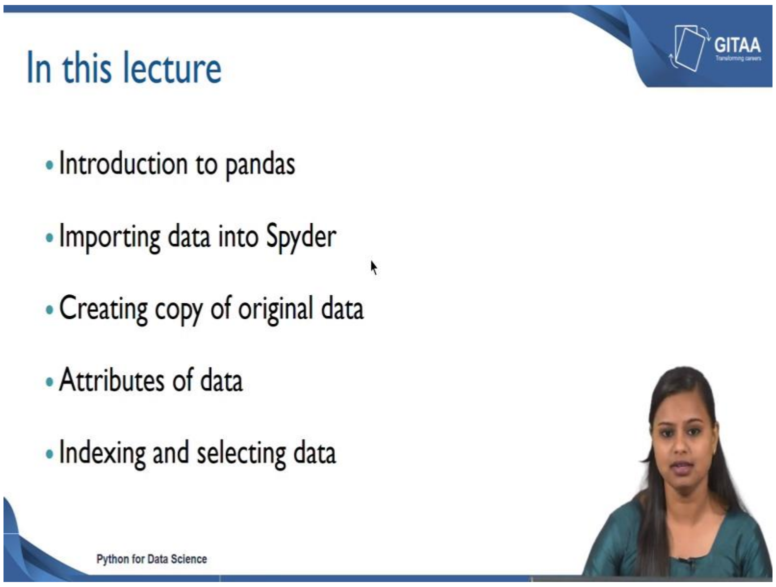
Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 14
Pandas Dataframes
Part-I

(Refer Slide Time: 00:19)

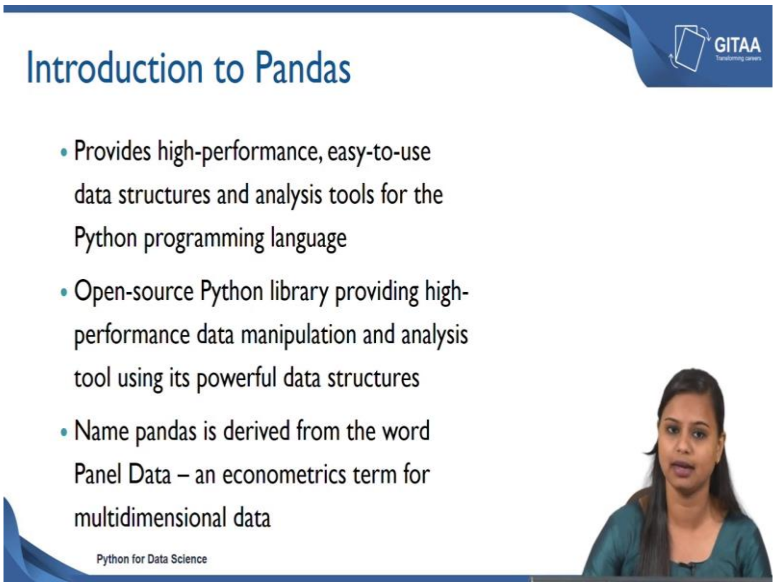


In this lecture

- Introduction to pandas
- Importing data into Spyder
- Creating copy of original data
- Attributes of data
- Indexing and selecting data

Hello all, welcome to the lecture on Pandas Dataframes. In this lecture, we are going to see about the pandas library. Specifically we are going to look at the following topics. First we will get introduced to the pandas library; after that we are going to see how to import the data into Spyder. We will be looking at how to create a copy of original data. We will also be looking at how to get the attributes of data; followed by that we will see or how to do indexing and selecting data.

(Refer Slide Time: 00:44)



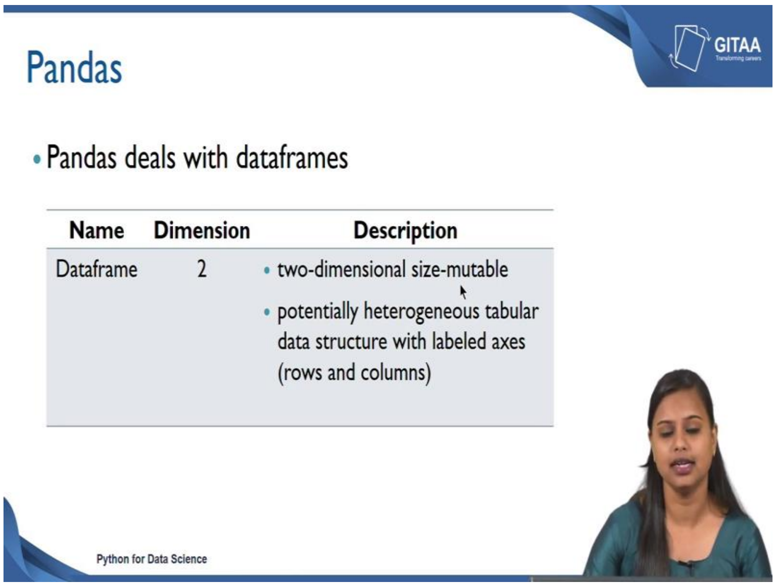
Introduction to Pandas

- Provides high-performance, easy-to-use data structures and analysis tools for the Python programming language
- Open-source Python library providing high-performance data manipulation and analysis tool using its powerful data structures
- Name pandas is derived from the word Panel Data – an econometrics term for multidimensional data

Python for Data Science

To introduce you to the pandas library, it provides high performance, easy to use data structures, and analysis tools for the python programming language. It is an open source python library which provides high performance data manipulation and analysis tool using its powerful data structures. And, it is also considered one of the powerful data structures when compared with other data structures because of its performance and data manipulation techniques it is available in pandas. And, the name pandas is derived from the word panel data which is an econometrics term for multi dimensional data.

(Refer Slide Time: 01:20)



Pandas

- Pandas deals with dataframes

Name	Dimension	Description
Dataframe	2	• two-dimensional size-mutable • potentially heterogeneous tabular data structure with labeled axes (rows and columns)

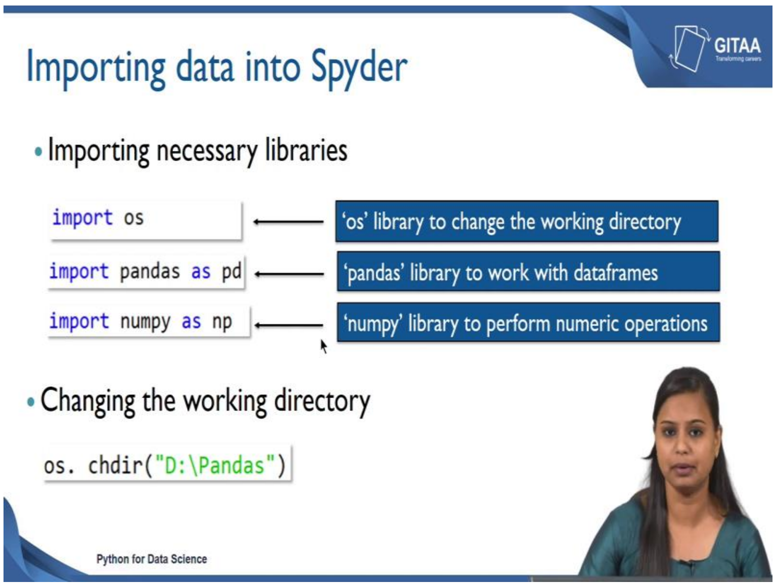
Python for Data Science

And here is the description of about the dataframe. Dataframe consist of two dimensions; the first dimension is the row and the second dimension is the column that is what we mean by two-dimensional and size-mutable. And whenever we say dataframe, dataframe is a collection of data in a tabular fashion, and the data will be arranged in rows and columns. Where we say each row represent a sample or a record, and each column represent a variable. The variable in the sense the properties, that are associated with each sample. The second point is that potentially heterogeneous tabular data structure with labelled axes.

Heterogeneous tabular data structure in the sense whenever we read a data into spyder, it becomes a dataframe and each and every variable gets a data type associated with that whenever you read it. We do not need to explicitly specify the data type to each and every variables, and that is basically based on the data or the type of data that is contained in a each variable or a column. And we mean labelled axes each and every row and columns will be labelled, row labelling the index for each rows which is starting from 0 to n-1, and the labels for column in the sense the names for each variables those are called labelled axes.

So, whenever we say labelled axes, the row labels are nothing but the row index and the column labels are nothing, but the column names, and this is about the basic description of the dataframe.

(Refer Slide Time: 02:56)



Importing data into Spyder

- Importing necessary libraries

```
import os
import pandas as pd
import numpy as np
```

'os' library to change the working directory
'pandas' library to work with dataframes
'numpy' library to perform numeric operations

- Changing the working directory

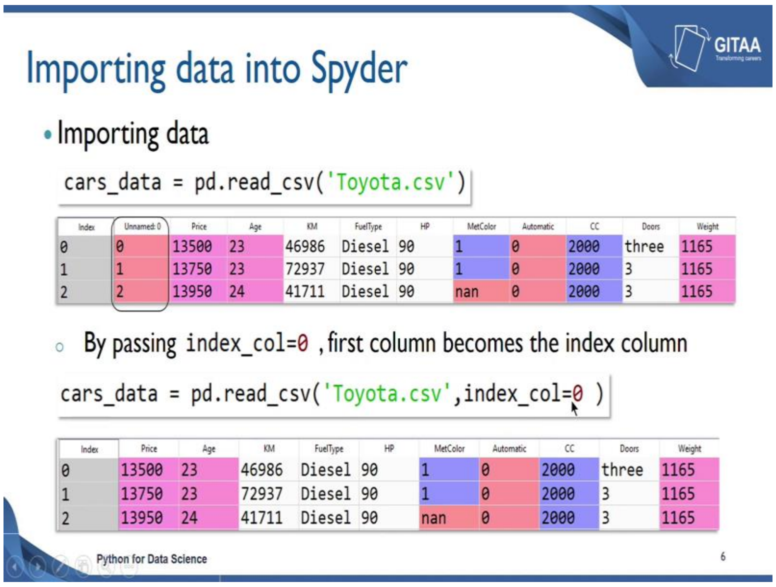
```
os. chdir("D:\Pandas")
```

So, next we will see how to import the data into Spyder. In order to import data into Spyder, we need to import necessary libraries; one of it is called os. Whenever you import any library, we use the command import, and OS is the library that is used to change the working directory. Once you open your Spyder, the default working directory will be wherever you have installed your python, and we import os to basically change the working directory, so that you will be able to access the data from your directory.

Next we are going to import the pandas library using the command input pandas as pd. Pd is just an alias to pandas. So, whenever I am accessing or getting any functions from pandas library, I will be using it as pd. And we have imported pandas to work with dataframes. We are also importing the numpy library as np to perform any numerical operations. Now, we have imported the library called os. And chdir is the function which is used to set a path from which you can access the file from. And inside the function, I have just specified my path wherever the data that I am going to import into Spyder is like my data is in the D drive under the folder pandas.

So now, this is how we change or set the working directory. Once we set the working directory, we are set to import any data into Spyder.

(Refer Slide Time: 04:32)



Importing data into Spyder

- Importing data

```
cars_data = pd.read_csv('Toyota.csv')
```

Index	Unnamed: 0	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
0	0	13500	23	46986	Diesel	90	1	0	2000	three	1165
1	1	13750	23	72937	Diesel	90	1	0	2000	3	1165
2	2	13950	24	41711	Diesel	90	nan	0	2000	3	1165

  - By passing `index_col=0` , first column becomes the index column

```
cars_data = pd.read_csv('Toyota.csv',index_col=0 )
```

Index	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
0	13500	23	46986	Diesel	90	1	0	2000	three	1165
1	13750	23	72937	Diesel	90	1	0	2000	3	1165
2	13950	24	41711	Diesel	90	nan	0	2000	3	1165

Python for Data Science 6

So, now we will see how to import the data into Spyder. So, to import the data into Spyder, we use the command read_csv. Since we are going to import a csv file and the read_csv is from the library pandas, so I have used pd.read_csv. And inside the function, you just need to give the file name within single or double quote and along with the extension.csv. And I am saving it to an object called cars_data. So, once I read it and save it to an object, my cars_data becomes the dataframe.

And once you read it, you will get the details in the environment tab, where you will see the object name, the type of the object and number of elements under that object. And once you double click on that object or the dataframe, you will get a window where you will be able to see all the data that is available from your Toyota file. This is just a snippet of three rows with all the columns. And I have multiple variables here first being the index. Whenever you read any dataframe into Spyder, the first column will be index; it is just the row labels for all the rows.

The next is the unnamed colon zero column. According to our data, we already have a column which serves the purpose for row labels. So, this is just an unwanted column. And next being the price variable which describes the price of the cars, because this data is about the details of the cars, and the properties that are associated with each cars. So, the each rows represents the car details, the car details being price, age, kilometre, fuel type, horsepower and so on.

First let us look at what each variable means price being price of the car all the details are about the pre owned cars. Next being the age of the car, and the age is being represented in terms of months and the kilometre, how many kilometre that the car has travelled, the fuel type that the car possess, one of the type is diesel, next being the horsepower. And we have another variable called MetColor that basically represents whether the car has a metallic colour or not; 0 means the car does not have a metallic colour and 1 means the car does have a metallic colour.

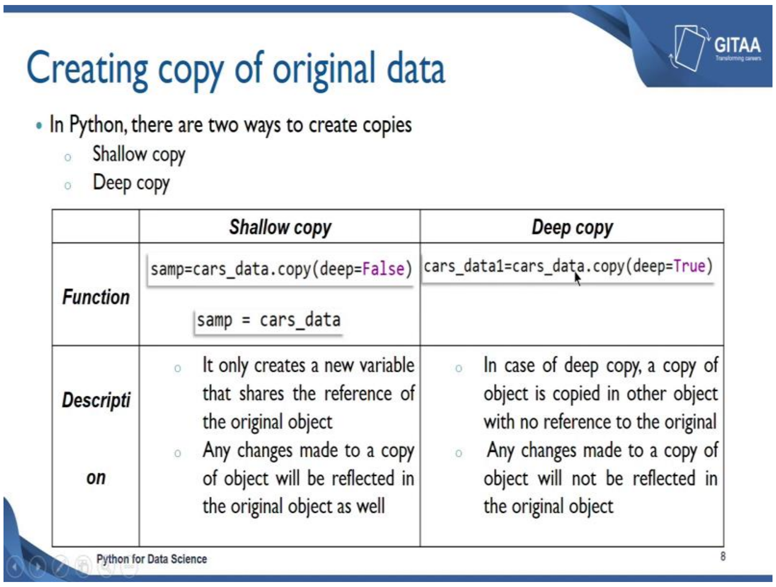
And next being automatic, what is the type of gearbox that the car possess; if it is automatic, it will be represented as 1; and if it is manual, it will be represented as 0. Next is being the CC of the car, and the doors represents how many number of doors that the car has, and the last being the weight of the car in kgs. So, this is just a description of all the variables in the Toyota csv. And we have also found out that there are two columns which serves the purpose for row labels, instead of having two columns we can remove either one of it, index is the default one. So, we can remove unnamed colon zero column.

So, how to get rid of this? Whenever you read any csv file by passing index_column = 0, the first column becomes the index column.

So now, let us see how to do that. So, whenever we read the data using the read_csv, we can just add another argument called index_column = 0. And the value 0 represents which column you should treat it as a index. I need the first column should be treated as the index. So, basically I have renamed, unnamed colon 0 to index. So, if you use 1 here, then price will be treated as row index. You will get the column name as index, but all the values will be the price column values.

So, but I do not want that since I already have a column which is in the name of unnamed, I am using that column as my index column. So, whenever I use index_column = 0. The first column will be treated as index column.

(Refer Slide Time: 08:47)



Creating copy of original data

- In Python, there are two ways to create copies
  - Shallow copy
  - Deep copy

	Shallow copy	*Deep copy*
Function	`samp=cars_data.copy(deep=False)` `samp = cars_data`	`cars_data1=cars_data.copy(deep=True)`
Description	◦ It only creates a new variable that shares the reference of the original object ◦ Any changes made to a copy of object will be reflected in the original object as well	◦ In case of deep copy, a copy of object is copied in other object with no reference to the original ◦ Any changes made to a copy of object will not be reflected in the original object

Python for Data Science 8

So, now we know how to import the data into Spyder. Let us see how to create the copy of original data, because there might be cases where we need to work with the copy of the data without doing any modifications to the original data. So, let us see in detail about how we can create a copy of original data. So, in python there are two ways to create copies, one is shallow copy and another one is deep copy.

First let us look at the shallow copy. The function row represents how to use the function, and the description represents what does that function means. So, in shallow copy, you can use the.copy function that can be accessed whenever you have a dataframe. Since, I have cars_data as a dataframe, I can use.copy. If you want to do a shallow copy, you can use deep = false by default the value will be true.

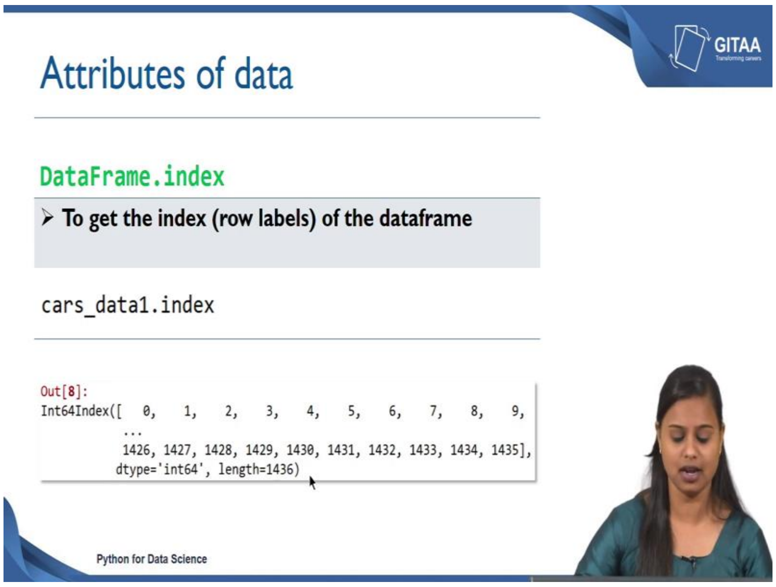
So, there are two ways to do a shallow copy, one is by using the.copy function, another one is by just assigning the same data frame into a new object. I have assigned cars_data as samp using the assignment operator. So, this also means you are doing a shallow copy. So, what shallow copy means in the sense basically if you are doing a shallow copy, it only creates a new variable that shares the reference of the original object; it does not create a new object at all. Also any changes made to a copy of object, it will be reflected in the original object as well.

So, whenever you want to work with the mutable object, then you can do a shallow copy, where all the changes that you are making into samp will be reflected in your cars_data. Now, let us see about the deep copy. To do a deep copy, we use a same command.copy, but we said the deep as true. And by default the deep value will be true. So, whenever you use.copy, you are doing a deep copy. As you see I am doing a deep copy and by creating a new object called cars_data1, where cars_data1 is the copy of the original data cars_data.

And what deep copy represents means in case of a deep copy, a copy of object is copied in another object like the copy of cars_data is being copied in another object called cars_data with no reference to the original. And whatever changes you are making it to the copy of object that will not be reflected in the original object at all. Whatever modifications you are doing it in cars_data1 that will be reflected in that dataframe alone, the original dataframe will not get affected by your modifications.

So, there are two cases, you can choose any of the copies according to the requirements. Whenever you want to do any modifications and reflect back to the original data, in that case we can go for shallow copy. But if you want to keep the original data untouched and whatever changes you are making that should be reflected in the copy alone, then in that case you can use a deep copy.

(Refer Slide Time: 11:58)



Attributes of data

DataFrame.index

- To get the index (row labels) of the dataframe

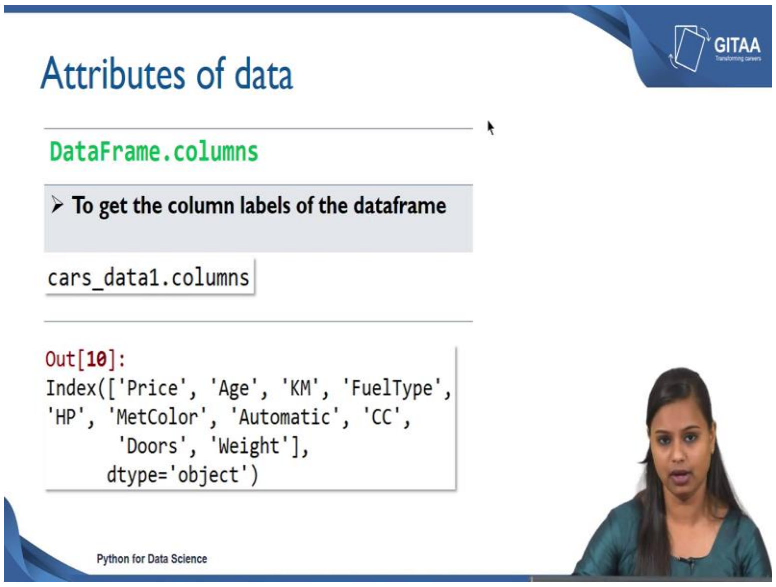
```
cars_data1.index
```

```
Out[8]:
Int64Index([   0,    1,    2,    3,    4,    5,    6,    7,    8,    9,
            ...
            1426, 1427, 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435],
           dtype='int64', length=1436)
```

So, now we will see how to get attributes of data, attributes in the sense getting the basic informations out of data, one of it is called getting the index from the dataframe. So, the syntax being dataframe.index,.index can be used whenever you have a dataframe. So, to get the index, index means row labels here. Whenever you want to get the row labels of the data frame, you can use data frame.index.

Here dataframe being cars_data1, and I am using.index function that will give me the output for the row labels of the dataframe. If you see the row labels is ranging from 0 to 1435 where the length is 1436. So, the indexing in python starts from 0 to n-1 here. So, this is how we get row labels from the dataframe.

(Refer Slide Time: 12:50)


Attributes of data

`DataFrame.columns`

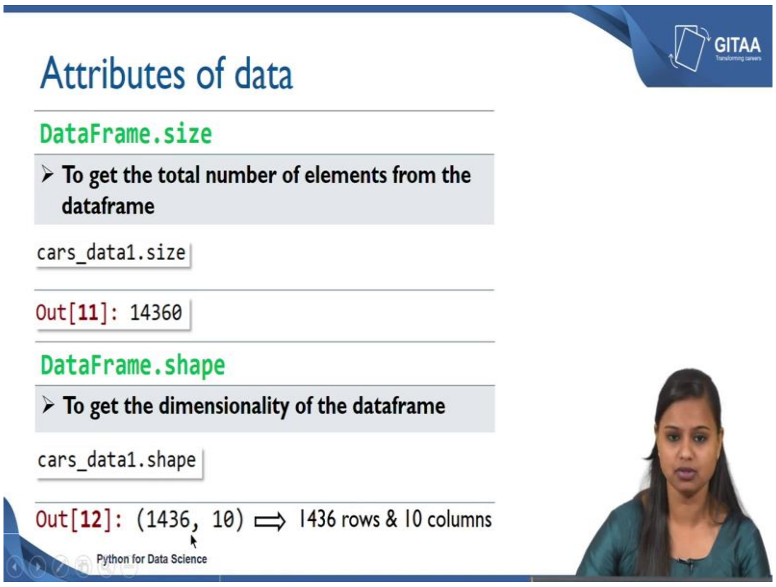
➢ To get the column labels of the dataframe

```
cars_data1.columns
```

```
Out[10]:
Index(['Price', 'Age', 'KM', 'FuelType',
'HP', 'MetColor', 'Automatic', 'CC',
       'Doors', 'Weight'],
      dtype='object')
```

Next we will see about how to get the column names of the dataframe. You can get the column labels of the dataframes using.columns. So, cars_data1.columns will give you all the column names of your dataframe. Basically the output is just an object which is a list of all the column names from the dataframe cars_data1. By getting the attributes of the data like the row labels and the column labels, you will be able to know from which range your row labels are starting from and what are all the column names, that you have in your dataframe.

(Refer Slide Time: 13:29)


Attributes of data

`DataFrame.size`

➢ To get the total number of elements from the dataframe

```
cars_data1.size
```

```
Out[11]: 14360
```

`DataFrame.shape`

➢ To get the dimensionality of the dataframe

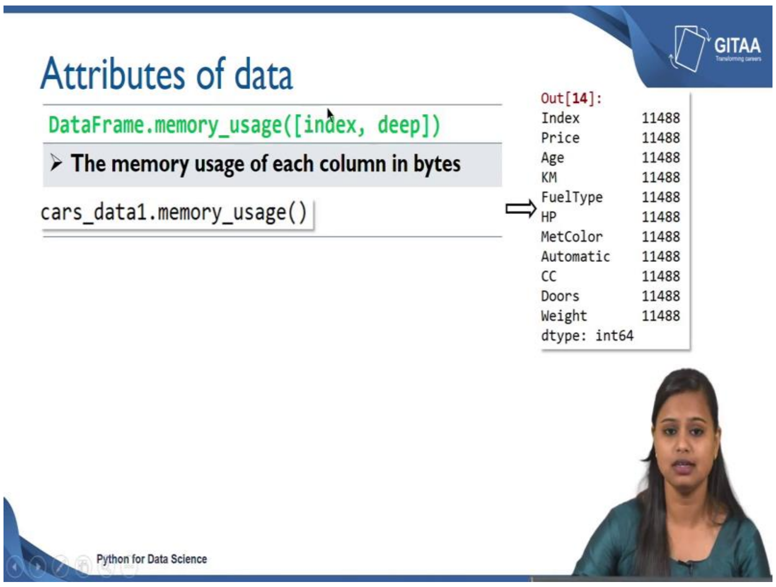
```
cars_data1.shape
```

```
Out[12]: (1436, 10)
```
⟹ 1436 rows & 10 columns

Next we can also get the size that is we can also get the total number of elements from the dataframe using the command.size. Here this is just the multiplication of 1436 into 10, where 1436 rows are there and 10 columns are there. So, when you multiply that you will get the total number of elements that is what the output represents, you can also get the shape or the dimensionality of the dataframe using the command.shape.

So, cars_data1.shape will give you how many rows are there and how many columns are there explicitly. The first value represents rows 1436 rows are there, and 10 columns are there. So, you will be able to get the total number of elements as well as how many number of rows, and how many number of columns are there separately also.

(Refer Slide Time: 14:20)



Attributes of data

```
DataFrame.memory_usage([index, deep])
```

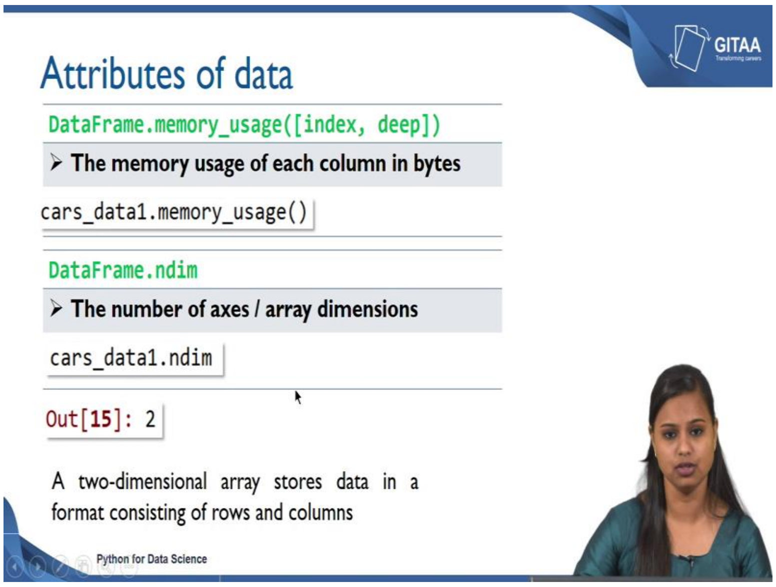
➢ The memory usage of each column in bytes

```
cars_data1.memory_usage()
```

```
Out[14]:
Index        11488
Price        11488
Age          11488
KM           11488
FuelType     11488
HP           11488
MetColor     11488
Automatic    11488
CC           11488
Doors        11488
Weight       11488
dtype: int64
```

So, next we will see about the memory usage of each column in bytes. So, to get the memory usage of each column in bytes, so we use the command.memory_usage, and the.memory_usage will give you the memory used by each column that is in terms of bytes. So, if you see all the variables has used the same memory, there is no precedents or there is no higher memory that is used for any particular variable. All the variable has used the same memory and the data type that you are seeing here is the data type of the output.

(Refer Slide Time: 14:57)



Attributes of data

```
DataFrame.memory_usage([index, deep])
```

➢ The memory usage of each column in bytes

```
cars_data1.memory_usage()
```

```
DataFrame.ndim
```

➢ The number of axes / array dimensions

```
cars_data1.ndim
```

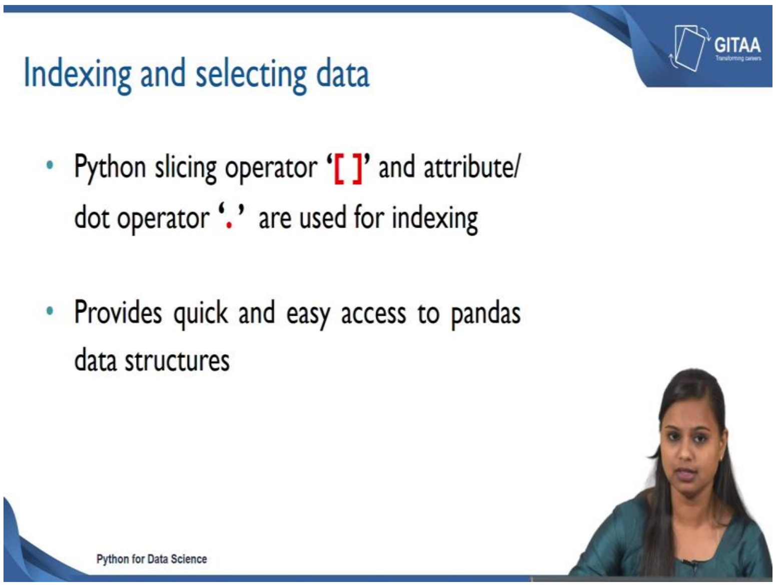
```
Out[15]: 2
```

A two-dimensional array stores data in a format consisting of rows and columns

Next, how to get the number of axes or the array dimensions. Basically to check how many number of axes are there in your dataframe, you can get that using.ndim function. So, I have used cars_data1.ndim that will give you how many number of axes are there that is basically how many number of dimensions that are available for your dataframe. It basically the output says as to because the cars_data1 has two dimension, one dimension being rows and the other dimension being columns.

All the rows forms one dimension, and all the columns forms the other dimension. And just because we have multiple variables, it does not mean that we have multi dimension to your data. The data frame just consist of two dimensions. So, it becomes a two-dimensional data frame. And if you see a two-dimensional array stores a data in a format consisting of rows and columns, that is why our dataframes dimension is 2.

(Refer Slide Time: 16:04)

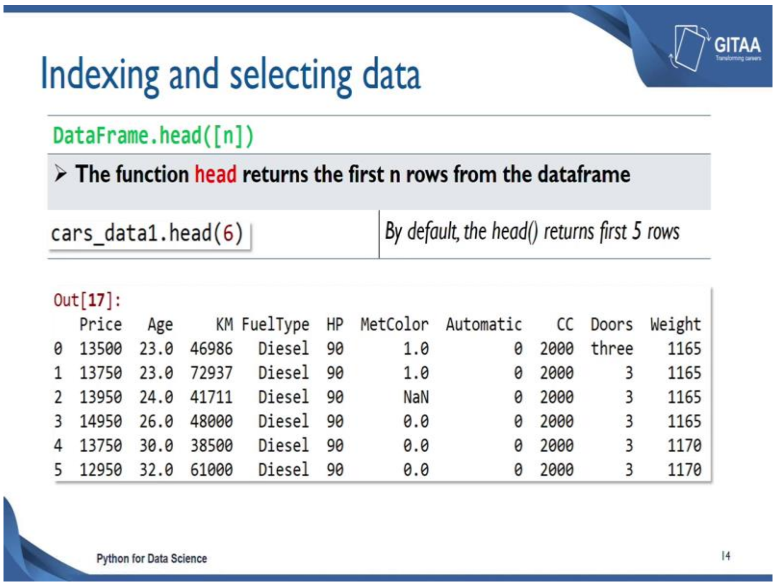

Indexing and selecting data

- Python slicing operator '[ ]' and attribute/ dot operator '.' are used for indexing
- Provides quick and easy access to pandas data structures

Python for Data Science

So, next we will see how to do indexing and selecting data. The python slicing operator which is also known as the square braces and attribute called.operator which is being represented as a period, that are used for indexing. And, indexing basically provides quick and easy access to pandas data structures. Whenever you want to index or select any particular data from your dataframe, the quick and easy way to do that will be using the slicing operator and a dot operator.

(Refer Slide Time: 16:40)


Indexing and selecting data

DataFrame.head([n])

➢ The function head returns the first n rows from the dataframe

cars_data1.head(6)	*By default, the head() returns first 5 rows*

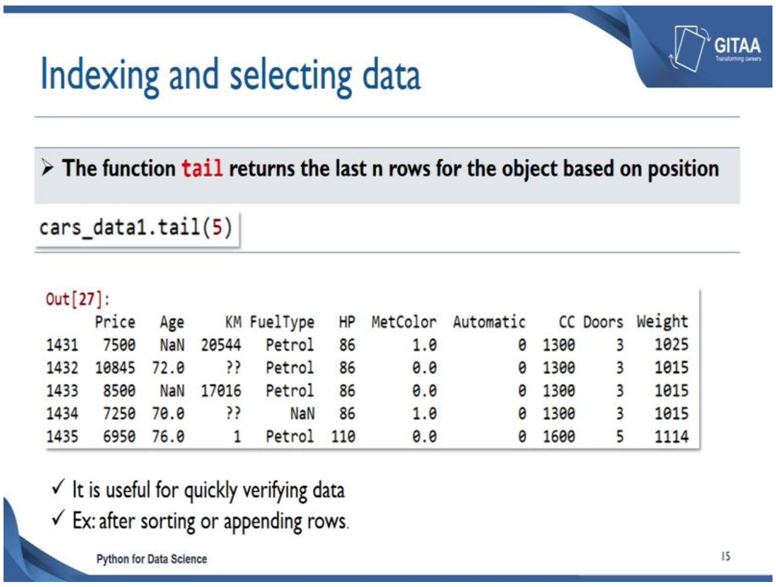
```
Out[17]:
   Price   Age     KM FuelType  HP  MetColor  Automatic    CC  Doors  Weight
0  13500  23.0  46986   Diesel  90       1.0          0  2000  three    1165
1  13750  23.0  72937   Diesel  90       1.0          0  2000      3    1165
2  13950  24.0  41711   Diesel  90       NaN          0  2000      3    1165
3  14950  26.0  48000   Diesel  90       0.0          0  2000      3    1165
4  13750  30.0  38500   Diesel  90       0.0          0  2000      3    1170
5  12950  32.0  61000   Diesel  90       0.0          0  2000      3    1170
```

Python for Data Science 14

So, now we will see how to index and select the data. First thing that we are going to see is about the head function. Basically the head function returns a first n rows from the dataframe. The syntax being dataframe.head inside the function you can just give how many number of rows it should return. And I have used 6 here that means, that the head function will return me 7 rows. If you note by default the head function returns only the first 5 rows from your dataframe.

You can also specify your desired value inside the head function. If you do not give any value to it by default, it will give the first 5 rows. So, if you see, it returns 6 rows with all the column values, so this will be useful whenever you want to get the schema of your dataframe. Just to check what are all the variables that are available in your dataframe, and what each variable value consists of. In that case, the quick method or the quick way to do is using the head function.

(Refer Slide Time: 17:43)



Indexing and selecting data

➢ The function tail returns the last n rows for the object based on position

```
cars_data1.tail(5)
```

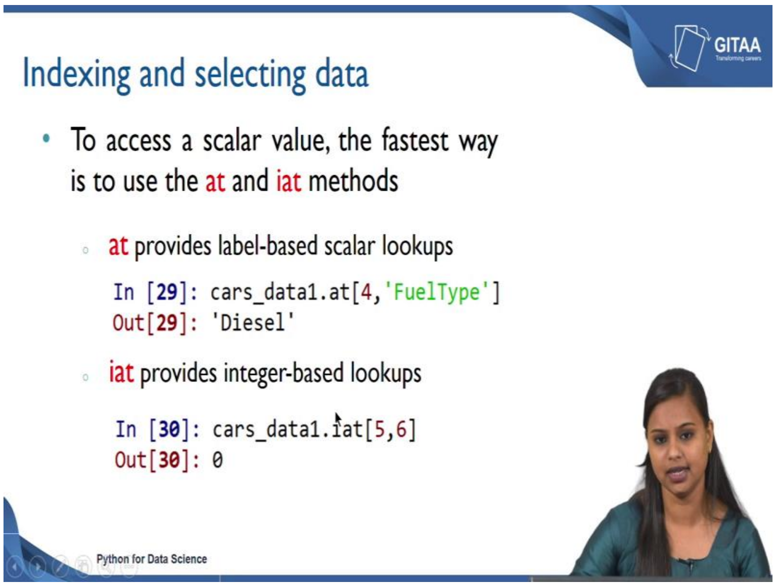
```
Out[27]:
      Price   Age     KM FuelType   HP  MetColor  Automatic    CC  Doors  Weight
1431   7500   NaN  20544   Petrol   86       1.0          0  1300      3    1025
1432  10845  72.0     ??   Petrol   86       0.0          0  1300      3    1015
1433   8500   NaN  17016   Petrol   86       0.0          0  1300      3    1015
1434   7250  70.0     ??      NaN   86       1.0          0  1300      3    1015
1435   6950  76.0      1   Petrol  110       0.0          0  1600      5    1114
```

✓ It is useful for quickly verifying data
✓ Ex: after sorting or appending rows.

Python for Data Science 15

There is also an option where you can get the last few rows from your data frame using the tail function. Basically the function tail returns a last n rows for the dataframe that you have specified. I have used cars_data1.tail, and inside the function I have used 5. Even if you do not give 5, it will return the last 5 rows from your dataframe. This will be a quickest way to verify your data whenever you are doing sorting or appending rows.

(Refer Slide Time: 18:16)



Indexing and selecting data

- To access a scalar value, the fastest way is to use the at and iat methods
  - at provides label-based scalar lookups

```
In [29]: cars_data1.at[4,'FuelType']
Out[29]: 'Diesel'
```

  - iat provides integer-based lookups

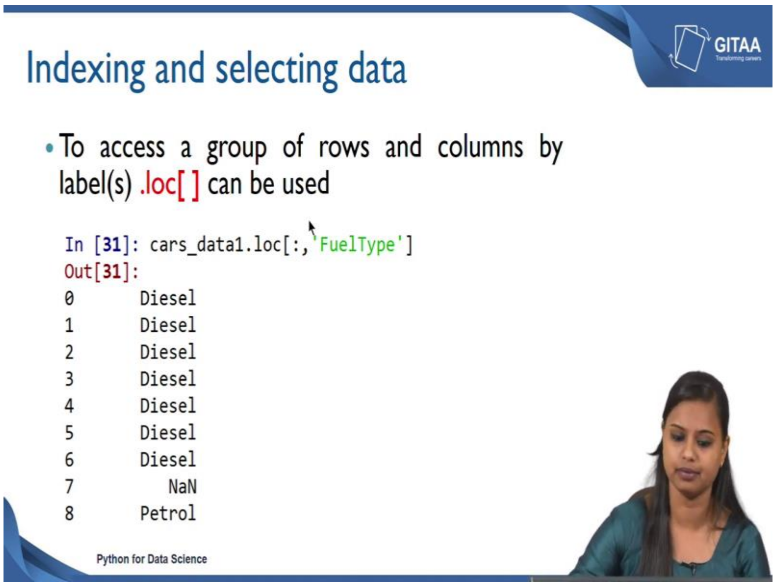
```
In [30]: cars_data1.iat[5,6]
Out[30]: 0
```

So, now we have seen how to access the first few rows and the last few rows from the dataframe. There is also a method to access a scalar value. The fastest way is to use the at and iat methods. Basically the at provides label based scalar look ups. Whenever you want to access a scalar value, you can either use at function or use iat methods. So, if you are using at function, you basically need to give the labels inside the function that is what it means as label-based scalar look ups, I am accessing a function called.at.

And inside the function the first value should be your row label and the second value should be your column label. And I am accessing the scalar value which corresponds to 5th row and corresponds to the fuel type column. So, whatever I have given here is just the labels for rows and columns. And the value corresponds to 5th row and the fuel type is diesel. So, this is how you access a scalar value using the at function using just the row labels and the column labels.

There is also another method called iat. And the iat provides integer based look ups where you have to use the row index and the column index, instead of using labels you can also give row index and the column index. If you are sure about the index, then you can go for iat, but if you are sure about just the column names then in that case you will go for at function. So, if you see here I have used dot iat function where 5 is the 6th row, and 6 is the 7th column. And the value corresponding to 6th row and 7th column would be 0. So, this is how I access the scalar value using the row index and the column index.

(Refer Slide Time: 20:11)



Indexing and selecting data

- To access a group of rows and columns by label(s) .loc[ ] can be used

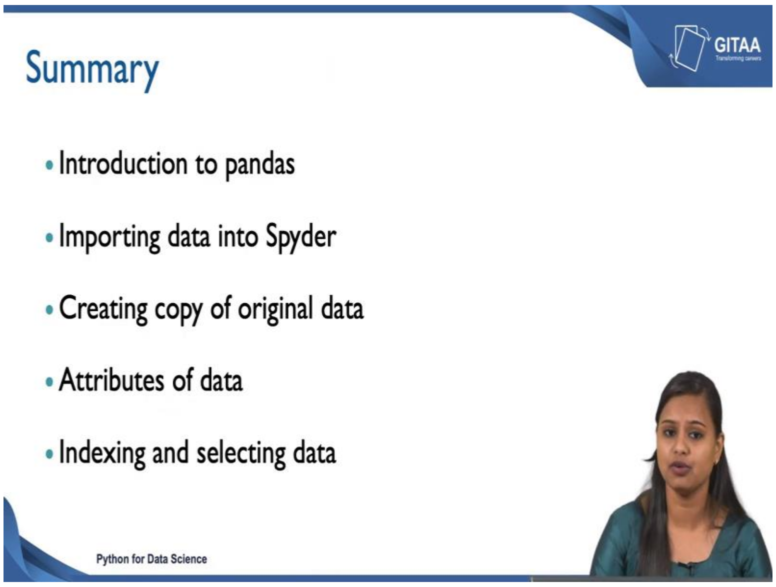
```
In [31]: cars_data1.loc[:,'FuelType']
Out[31]:
0       Diesel
1       Diesel
2       Diesel
3       Diesel
4       Diesel
5       Diesel
6       Diesel
7          NaN
8       Petrol
```

Python for Data Science

So, now we have seen how to access a scalar value from your dataframe, there is also an option where you can access a group of rows and columns by labels that can be done using the.loc operator. So, now, we will see how to use the.loc operator to fetch few rows or columns from your dataframe. So, here also you have to use the dataframe name that is cars_data one. So, whenever you are using the.loc operator, you should be using a slicing operator followed by the function. So, inside the function, you just basically need to give two values. One should represents the row labels, and the other should represent the column labels.

So, here I want to fetch all the row values from a column called FuelType in that case I can use colon to represents all, but here is just the snippet of 9 rows just for explanation purpose, but in your Spyder you will be getting all the rows which are under the FuelType column. You can also give multiple columns. For example, you can give FuelType and price in as a list basically when we say list you can just give the values inside the square brackets, in that case you will get all the row values based on multiple columns. So, this is how we access group of rows and columns using.loc operator.

(Refer Slide Time: 21:35)



Summary

- Introduction to pandas
- Importing data into Spyder
- Creating copy of original data
- Attributes of data
- Indexing and selecting data

GITAA
Python for Data Science

So, in this lecture, we have seen about the pandas library. We have also seen how to import data into Spyder. After importing we have also seen how to get the copy of your original dataframe; followed by that we have seen how to get the attributes of data like row labels and column labels from your data. And after that we have seen how to do indexing in selecting data using iat, at and.loc operators.