Age          100
KM            15
FuelType     100
HP             6
MetColor     150
Automatic      0
CC             0
Doors          0
Weight         0
dtype: int64
```

So, as I have mention these isna and isnull are the functions that are used to check the missing values in the Pandas data frames. So, we are going to get the count of missing values present in each column. Let us see how to do that, because I do not want to look at how many missing values are there in a complete data frame rather I will just want to look at how many number of rows are missing under each column.

So, that it will give me an idea on how to proceed with the missing value, so that it will give me an idea how to deal with that missing values. So, the syntax is Dataframe.isna.sum, in order to get the count of missing values present in each column. And you can also use isnull function with the same syntax. So, I have shown the code here; that is cars data2isna.sum. We will see what the dot sum is used for. Similarly I have used it for isnull, though both of the functions give you the same output and gives you the same information you can choose any one of it.

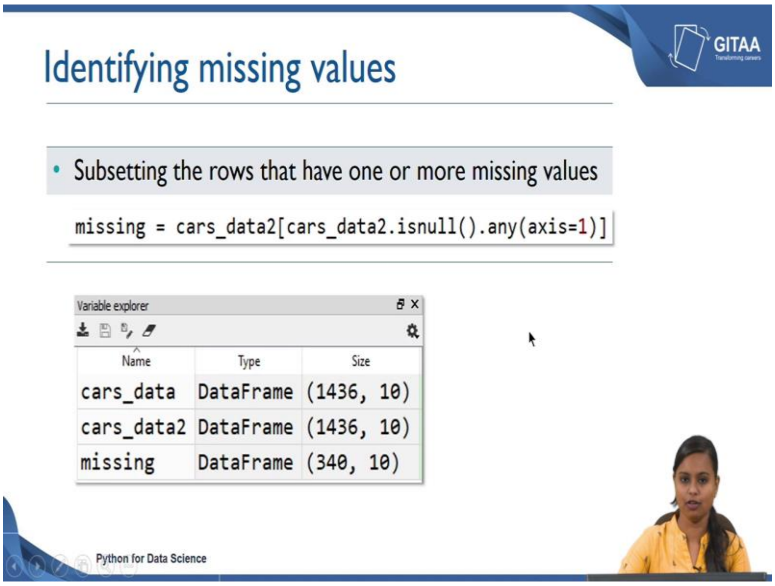
So, now let us look at the output. So, the output shown is, the output shown here is for the function isna. And you are getting the column names correspondingly you are also getting how many number of rows are missing under that particular column. For example, age has 100 rows where there are missing values, kilometer has 15 rows where there are missing values; similarly FuelType has 100 rows with missing values, horsepower has only 6 rows with missing values, and the MetColor has 150 rows with missing values none of the other variables have missing values. So, now we have got an idea about how many number of rows are missing under each of the columns.

Now, we got an idea about how many number of rows are missing under each columns. Now it is not enough to know that, because we need to check whether 100 rows missing under age is completely different from the 100 rows that are missing under FuelType, or there are any sequence, or there are any combination of missing values that are available in our data frame. In a same row, you want to see whether only one column is missing or multiple column values are missing; because whenever you have missing values you need to have two types of approach.

One is to completely get rid of all the rows wherever there are missing values. And the other one is, to have an approach to logically fill up those missing values. So, these two cases depends upon the problem that you have. So, if you have a row when all the column values are missing or most of the column values are missing; then there is no point in refilling the missing values with any substitute value right, you can just get rid of that particular row alone.

In that case, removing the rows will help you and even if there are; and the second thing is if there is no pattern to your missing values like if it is just randomly missing all over the cells, then you should go and look at an approach to fill in those missing values. In that case we this numbers will not give us an idea about that, we need the subset all the rows and see where the missing values are there and then we can decide on what we have to do with the missing values.

(Refer Slide Time: 07:37)



Identifying missing values

- Subsetting the rows that have one or more missing values

```
missing = cars_data2[cars_data2.isnull().any(axis=1)]
```

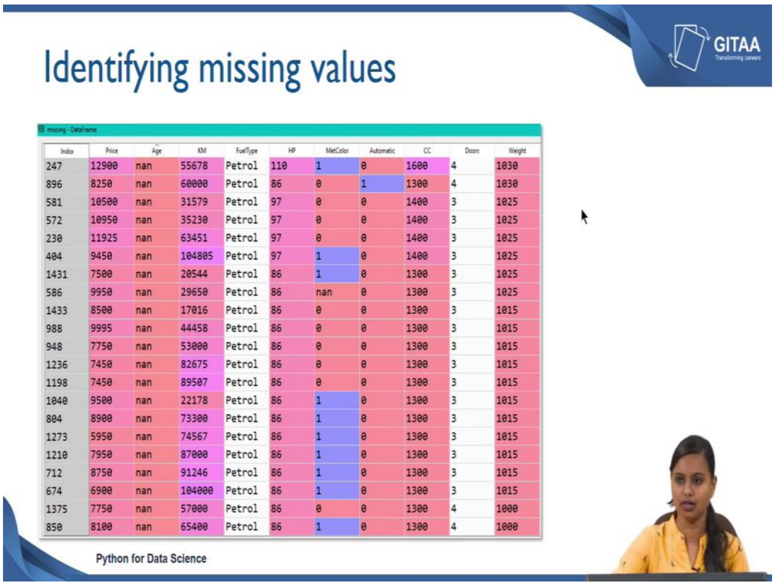
Variable explorer

Name	Type	Size
cars_data	DataFrame	(1436, 10)
cars_data2	DataFrame	(1436, 10)
missing	DataFrame	(340, 10)

So, now we are going to subset the rows that have one or more missing values, because I need to consider a row wherever there are only one column is missing and as well as more than one column is missing. In that case, the code shown here will do that, from the cars data2, I am subsetting the rows wherever there are missing values using the function isnull and I have given.any of axis = 1; the 1 represents column.

I am telling the function give me all the rows wherever at least one column value is missing. And we have save that to an object call missing, so it becomes a data frame. Let see how the missing values are or now let us see where the missing values are there. So, once you read that, you will get that in your variable explore. Let us check the dimension of it, it is a data frame now; and the dimension of it is 340 rows with 10 columns. So, there are 340 rows, where at least one column value is missing. So, now let us see where the missing values are there.

(Refer Slide Time: 08:43)



Identifying missing values

missing - DataFrame

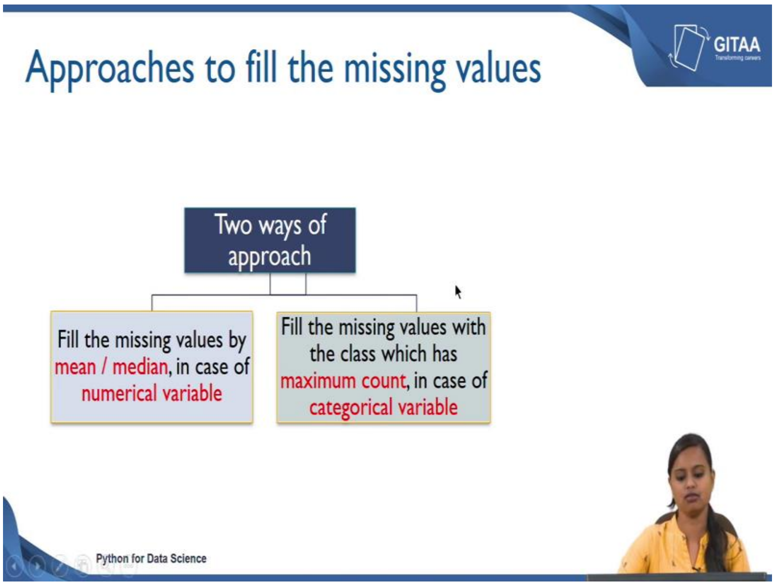
Index	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
247	12900	nan	55678	Petrol	110	1	0	1600	4	1030
896	8250	nan	60000	Petrol	86	0	1	1300	4	1030
581	10500	nan	31579	Petrol	97	0	0	1400	3	1025
572	10950	nan	35230	Petrol	97	0	0	1400	3	1025
230	11925	nan	63451	Petrol	97	0	0	1400	3	1025
404	9450	nan	104805	Petrol	97	1	0	1400	3	1025
1431	7500	nan	20544	Petrol	86	1	0	1300	3	1025
586	9950	nan	29650	Petrol	86	nan	0	1300	3	1025
1433	8500	nan	17016	Petrol	86	0	0	1300	3	1015
988	9995	nan	44458	Petrol	86	0	0	1300	3	1015
948	7750	nan	53000	Petrol	86	0	0	1300	3	1015
1236	7450	nan	82675	Petrol	86	0	0	1300	3	1015
1198	7450	nan	89507	Petrol	86	0	0	1300	3	1015
1040	9500	nan	22178	Petrol	86	1	0	1300	3	1015
804	8900	nan	73300	Petrol	86	1	0	1300	3	1015
1273	5950	nan	74567	Petrol	86	1	0	1300	3	1015
1210	7950	nan	87000	Petrol	86	1	0	1300	3	1015
712	8750	nan	91246	Petrol	86	1	0	1300	3	1015
674	6900	nan	104000	Petrol	86	1	0	1300	3	1015
1375	7750	nan	57000	Petrol	86	0	0	1300	4	1000
850	8100	nan	65400	Petrol	86	1	0	1300	4	1000

Python for Data Science

Right click your missing data frame, you will get this window. So, the snippet shown here is the missing data frame and I have flitted out all the rows wherever there are missing values under the age variable. And, if we look at the corresponding columns, there are no missing values accept only one. There is no pattern to the missing values, it is just missing completely in the random case; because it is completely randomly missing under each variable. If you were to subset all the rows wherever there are missing values under kilometer, then that is also the case.

You do not have any pattern to it that holds good for the other columns. I have just shown you for an example, to illustrate that there are no pattern to the missing values that are there here, rather the missing values are completely random in the variable age. So, now, and in that case, we cannot just go ahead and drop all the missing values that are there in your data frame, because if you consider removing a row wherever there are at least one missing value; then you will get rid of 340 observation from 1436 observations, right; that is a huge loss of information that you are going to have. Now, we should have an approach to fill in those missing values.

(Refer Slide Time: 10:00)



Approaches to fill the missing values

Two ways of approach

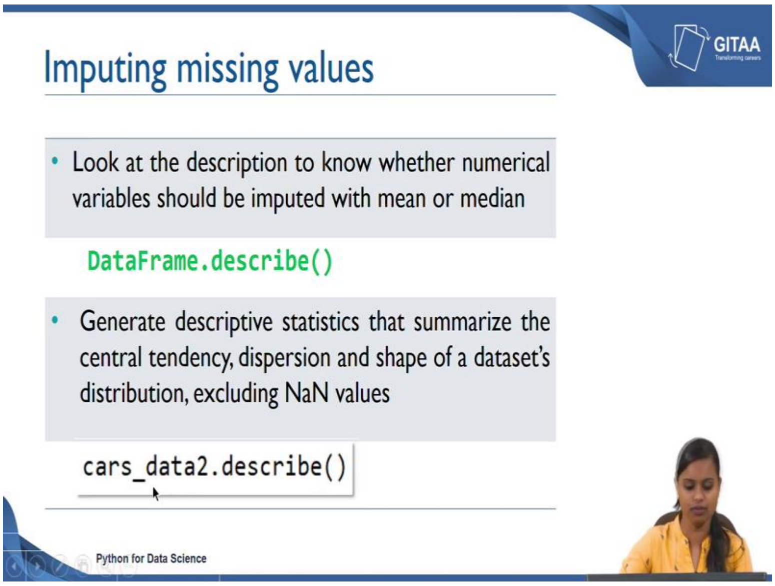
Fill the missing values by mean / median, in case of numerical variable

Fill the missing values with the class which has maximum count, in case of categorical variable

What can be the approaches to fill the missing values? There are several ways to fill in the missing values. In this lecture we are going to look at two ways of approach; the first way is filling the missing values by mean or median, in case of a numerical variable that is one of the standard way or that is one of the simplest way based on which you can fill in all the values. So, for any numerical variable you can look at the average of it or median of it, then you can have that value to fill in all the missing values.

Similarly, you can look at a model value for the categorical variable to fill in all the missing values in your data frame. The model value is nothing, but whichever category of the variable has the highest frequency; if that category occurs most frequently, than you can replace all the missing values with that category itself. So, these are the two ways of approaches that we going to see to fill the missing values here. As I mentioned there are several ways through which you will be able to fill in the missing values as well.

(Refer Slide Time: 10:59)



Imputing missing values

- Look at the description to know whether numerical variables should be imputed with mean or median

```
DataFrame.describe()
```

- Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values

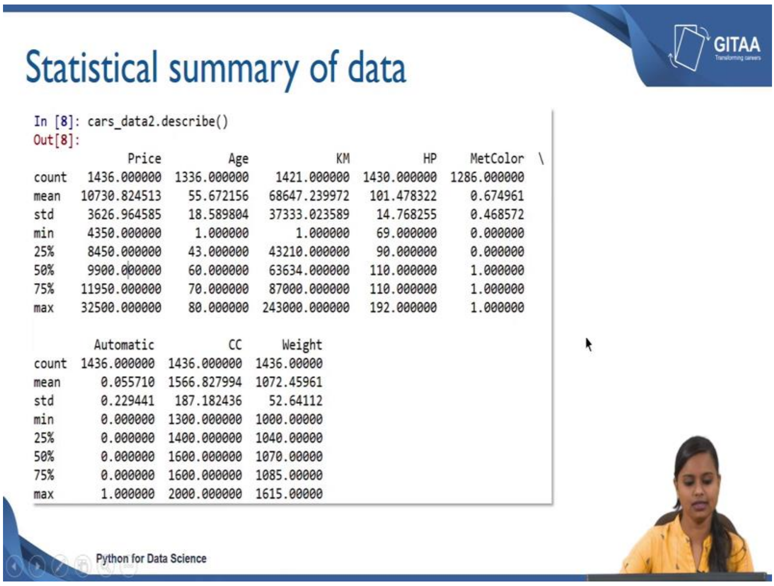
```
cars_data2.describe()
```

So, now we are going to impute the missing values; prior to that we have to look at the description to know whether numerical variable should be imputed with the mean value or the median value right. Because if you are going ahead and filling it filling the missing values with the mean value; that could also be a problem, because if you have any extreme values in your data that might cause you the mean value very high or very low. If you have very low value in your data that can tweak your mean value to the lowest value; but if you have an extreme value in the other end, like higher end then that can also mislead your mean value.

In that case you should always go for median, because a while calculating the median you basically sort all the values in ascending order and you will be taking exactly the middle value, if you have odd number of observation. But if you have a even number of observation, you will take the middle two; and then you take an average of it and then you will use that value as the median. We have to know about the distribution of the variables to basically decide on whether we are going to imputed with median or mean. So, dataframe.describe is the function that is used to get the descriptive statistics that summarizes the central .describe function basically.

So, there is a function called describe, you can use that for a data frame, so the syntax is the data frame.describe. The describe function basically generate descriptive statistics that summarize the central tendency dispersion and shape of data sets distribution; and that excludes all the NaN values and give you those statistics. So, now, we are going to check that using the describe function. So, this is how we use the function, we use the describe function followed by the data frame name.

(Refer Slide Time: 12:51)



Statistical summary of data

```
In [8]: cars_data2.describe()
Out[8]:
              Price          Age             KM           HP     MetColor  \
count   1436.000000  1336.000000    1421.000000  1430.000000  1286.000000
mean   10730.824513    55.672156   68647.239972   101.478322     0.674961
std     3626.964585    18.589804   37333.023589    14.768255     0.468572
min     4350.000000     1.000000       1.000000    69.000000     0.000000
25%     8450.000000    43.000000   43210.000000    90.000000     0.000000
50%     9900.000000    60.000000   63634.000000   110.000000     1.000000
75%    11950.000000    70.000000   87000.000000   110.000000     1.000000
max    32500.000000    80.000000  243000.000000   192.000000     1.000000

         Automatic           CC      Weight
count  1436.000000  1436.000000  1436.00000
mean      0.055710  1566.827994  1072.45961
std       0.229441   187.182436    52.64112
min       0.000000  1300.000000  1000.00000
25%       0.000000  1400.000000  1040.00000
50%       0.000000  1600.000000  1070.00000
75%       0.000000  1600.000000  1085.00000
max       1.000000  2000.000000  1615.00000
```

So, let us look at the output of it; the input being shown here and the output being shown here. So, if you look at the price variable, it basically gives you count, mean, standard deviation; from minimum to maximum it is called as the five number summary, because it gives you minimum maximum and the three quantiles of it. So, the count gives you how many number of observations are there under price; and the mean is nothing, but the average of the price.

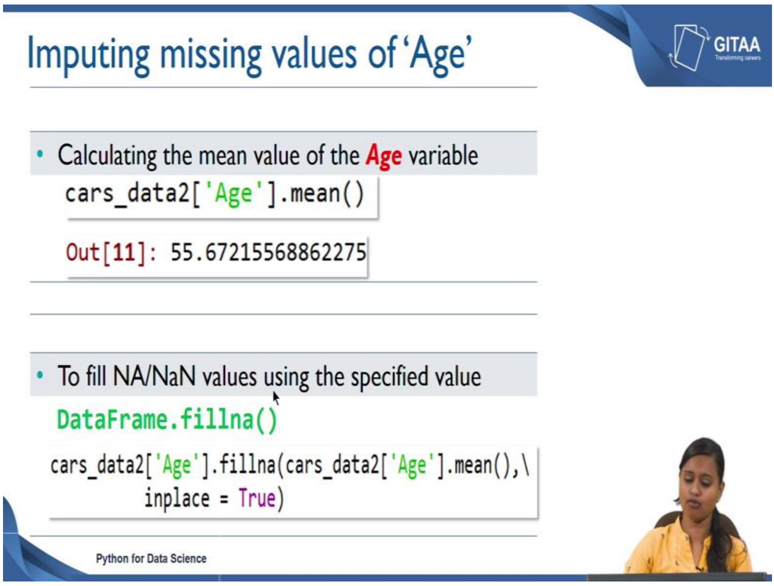
The average of the price of the car is 10730 Euros, and the standard deviation is around 3626, the minimum price of the car is 4350, and 25 percent represent that; 25 percentage of the price of the car is less than 8450, and 50 percentage of the cars price is less than 9900 Euros. And if you look at 75 percentage, then you can say that 75 percent of the cars price is less than 11950, and the maximum price of the car from the data frame that we have here is 32500.

So, when you look at the numerical variables the missing values are there under age, kilometer, and horsepower. So, if you look at the mean of the age, on an average the age of the car is around 55 months; and if you look at the median you can see that, median is also represented by 50 percent. So, you can say that and the value being here is 60; the mean and the median are closer, there is no much deviation from the mean. So, according to the age variable we can go ahead and imputed with the average age itself, instead of going ahead with the median value.

Next when we look at the kilometer, the average kilometer that the car has travelled is 68647 kilometers; and the median kilometer when the car has travelled is 63634 kilometers. But if you see there is a huge difference in the kilometers travelled, there is about 5 thousand and odd difference. So, in this case the mean is far away from the median. So, in this case you cannot go ahead and impute it with the mean, because that gives you a higher number that can be a reason because of any extreme values under the kilometer variable; rather we can go ahead with the imputation of median for the kilometer, because it basically gives us the exactly the middle value of it.

So, we can use median for kilometer. And then we can look at horsepower, the average horsepower of the car is 1000, the average horsepower of the car is 101; and the median horsepower of the car is 110. There is no much difference between 110 and 101, we can fill in the missing values using the mean value.

(Refer Slide Time: 15:54)



Imputing missing values of 'Age'

- Calculating the mean value of the Age variable

```
cars_data2['Age'].mean()
```

```
Out[11]: 55.67215568862275
```

- To fill NA/NaN values using the specified value

```
DataFrame.fillna()
```

```
cars_data2['Age'].fillna(cars_data2['Age'].mean(),\
        inplace = True)
```

Python for Data Science

So, now, let us try to impute the missing values of the age variable, so for that, as we know that since there is no much difference between the mean and the median, we are going to impute the age variable with the mean value of it.

So, let us just try to see, what the mean values for age variable. So, let us calculate the mean value of the age variable by using the mean function. And you can use the mean function along with those specified variables from a data frame. I am accessing the age variable from cars_data2; average age of the car is turned out to be 56 months old.

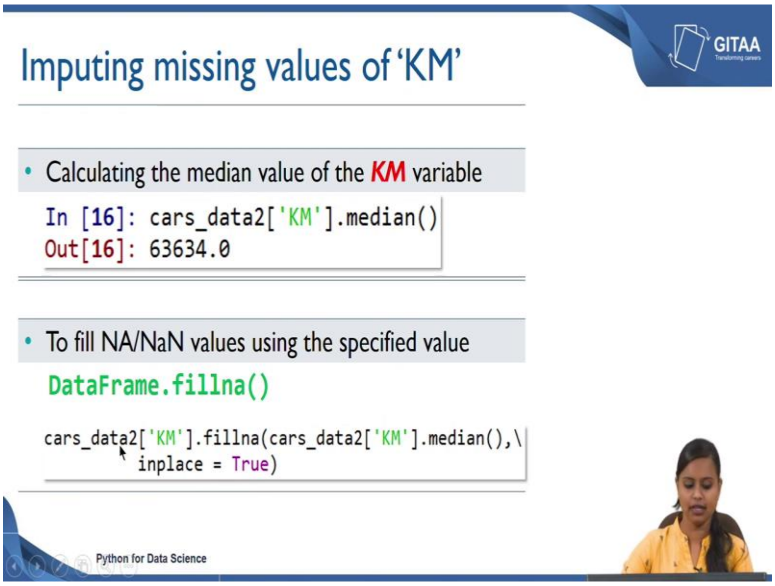
Now we are going to use this value to replace all the missing values under the variable age. So, to fill any NA or NaN values using a specified value, there is a function called fillna. Again there are so many functions that can be used to replace the missing values with a given value; but here I am illustrating you the example with fillna function, you can use that on a data frame.

So, the syntax being data frame.fillna, we will see how to use this function. Basically the function is used to fill in all the blank or NaN values with the given value. So, basically inside the fillna function you can specify the value with which you are going to replace all the missing values with. So, here I have given the function that needs to be used to calculate a value, and that value can be used to fill in all the missing values.

I have not given the exact value here, rather I have just journalize the function saying; so calculate the Age mean, calculate the mean of the age variable and then use that value to fill in all the missing values of the age variable. And what is the value, I have shown here that is 55.67.

Basically I wanted to fill in all the missing values on the data frame cars_data2. So, I have given inplace = true. I break the line here for the presentation purpose, you can continue in the same line also. So, now, we have replace the missing values under the age variable. So, now, we are going to impute the missing values of the variable kilometer.

(Refer Slide Time: 18:00)



Imputing missing values of 'KM'

- Calculating the median value of the ***KM*** variable

```
In [16]: cars_data2['KM'].median()
Out[16]: 63634.0
```

- To fill NA/NaN values using the specified value

```
DataFrame.fillna()
```

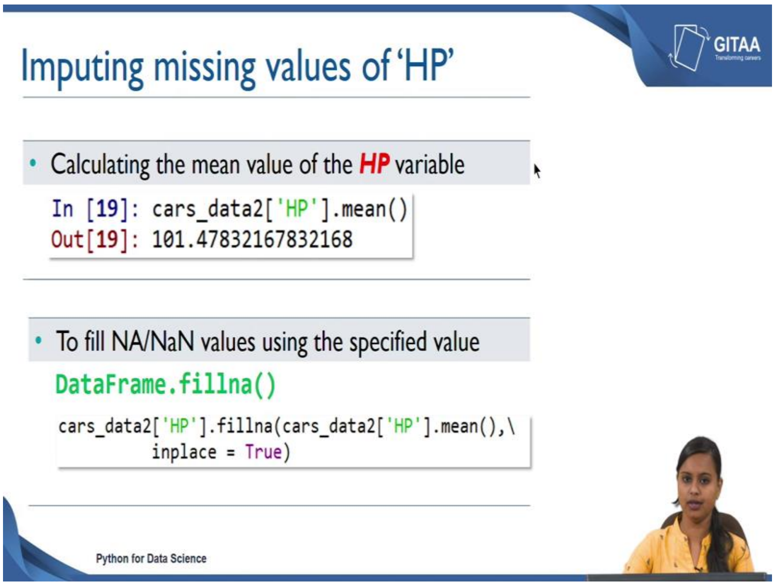
```
cars_data2['KM'].fillna(cars_data2['KM'].median(),\
                        inplace = True)
```

So, here as we saw from the kilometer variable the mean values really deviating from the median value that might be the reason, because they can be in extreme values under the kilometer variable. So, to get rid of that confusion we are going to use the median function. So, that it will give us the exact middle value of it. So, we are going to calculate the median value of kilometer to see what the median value is?

The median value you can calculated using the function median followed by the variable from a data frame and then you got a value called 63634; that means, that the median kilometer that the car has traveled is 63634 kilometers. So, now, we will use this value to replace all the missing values. So, we are going to use the same function that is fillna. So, I am going to apply the fillna function on to data frame called cars2 under the variable kilometer, and inside the function I have just given cars data2kilometer.median.

So, that the median value will be calculated from the kilometer and then that value will be used to replace all the missing values. And I have given inplace is equal true, so that all the missing values will be replaced in the existing data frame itself. So, now, we are going to impute the values, now we are going to impute the missing values of the horsepower variable.

(Refer Slide Time: 19:30)



Imputing missing values of 'HP'

- Calculating the mean value of the ***HP*** variable

```
In [19]: cars_data2['HP'].mean()
Out[19]: 101.47832167832168
```

- To fill NA/NaN values using the specified value

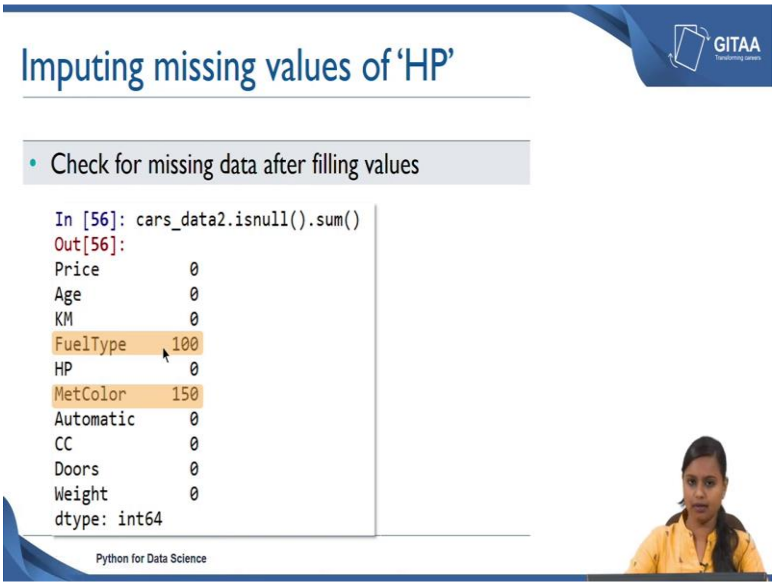
`DataFrame.fillna()`

```
cars_data2['HP'].fillna(cars_data2['HP'].mean(),\
              inplace = True)
```

So, if you look at the horsepower variable, we have seen that the mean and median are not very far away. The mean was closer to the median, so in that case we can go ahead and impute the missing values of the horsepower with the mean value. So, here I am calculating the mean of horsepower by accessing the horsepower variable from the data frame cars_data2.

And if you look at the output, the average horsepower of the car is 101.47 and odd. So, now, we are going to use this value to replace all the missing values of horsepower. So, using the same function fillna, I am replacing all the missing values using the mean of horsepower and we are filling the missing values in the existing data frame by giving inplace = true. Now, we have replaced the missing values under all the numerical variables like age, kilometer, and horsepower.

(Refer Slide Time: 20:25)



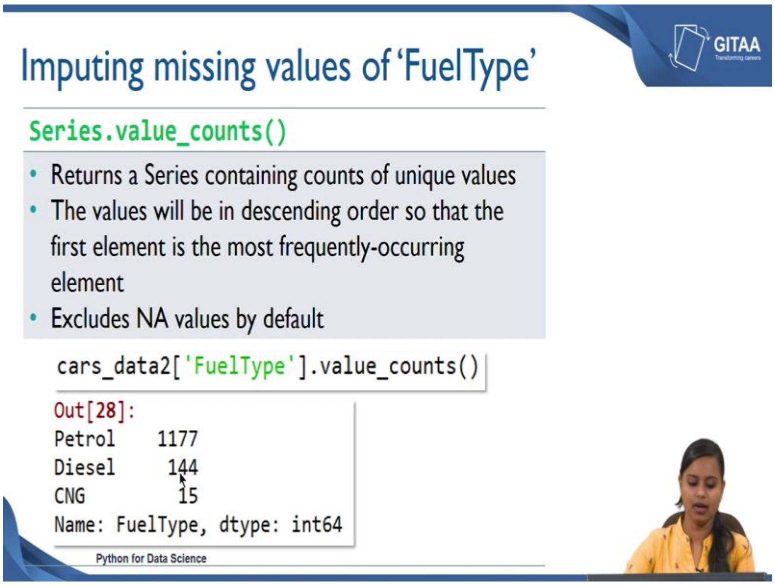
Imputing missing values of 'HP'

- Check for missing data after filling values

```
In [56]: cars_data2.isnull().sum()
Out[56]:
Price          0
Age            0
KM             0
FuelType     100
HP             0
MetColor     150
Automatic      0
CC             0
Doors          0
Weight         0
dtype: int64
```

So, now let us check for the missing data after filling those values. So, now, we have replace the missing values under age, kilometer, and the horsepower; let us see whether there are any missing values under that or not, yes there are no missing values. But if you can see there are still missing values under FuelType and MetColor, because we have not still touched on that variables. So, now we are going to impute the missing values of the categorical variables.

(Refer Slide Time: 20:52)



Imputing missing values of 'FuelType'

Series.value_counts()

- Returns a Series containing counts of unique values
- The values will be in descending order so that the first element is the most frequently-occurring element
- Excludes NA values by default

```
cars_data2['FuelType'].value_counts()
```

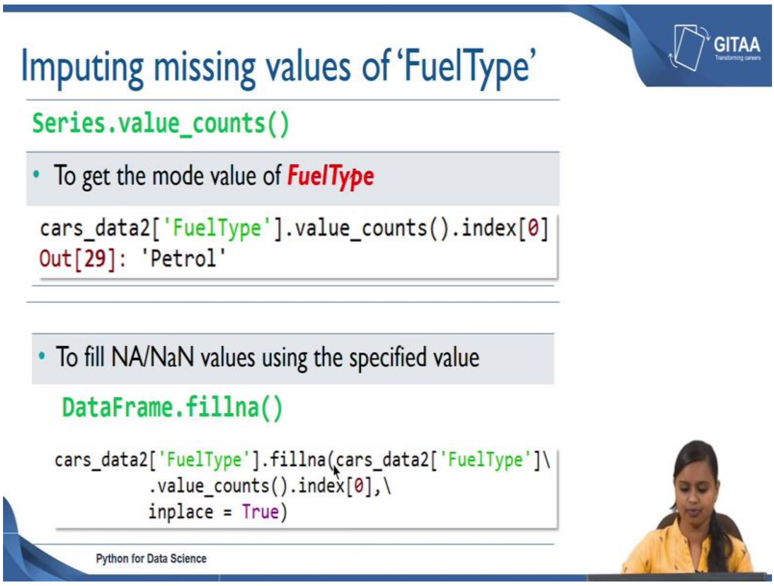
```
Out[28]:
Petrol    1177
Diesel     144
CNG         15
Name: FuelType, dtype: int64
```

So, as I mentioned earlier for any categorical variable, the simplest way to fill in the missing values of categorical variable is to check for the most frequently occurring category and then replacing that with the missing values. So, for that we need to understand what are the categories are there under the FuelType, right; and we need to understand the corresponding frequencies as well. So, for that there is a function called value_counts that can be applied on to a series.

And the value_counts basically returns a series containing counts of unique values. The output of the function will give you the values in the descending order, so that the first element is the most frequently occurring element and it gives you the frequencies of each category by excluding all the NA values by default. So, we do not need to remove the missing values and then get the frequencies of it, by default it excludes all the missing values.

So, this is how we use the function. cars_data2 is the data frame name, under that I am accessing the variable called FuelType; by giving.value_counts you will get an output which is shown here. For example, the petrol has the highest frequency that is 1177 observations are of petrol FuelType, and the only 144 cars have diesel FuelType, and only 15 cars has CNG FuelType. In this case it is very clear that the model value would be petrol; if you want to get the model value separately, then you can also do that, you can also use the same function you can give the index as 0.

(Refer Slide Time: 22:25)



Imputing missing values of 'FuelType'

Series.value_counts()

- To get the mode value of *FuelType*

```
cars_data2['FuelType'].value_counts().index[0]
Out[29]: 'Petrol'
```

- To fill NA/NaN values using the specified value

DataFrame.fillna()

```
cars_data2['FuelType'].fillna(cars_data2['FuelType']\
            .value_counts().index[0],\
            inplace = True)
```

Python for Data Science

.

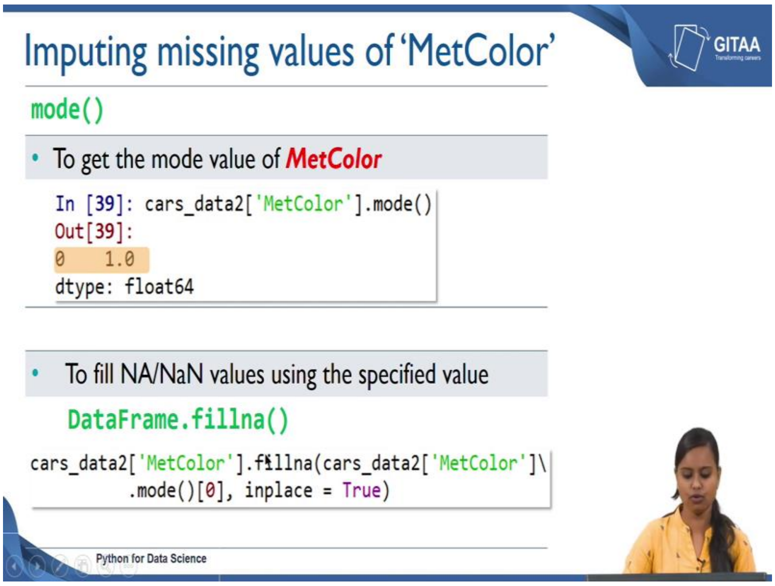
Since in python the indexing starts from 0, the 0 will give you the first value of the output of value_count. What would be the first value of the value_counts output? It would be petrol, because since it has the highest number of observation it was shown at the top. So, when you access it using the index 0, you will get a value called petrol.

Now you will be able to use this value to fill all the missing values of FuelType variable. Let us see how to do that; you can use the same function that is.fillna. I am using the same function from the cars data2data frame; I am accessing the variable FuelType. And I am applying the fillna function on it, saying that and inside the function we need to give a value, so that the missing value will get replaced.

Here I am not giving petrol directly, rather I am giving the whole function here, so that the code will be generalized; it is you do not need to tweak it whenever you want it, because just to write a generalized code. So, here I am filling all the missing values with the most frequently occurring category of FuelType. So, petrol will be used to fill in all the missing values of the variable FuelType. And you know here, we have used inplace = true, so that we are replacing the existing data frame itself.

Now, by using fillna we have replace the FuelType missing values with the petrol FuelType, we are going to now consider the MetColor variable.

(Refer Slide Time: 23:58)



Imputing missing values of 'MetColor'

mode()

- To get the mode value of ***MetColor***

```
In [39]: cars_data2['MetColor'].mode()
Out[39]:
0    1.0
dtype: float64
```

- To fill NA/NaN values using the specified value

DataFrame.fillna()

```
cars_data2['MetColor'].fillna(cars_data2['MetColor']\
          .mode()[0], inplace = True)
```

Python for Data Science

Here also we are going to impute all the missing values of MetColor with a model value of the variable MetColor. MetColor basically the metallic color of the car, basically it will have a two values. As we know the MetColor has two values; 0 represents the car does not have a metallic colour, and 1 represents the car has a metallic color. So, there are several ways using which you will be able to fill in the model value. In the previous example, we have seen how to fill in the missing values using value_counts and accessing it from the index.

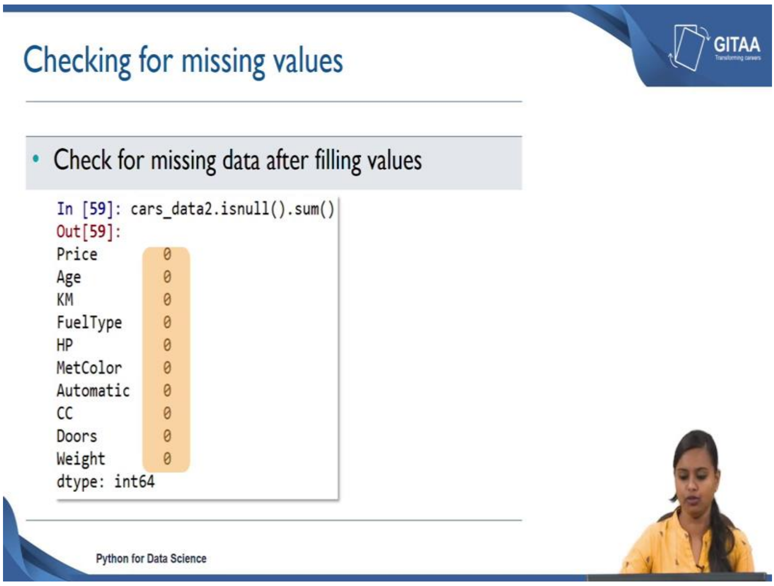
Now, we are going to use the mode function to calculate the model value of the categorical variable. So, to get the mode value of the MetColor, we can use it as data frame and access the respect to variable and then use the function.mode. The.mode function gives you an output which is shown here, it basically gives the value along with the index. The mode value of the MetColor is 1.0, since it has missing values it is giving you an floating points that is 1.0, ideally it should be 1. And the index of the value is 0. So, why you are also getting in index, while calculating the median or mean you would not be able to get the index.

Because median or mean can only be a scalar value; but mode cannot be a scalar value or mode need not be a scalar value. Mode can either be bimodal, there are many cases where a variable can have bimodal or more than two models, so in that case you will have continuous index. In this case, since our variable MetColor has only one value as more you are getting only 0. So, now, to fill in all the missing values using the specified value, we use the function.fillna. So, now, I have use the fillna function on to a variable metallic color that is from the data frame cars_data2; and inside the function I have specified that, calculate the mode from metallic color variable and replace all the missing values.

So, here I have also given the index to it, that is just because the output always comes with the index; if you have bimodal, then you will always have another index as 1 and you will have another value, that will be the case where you have a both the categories have the same frequency. In that case you can choose any one of the value which is required for the analysis; but here which is more important for the analysis. But here, since I am very clear that I have only one value and that the model values index is 0, I have given it has 0. And I am making all the modifications onto the same data frame called data2, so I have given inplace = true.

So, now we have replaced both FuelType and MetColor variable. Now we have replaced combinedly both numerical as well as categorical variables using different approaches.

(Refer Slide Time: 27:00)



Checking for missing values

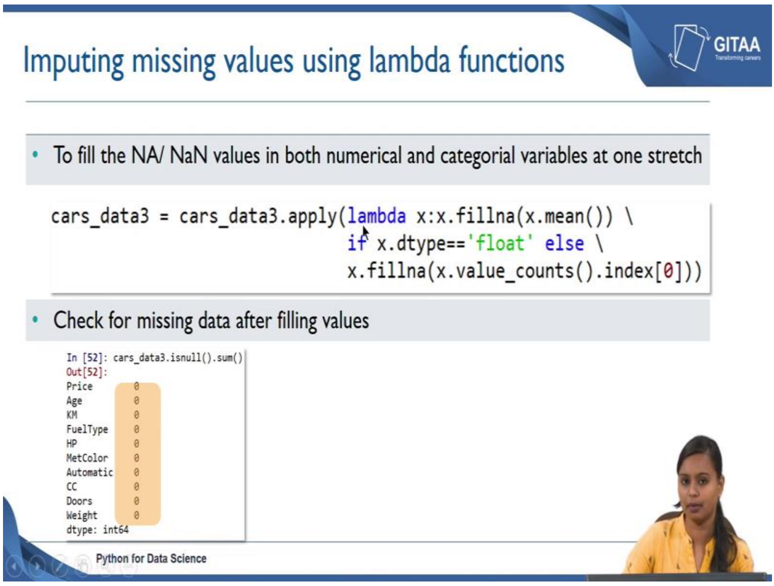
- Check for missing data after filling values

```
In [59]: cars_data2.isnull().sum()
Out[59]:
Price        0
Age          0
KM           0
FuelType     0
HP           0
MetColor     0
Automatic    0
CC           0
Doors        0
Weight       0
dtype: int64
```

So, let us check for the missing data after filling in all the values, using the isnull.sum function that has to be applied on to the data frame. In a data frame cars_2 check whether there are any missing values. If there are any missing values get the sum of it under each column that is what the function describes. And if you see here none of the columns have missing values now; since we have already imputed all the missing values with the logical approach that we have imputed for all of the variable separately using a same logic, right.

For example for a numerical variable we have imputed with the mean or median, and for all categorical variables we have imputed with a model value. So, this can be the case where you have only missing values in 6 to 7 columns. What will you do whenever you have missing values in multiple columns like 50 to 60? In that case there can be a simple function which will do the imputation in one shot; we are going to use that.

(Refer Slide Time: 28:00)



Imputing missing values using lambda functions

- To fill the NA/ NaN values in both numerical and categorial variables at one stretch

```
cars_data3 = cars_data3.apply(lambda x:x.fillna(x.mean()) \
                              if x.dtype=='float' else \
                              x.fillna(x.value_counts().index[0]))
```

- Check for missing data after filling values

```
In [52]: cars_data3.isnull().sum()
Out[52]:
Price        0
Age          0
KM           0
FuelType     0
HP           0
MetColor     0
Automatic    0
CC           0
Doors        0
Weight       0
dtype: int64
```

.

So, to fill the missing values that is NaN values in both numerical and categorical variables at one shot we are going to use a lambda function. Apply functions can be used whenever you want to perform any operations column wise or row wise, it can be used any other cases; but here, this apply function will be used across columns.

And if you see here, I am not using the cars_data2; because the cars_data2 does not have any missing values now, since we have already replaced everything. If you can recall, I have created another copy from the cars_data while reading itself. In that case the cars_data3 is still have missing values, because we have not touch this data at all. So, I am using that data set and I am using and apply function, so that whatever function that I am giving inside the apply function that will be applied across all the columns.

And the function that I am using inside the apply function is a lambda function; lambda function is an anonymous function and a powerful function whenever you use it inside a function. And here I am using it inside an apply function; and I am defining a function call x, and I am using a predefined function called fillna to apply it across all the columns. So, what it basically does means? fillna we know that, wherever there are missing values it will replaced with the given value that we are giving it inside the function.

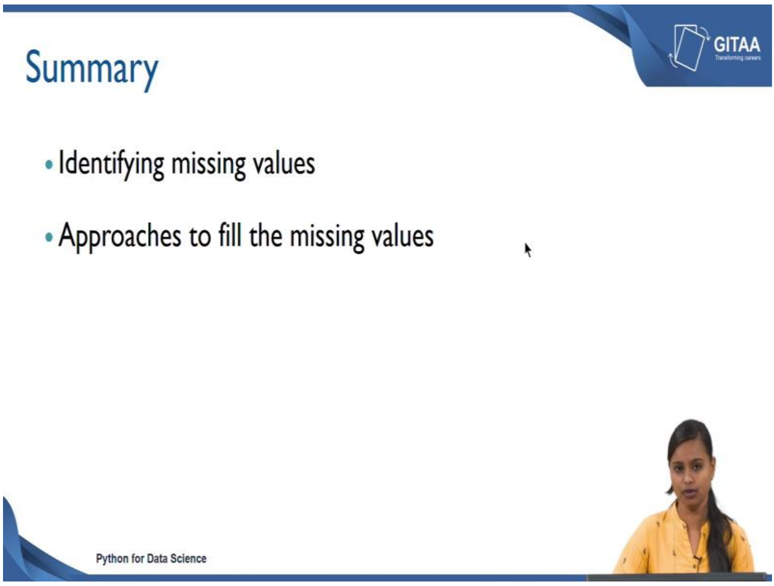
So, what value that we are giving here? We are not giving a value, rather we are giving conditions to it; we are giving if else conditions, like basically our target is to fill in all the missing values of numerical variable with mean, and to fill in all the missing values of categorical variable with mode. In that case, give a condition saying that calculate the mean of each and every variable, if the various data type is a float; else calculate the mode value of each and every variable and then use that value to replace with.

So, that is what the function does. So, using this function we have just replaced all the missing values at one shot; wherever there are missing values in numerical variable everything has been replaced with the mean value of each and every variable. And similarly wherever there are missing values under any categorical variable, for each and every categorical variable separately the model value will be calculated and then that will be used for filling those missing values.

So, once we do that, let us check for missing data after filling all the values. So, I am checking whether there are any missing values under the data frame cars_data3. The output is also shown here, if you see all the values are 0 here; because we have already imputed all the missing values that is why you are seeing all zeros here.

So, now we do not have any missing values under any variable; but as I mention this is not the only way that you can go ahead with the imputing with missing values. There are several approaches that are available to impute all the missing values, the way that we have shown here is the simplest of all by just imputing it with mean or the model value.

(Refer Slide Time: 31:16)



Summary

- Identifying missing values
- Approaches to fill the missing values

.

Let us summarize whatever we seen till here; we have seen how to identify the missing values, you also seen how you can identify the pattern of your missing values by subsetting all the rows and seeing whether one row has only one column value missing or multiple columns are missing; and then by arriving at a decision that the missing values are there randomly under each column; then we found out an approach to fill in all the missing values.

We have seen two approaches that is mean or median imputation and mode imputation; mean or median imputation is only for a numerical variable and you use mode imputation for a categorical variable.

Thank you.