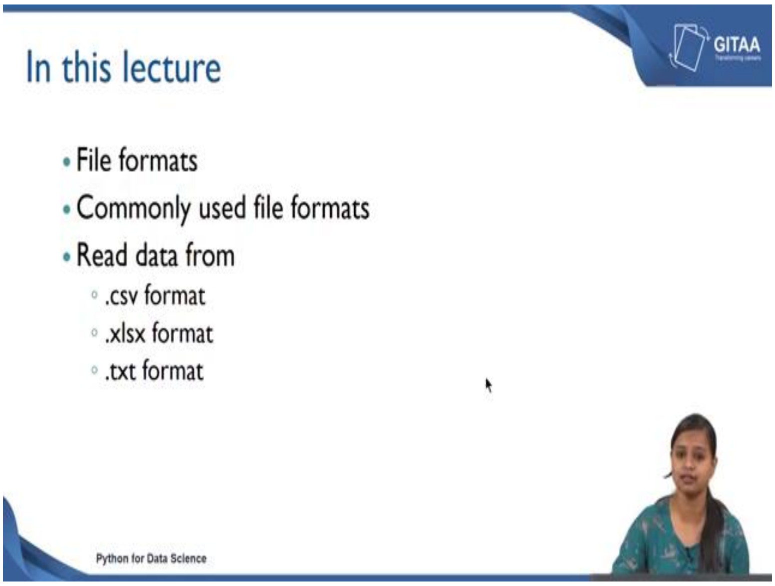
Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 13
Reading Data

(Refer Slide Time: 00:18)

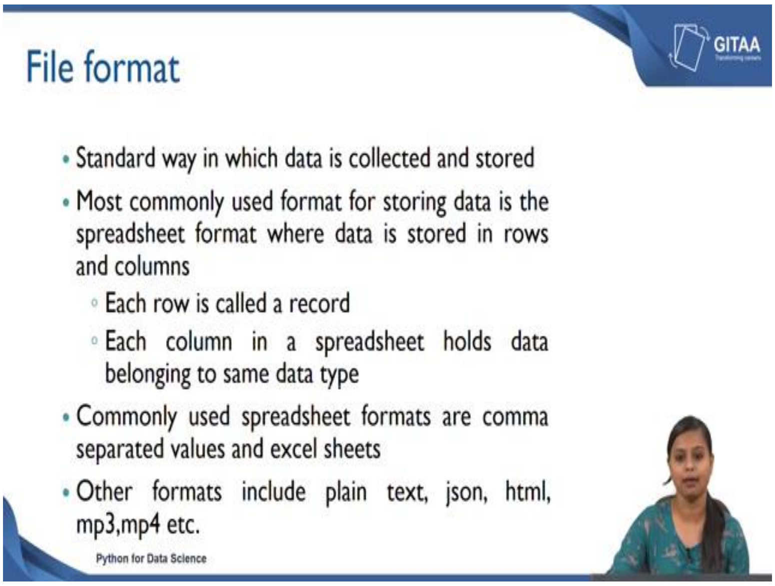


In this lecture

- File formats
- Commonly used file formats
- Read data from
  - .csv format
  - .xlsx format
  - .txt format

Hello, all welcome to the lecture on Reading Data. So, in this lecture we are going to look different file formats and what are the commonly used file formats, how do we read them into Spyder and the commonly used file formats that we will be looking into this lecture is dot csv format, the dot xlsx format and the .text format. We will see in detail about each and every data format on how to read them into Spyder.

(Refer Slide Time: 00:43)



File format

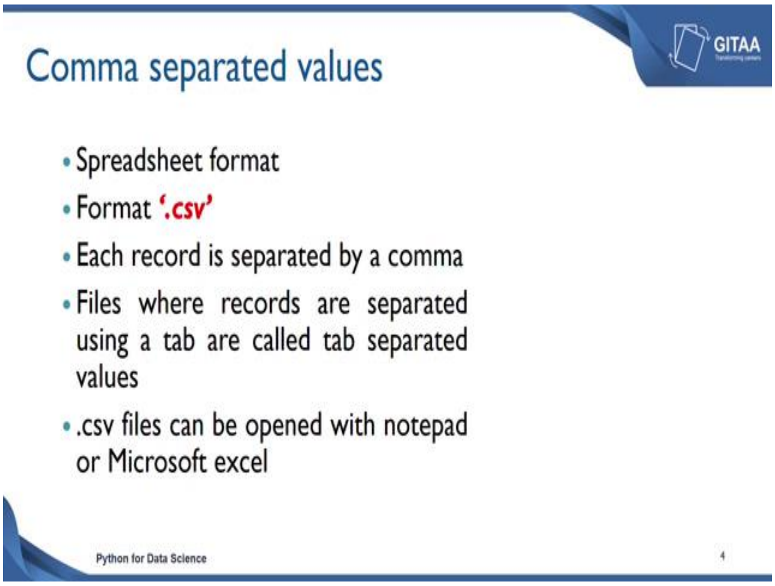
- Standard way in which data is collected and stored
- Most commonly used format for storing data is the spreadsheet format where data is stored in rows and columns
    - Each row is called a record
    - Each column in a spreadsheet holds data belonging to same data type
- Commonly used spreadsheet formats are comma separated values and excel sheets
- Other formats include plain text, json, html, mp3,mp4 etc.

So, now we will see what is the format of file. It is just a standard way in which the data is collected and stored and we know that most commonly used format for storing the data is the spreadsheet format, where the data will be stored in rows and columns and where you see each row will be called as a record or a sample and each column will be represented as a variable.

And each column in a spreadsheet holds data belonging to the same data type and the commonly used spreadsheet formats or comma separated values and excel sheets there are so, many other formats where we can do analytics on that. But since this course is driven towards data science, we will be predominantly looking at how to work with comma separated values and excel sheets.

So, this lecture will be driven towards the file formats like text, comma separated values in the excel sheets. As I mentioned there are other formats like plain text files, json files, html files, mp3 files, mp4 etcetera. So, all these formats can be read into Spyder and you can do any analysis on that based on the requirement but in this course, we will be looking at how to deal with the spreadsheet formatted files like comma separated values in excel sheets.

(Refer Slide Time: 02:03)



Comma separated values

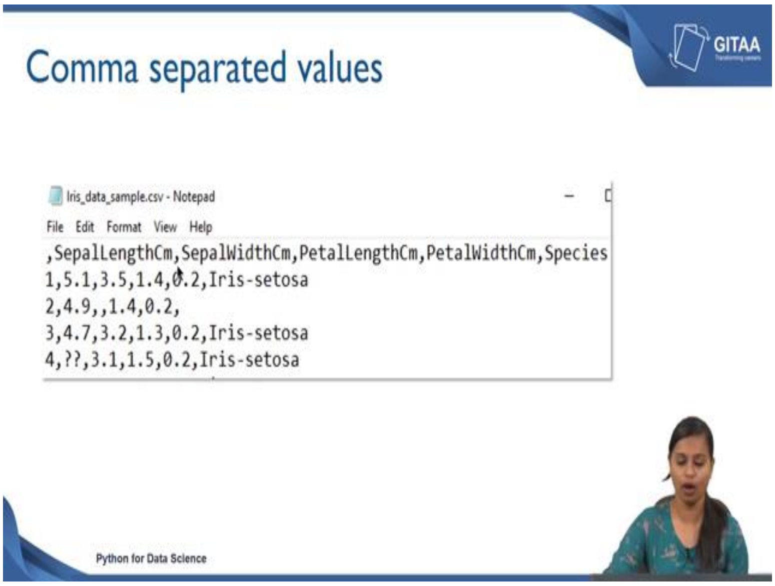
- Spreadsheet format
- Format **'.csv'**
- Each record is separated by a comma
- Files where records are separated using a tab are called tab separated values
- .csv files can be opened with notepad or Microsoft excel

Python for Data Science

4

So, now let us see comma separated values. Comma separated values can be stored in a spreadsheet formats and the format being represented here is csv, where it is just the abbreviation of comma separated values. So, whenever you see a spreadsheet with extension .csv then you call them as comma separated values. So, in this case each record is separated by a comma, but there are other ways where the record can also be separated. So, the files where records are separated using the tab or called tab separated values and the .csv files can also be opened with a notepad or the Microsoft excel.

(Refer Slide Time: 02:45)



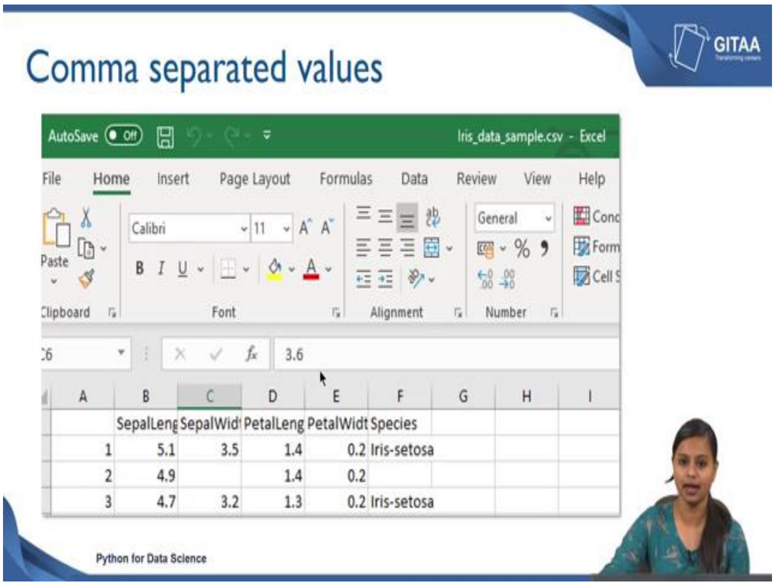
Comma separated values

```
,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species
1,5.1,3.5,1.4,0.2,Iris-setosa
2,4.9,,1.4,0.2,
3,4.7,3.2,1.3,0.2,Iris-setosa
4,??,3.1,1.5,0.2,Iris-setosa
```

Python for Data Science

So, let us see how it looks whenever we open the comma separated values in a notepad. So, this is how it will look like whenever you open the csv files, the .csv files using the notepad. So, you see all the records here, but you will not be able to differentiate between which is the row and which is the column and which cell belongs to which variable. So, that is the problem with viewing the .csv file using the notepad. However, you can also view the .csv file using the notepad as well as excel sheet.

(Refer Slide Time: 03:19)



Comma separated values

Iris_data_sample.csv - Excel

:6 3.6

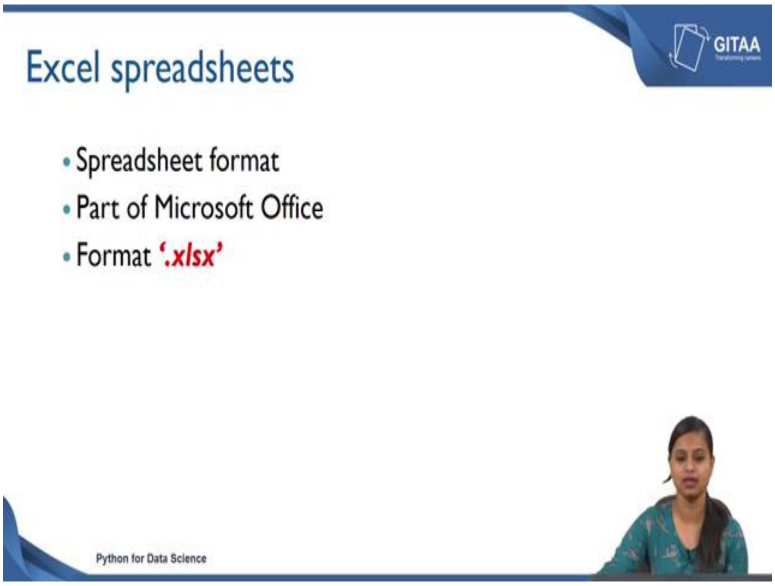
	A	B	C	D	E	F	G	H	I
		SepalLeng	SepalWidt	PetalLeng	PetalWidt	Species			
	1	5.1	3.5	1.4	0.2	Iris-setosa			
	2	4.9		1.4	0.2				
	3	4.7	3.2	1.3	0.2	Iris-setosa			

Python for Data Science

So, now we will see how it looks like whenever you open any .csv files in a excel sheet. The same data is being stored in a different format like every data has been stored in a tabular fashion, where it has been represented in terms of rows and columns and each row is representing samples and each columns are representing the variables. Here we have few variables like sepal length, sepal width, petal length, petal width and the other being the species and you have three records being shown up here.

So, now, we have seen about the comma separated values and what is the extension to it and we have seen how to open those comma separated values using notepad and excel sheet.

(Refer Slide Time: 04:08)



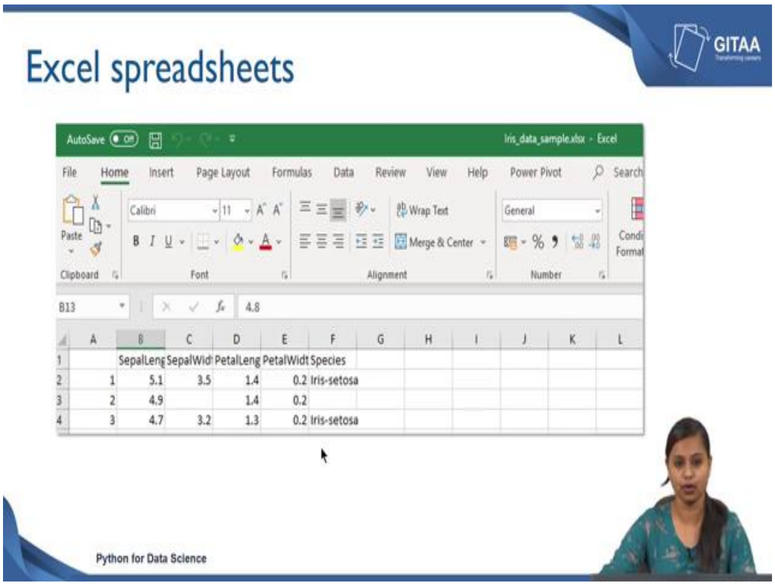
Excel spreadsheets

- Spreadsheet format
- Part of Microsoft Office
- Format *'.xlsx'*

Next we will move on to excel spreadsheets. It is also a spreadsheet format it is a part of Microsoft office. So, if you have a Microsoft Office in your machine then you will be able to work with excel and the format would be .xlsx. So, whenever you save any spreadsheet with the extension .xlsx, then that becomes the excel spreadsheets. So, this is the snippet on how it will look like whenever you save any records using the .xlsx extension.

So, this also gives you the same flavor on how you open your csv file using excel. All the variables have been represented in terms of columns and all the samples have being represented in terms of rows.

(Refer Slide Time: 04:31)



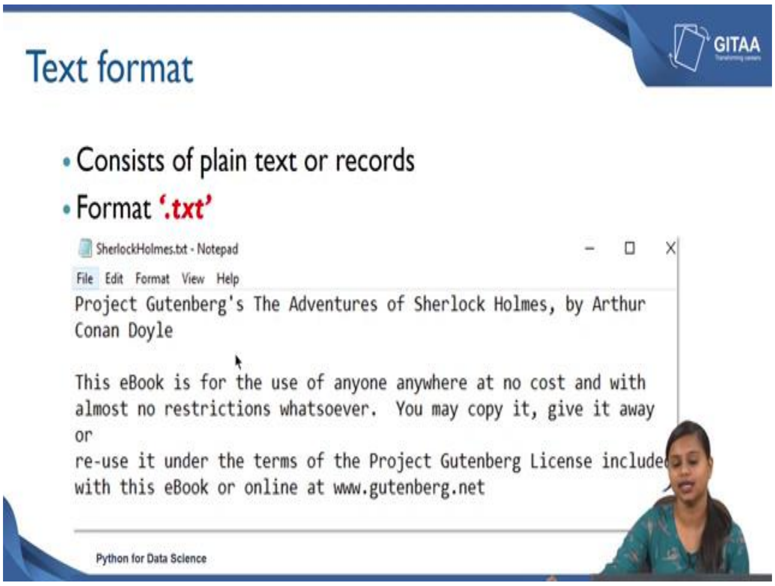
Excel spreadsheets

4.8

	A	B	C	D	E	F
1		SepalLeng	SepalWid	PetalLeng	PetalWidt	Species
2	1	5.1	3.5	1.4	0.2	Iris-setosa
3	2	4.9		1.4	0.2	
4	3	4.7	3.2	1.3	0.2	Iris-setosa

Python for Data Science

(Refer Slide Time: 04:57)



Text format

- Consists of plain text or records
- Format '.txt'

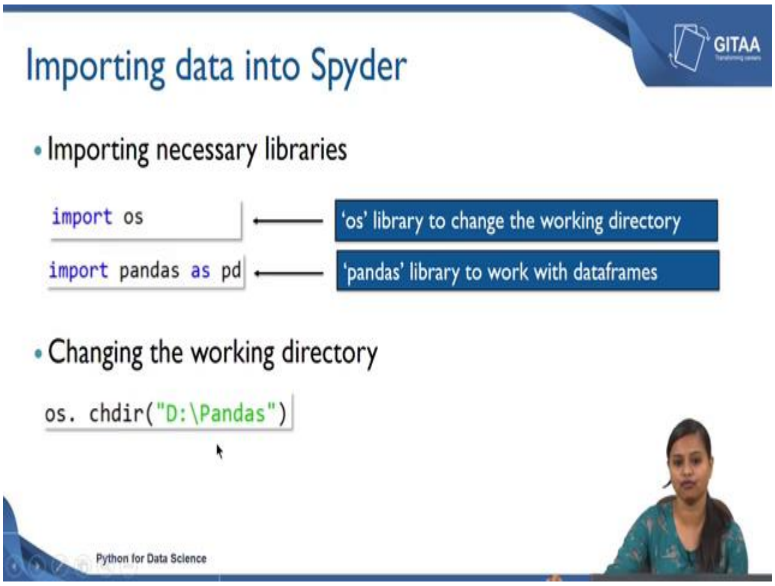
```
Project Gutenberg's The Adventures of Sherlock Holmes, by Arthur
Conan Doyle

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away
or
re-use it under the terms of the Project Gutenberg License include
with this eBook or online at www.gutenberg.net
```

Python for Data Science

Next we will move on to text format. Till now we have been covering about the spreadsheet formatted files where we have seen the two types; one is comma separated values and the other one is being excel sheets. Now, we are going to move to text formats. Text formats basically consists of plain text or records. So, whatever data which consists of plain text or records, we call them as text formatted data and the format would be with .txt extension.

So, whenever you save any file with the extension .txt, then that becomes the text formatted data. So, this is the snippet to show you how a text formatted data will look like. So, in the previous example you been shown up for the data where you have rows and columns. Text file also can contain only plain text but in this lecture, we will be completely focusing on how to read data from different formats like csv and excel.

(Refer Slide Time: 05:55)



Importing data into Spyder

- Importing necessary libraries

```
import os
import pandas as pd
```

'os' library to change the working directory

'pandas' library to work with dataframes

- Changing the working directory

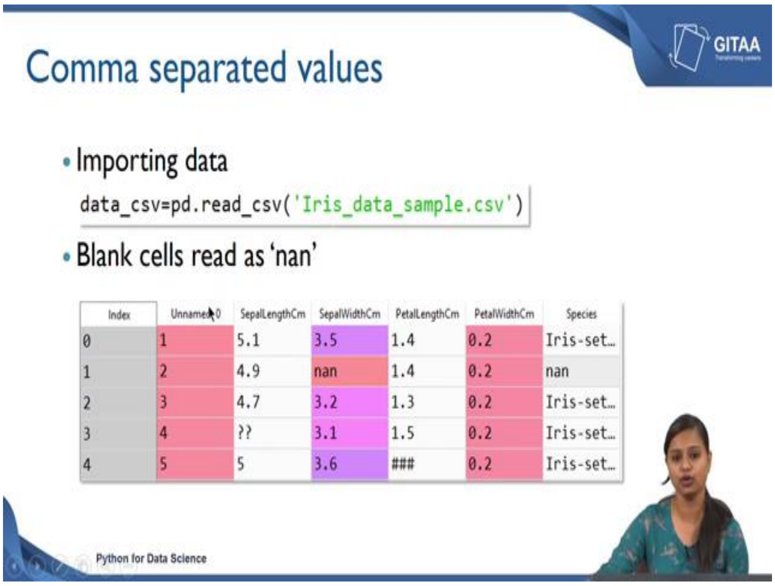
```
os. chdir("D:\Pandas")
```

So, now we will see how to import the data. So, we are going to look how to import data into Spyder. So, before importing data, you have to load or import the necessary libraries. The first library being os, in order to import any library we use the command import followed by the library name. So, we import os library to change the working directory. There might be cases where you want to work with some data in that case you can basically change your working directory to wherever you have saved your data.

So, that is why we have imported the os library and then we are importing the pandas library. So basically, we input pandas library to work with data frames. Whenever we read any data into Spyder, that becomes a data frame. That is what we call it as a data frame, where the data frames being represented in terms of a tabular fashioned data where each row will be represented as sample and each column will be represent are so variable.

So, that is why we are importing the pandas library as pd. So, here pd is just analyze to find out. So, whenever I want to access any function from the pandas library, I can just use pd with the .operator I can use the other related functions. So, that is why I have used pandas as pd. Next we have imported the os library so, using the os library you are using the .operator I am using the function called chdir which stands for changing directory and inside the function you just need to give the path of your file, wherever it is lying on your drive or in your system.

(Refer Slide Time: 07:37)



Comma separated values

- Importing data

```
data_csv=pd.read_csv('Iris_data_sample.csv')
```

- Blank cells read as 'nan'

Index	Unnamed: 0	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-set...
1	2	4.9	nan	1.4	0.2	nan
2	3	4.7	3.2	1.3	0.2	Iris-set...
3	4	??	3.1	1.5	0.2	Iris-set...
4	5	5	3.6	###	0.2	Iris-set...

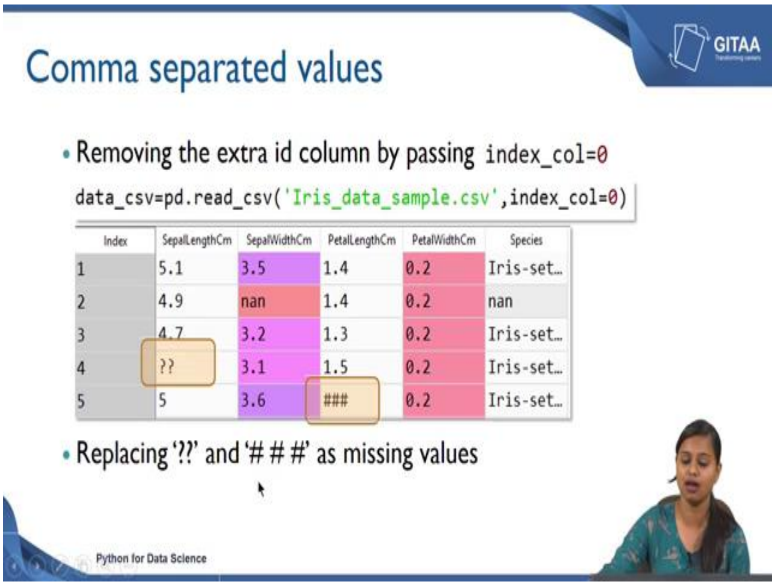
So, now let us see how to read a csv file into Spyder. So, read_csv is the command that is used to read any csv files into Spyder and that comes from the package called pandas. I have used pd.read_csv. Inside the function you just need to give the file name and the file name being Iris_data_sample and I have given the file name with the extension .csv and I have also given the file name within single quote.

So, these two things are very mandatory to read any file into Spyder. One is the extension and one is single or double quotes and the other is enclosing the file name inside the single or double quotes and if you see whenever I am reading, I am saving into an object called data_csv. Now data_csv becomes data frame whenever you execute this line and whenever you read any data into Spyder, all the blank cells will be read as nan because there might be cases where you have some missing values in your data as a blank value. So, in that case the Python will default all the blank values to nan. so that whenever you are going to use any function which is going to describe how many missing values are there in your record, then the nan will account for that.

So, this is just a screenshot of the data whenever you open it from your variable explorer tag. So, you have so many variables here starting from index to species and index being represented as the row labels starting from 0 to 4 here from the snippet and you have another column called unnamed colon 0 where the values are starting from 1 to 5.

If you feel this represents an id to each of the rows, then you can keep this column unwanted 0 because you do not want too many columns which have the same information to it I want to keep the unnamed 0 column and I want to get rid of the index column. So, that I have some ids to each of the observation that can be useful for the future analysis.

(Refer Slide Time: 09:41)



Comma separated values

- Removing the extra id column by passing `index_col=0`

```
data_csv=pd.read_csv('Iris_data_sample.csv',index_col=0)
```

Index	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-set...
2	4.9	nan	1.4	0.2	nan
3	4.7	3.2	1.3	0.2	Iris-set...
4	??	3.1	1.5	0.2	Iris-set...
5	5	3.6	###	0.2	Iris-set...

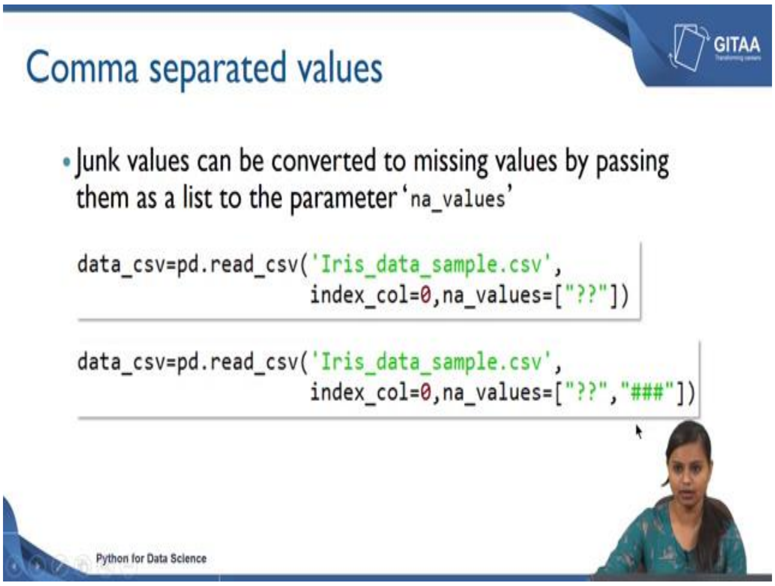
- Replacing '??' and '# # #' as missing values

So, let us see how to remove the unwanted column. So, if you want to remove the extra id column, you can pass an argument called index_col is equal to 0 whenever you read a data frame. So, let us see how to do that. So, I have used the same command called pd.read_csv inside the function. I have given the file name followed by that I have given another argument called index_column is equal to 0. That means, that I am making my first column as index column. In that case my index column would go and whatever I had it in the name of unnamed 0 that became the index column here.

So now, you have 6 variables where the first variable represents the serial number or id for each and every records in your data frame iris data sample. So, this is how we read a csv file and this is how we also set your index column by specifying the column number here. So, if you see here there are special characters other than the numerical values like question marks and hash.

These can be the representation of missing values, but I have mentioned that all the blank values will only be replaced with nan because in Python by default all the blank values will only be replaced with nans not, other special characters but if you are sure that these special characters just represents the missing values in your data and if you want to consider them as nan you can also do that by replacing all the question marks and hash as missing values.

(Refer Slide Time: 11:19)



Comma separated values

- Junk values can be converted to missing values by passing them as a list to the parameter 'na_values'

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

```
data_csv=pd.read_csv('Iris_data_sample.csv',
                     index_col=0,na_values=["??","###"])
```

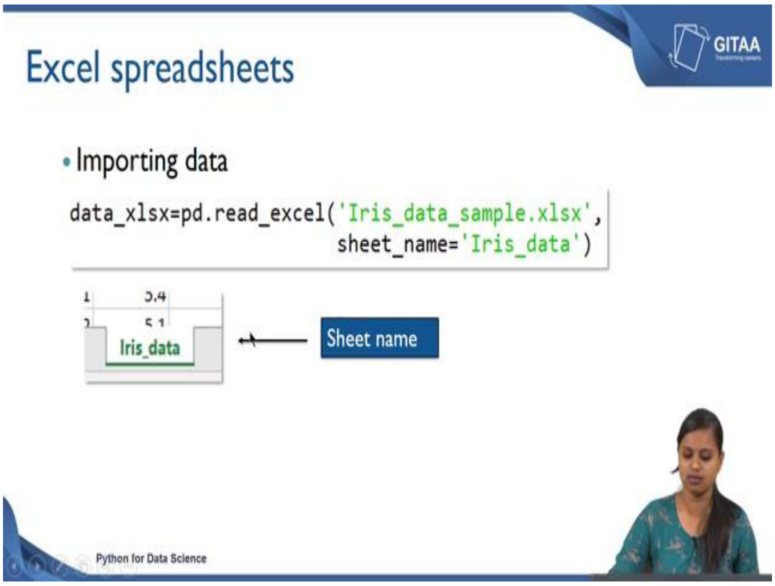
So, let us see how to do that. So, if you see all the junk values can be converted to missing values by passing them as a list to the parameters na_values. So, this can also be done whenever you read a data frame into Spyder. So, we are following the same command followed by the index call argument. You have to just give na_values equal to a character which should be enclosed inside the square bracket that represents it is a list.

So, I have first given question mark here that should be given in inside the single or double quotes, then it takes it as a character and it will check those characters in the data frame and if there are any question marks in the data frame then it defaults that question marks to nan values. So, this is what we want to do now.

So, I have included na_values argument and inside the square bracket I have just given question marks. So, once I read this, I will be able to get rid of all the question marks and that will be converted as nan values. Say, but in the in our case there are so many special characters that are representing the missing values.

So, I can just give multiple values inside the square brackets so that all these values will be read as nan. So, here I have just added the hash by using a comma. So, all the values have been treated as list here and it will check for these two values and replace all these special characters with a nan value.

(Refer Slide Time: 12:51)



Excel spreadsheets

- Importing data

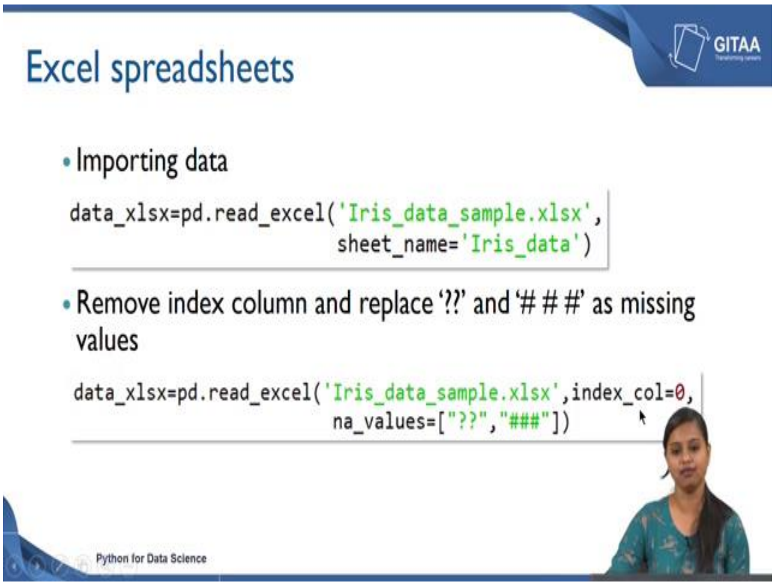
```
data_xlsx=pd.read_excel('Iris_data_sample.xlsx',
                        sheet_name='Iris_data')
```

So, now this is how we replace all those special characters, and which represents your missing values into nan. So, read_excel is the command to read any excel sheet into Spyder and that also comes from the pandas library that is why we have used pd.read_excel inside the function I have just given the file name with the extension .xlsx.

So, here the same file I am using with different formats. So, the extension being .xlsx here and if you are dealing with excel spreadsheets, then there might be cases where you have multiple tabs and a tabs being differentiated with different tab names. In that case if you want to access data from separate tab, then you give the sheet name under the sheet_name argument.

So, here iris_data is the sheet name from which I want to access the data frame. So, I have given iris_data under saved into an object called data_xlsx. Now, this becomes a data frame. So, there is a snippet from which you can get your sheet name. So, in your excel sheet you will have sheet names. So, you can just give the sheet name under the sheet_name argument.

(Refer Slide Time: 14:03)



Excel spreadsheets

- Importing data

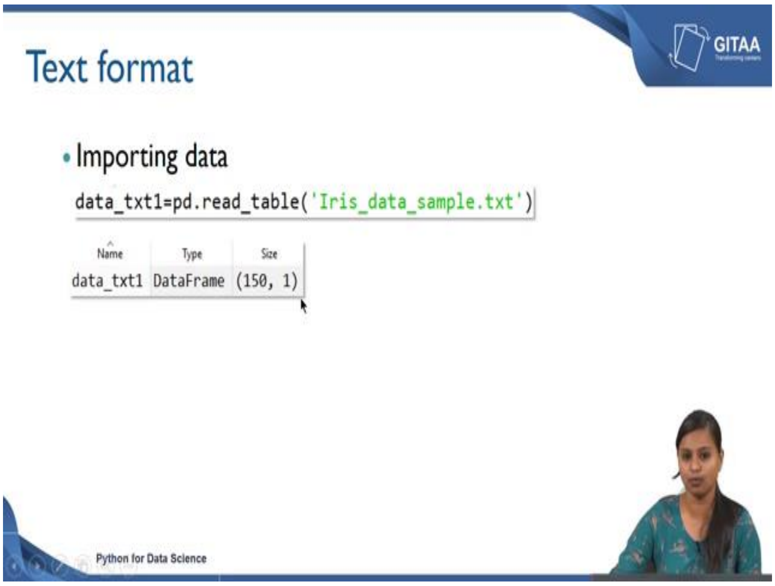
```
data_xlsx=pd.read_excel('Iris_data_sample.xlsx',
                        sheet_name='Iris_data')
```

- Remove index column and replace '??' and '# # #' as missing values

```
data_xlsx=pd.read_excel('Iris_data_sample.xlsx',index_col=0,
                        na_values=["??","###"])
```

And here also since we are using the same data with different formats here also you have the problem with the question marks and hash which represents the missing values. So, you can follow the same steps here to get rid of those special characters and to convert them as nan. So, let us do that also I have just included another argument called na_values where I have set the values to be converted as nan and I have also set my first variable as the index column. So, now, we have seen how to read excel file into Spyder.

(Refer Slide Time: 14:37)



Text format

- Importing data

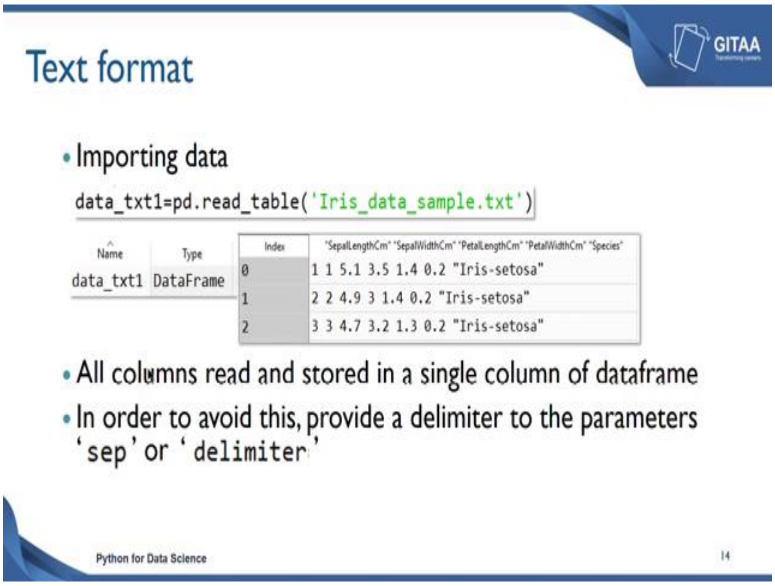
```
data_txt1=pd.read_table('Iris_data_sample.txt')
```

Name	Type	Size
data_txt1	DataFrame	(150, 1)

Next we are going to see how to import any text formatted data into Spyder. So, read_table is the command that is used to read any text format data that also comes from the pandas library and inside the function you just need to give the file name. Here the file name being iris_data_sample. The same file that we are working with and extension being .txt.

I have also saved my data frame as data_txt1 and once you read it this will be shown up in your variable explorer where you can see the name of your object is data_txt1 and the type of your object is data frame and the size being 150 rows and 1 column. So, 150 comma 1 represents rows and columns. So, there are 150 rows and only 1 column but we know that we had 5 variables in our data frame.

(Refer Slide Time: 15:40)



Text format

- Importing data

```
data_txt1=pd.read_table('Iris_data_sample.txt')
```

Name	Type
data_txt1	DataFrame

Index	"SepalLengthCm" "SepalWidthCm" "PetalLengthCm" "PetalWidthCm" "Species"
0	1 1 5.1 3.5 1.4 0.2 "Iris-setosa"
1	2 2 4.9 3 1.4 0.2 "Iris-setosa"
2	3 3 4.7 3.2 1.3 0.2 "Iris-setosa"

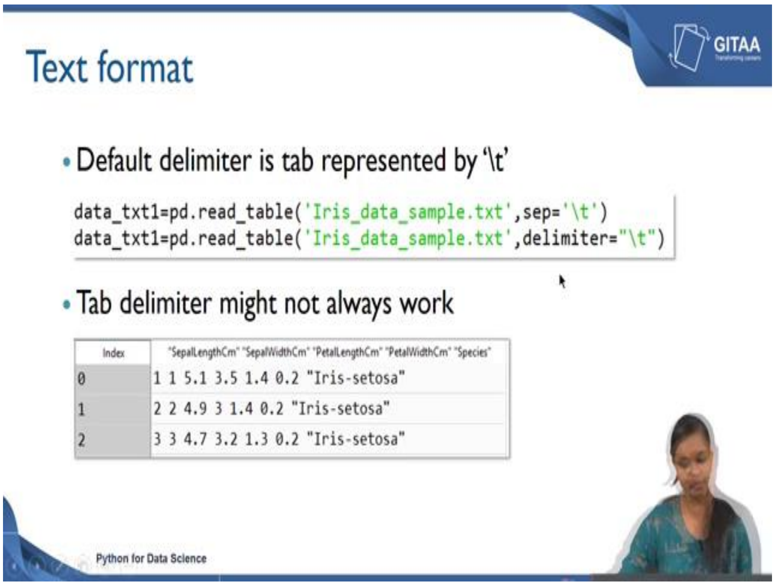
- All columns read and stored in a single column of dataframe
- In order to avoid this, provide a delimiter to the parameters 'sep' or 'delimiter'

Python for Data Science

14

So, in this case it has been showed up with only 1 column. Let us see what is the problem, because all the column have been read as a single column. That is why the dimension is being 150 cross 1. So, now, let us see how to encounter this problem. If you see all the columns are read and stored in a single column of a data frame. So, in order to avoid this problem provide a delimiter to the parameters sep or delimiter let us see how to do that.

(Refer Slide Time: 16:06)



Text format

- Default delimiter is tab represented by '\t'

```
data_txt1=pd.read_table('Iris_data_sample.txt',sep='\t')
data_txt1=pd.read_table('Iris_data_sample.txt',delimiter="\t")
```

- Tab delimiter might not always work

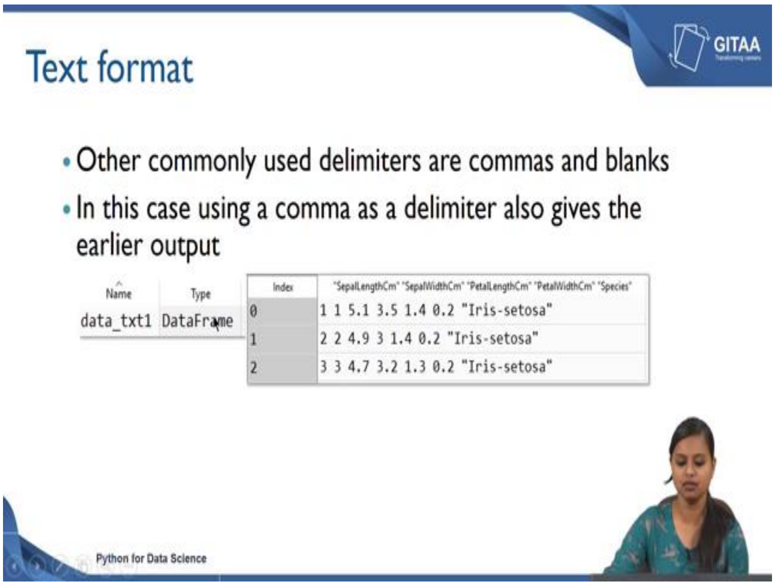
Index	"SepalLengthCm" "SepalWidthCm" "PetalLengthCm" "PetalWidthCm" "Species"
0	1 1 5.1 3.5 1.4 0.2 "Iris-setosa"
1	2 2 4.9 3 1.4 0.2 "Iris-setosa"
2	3 3 4.7 3.2 1.3 0.2 "Iris-setosa"

Python for Data Science

So, the default delimiter is tab represented by slash t and let us see how to use those parameters, sep is 1 of the parameters that is used to provide a delimiter. Here I have used the delimiter called tap even if you do not use tab here by default it will always take the tab delimiter data, but I have specified the tab delimiter here that can be given using two separate arguments; one is sep separator and another one is delimiter. So, you can use any one of it and the tab delimiter might not always work.

Let us see what is the problem, the data is not a tab delimiter data that is why it is not working. The other commonly used diameters or commas and blanks other than tabs. So, in this case, using a comma as a delimiter also gives the earlier output right. In this case I have tried with a comma and that is also giving the same output where all the columns have been represented as a single column.

(Refer Slide Time: 16:51)



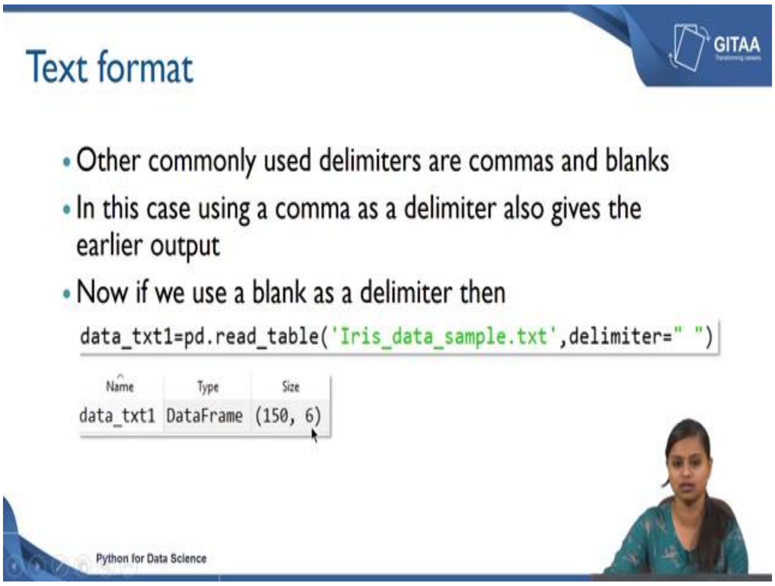
Text format

- Other commonly used delimiters are commas and blanks
- In this case using a comma as a delimiter also gives the earlier output

Name	Type
data_txt1	DataFrame

Index	"SepalLengthCm" "SepalWidthCm" "PetalLengthCm" "PetalWidthCm" "Species"
0	1 1 5.1 3.5 1.4 0.2 "Iris-setosa"
1	2 2 4.9 3 1.4 0.2 "Iris-setosa"
2	3 3 4.7 3.2 1.3 0.2 "Iris-setosa"

(Refer Slide Time: 17:10)



Text format

- Other commonly used delimiters are commas and blanks
- In this case using a comma as a delimiter also gives the earlier output
- Now if we use a blank as a delimiter then

```
data_txt1=pd.read_table('Iris_data_sample.txt',delimiter=" ")
```

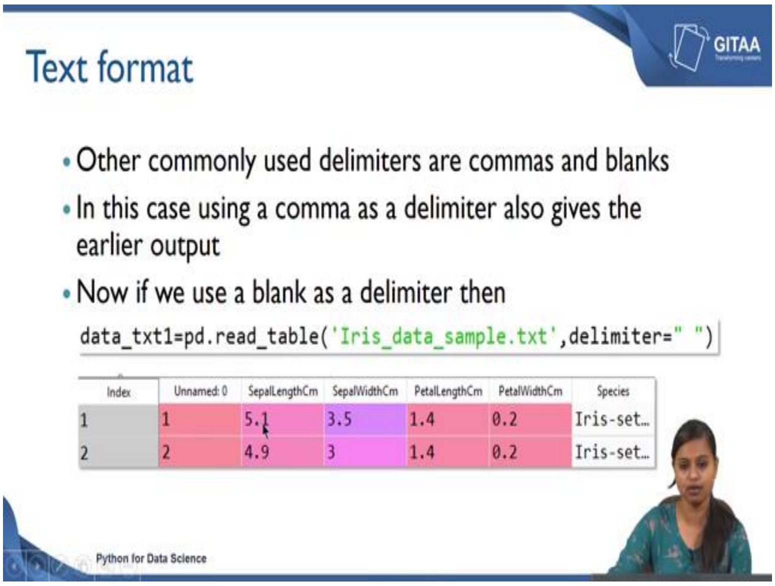
Name	Type	Size
data_txt1	DataFrame	(150, 6)

Python for Data Science

So, now if we use a blank as a delimiter then let us see what happens. So, I have used blank space inside the double quote that represents the delimiter as blank. So, if I try to read this, I will be able to see the data has been read with 150 rows and 6 columns. So, now, not the tab delimiter worked here and also the command did not worked here and I have since I have used the blank delimiter the data has been read with 150 rows and 6 columns.

So, you have to be really sure about your data and how many rows and how many columns it is expected to have and you will be able to cross verify that whether you got the correct data or not whenever you read it.

(Refer Slide Time: 17:56)



Text format

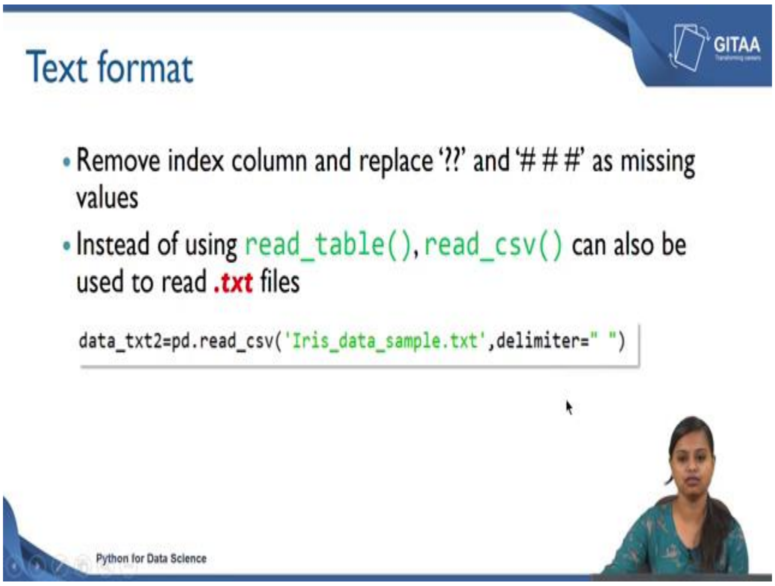
- Other commonly used delimiters are commas and blanks
- In this case using a comma as a delimiter also gives the earlier output
- Now if we use a blank as a delimiter then

```
data_txt1=pd.read_table('Iris_data_sample.txt',delimiter=" ")
```

Index	Unnamed: 0	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	1	5.1	3.5	1.4	0.2	Iris-set…
2	2	4.9	3	1.4	0.2	Iris-set…

Python for Data Science

(Refer Slide Time: 18:06)



Text format

- Remove index column and replace '??' and '# # #' as missing values
- Instead of using read_table(), read_csv() can also be used to read **.txt** files

```
data_txt2=pd.read_csv('Iris_data_sample.txt',delimiter=" ")
```

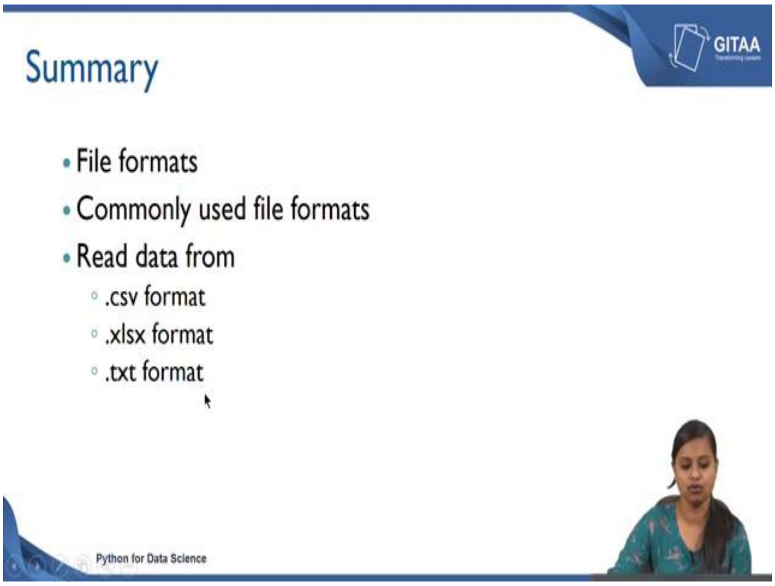
Python for Data Science

So, now if you see here all the records have been separated by cells by using the blank delimiter. So, in this case also you have to remove the index column and replace all the question marks, hash as missing values. I have not shown here but you can try that using the same format which you have used for csv and excel formats. Instead of using the read_table function, you can also use read_csv function which we used to read the csv file.

Now I am trying to show you how to read a text file using the .read_csv command.

So, you can use the same function that we used to do it whenever we want to import any csv file into Spyder but you have to just give the delimiter whenever you are reading any text file. That delimiter should corresponds to the type of data that you have in your sheet. So, here our data is delimeted by a blank and I have given the delimiter is equal to blank. So, now we have come to the end of the lecture we have where we have seen about importing different formats of data into Spyder.

(Refer Slide Time: 19:08)



Summary

- File formats
- Commonly used file formats
- Read data from
  - .csv format
  - .xlsx format
  - .txt format

So, let us summarize whatever we have seen in this lecture. So, we have seen different formats of file, what are the formats of file that we will be looking into in this course and what are the commonly used file formats. So, that we can read those formatted files into Spyder. We have seen three formats of data; one being .csv format, we have seen how a comma separated values data will look like and we have also seen how to read them into Spyder.

Followed by that we have seen how to read .xlsx format. We have also seen how an excel sheet and how the data inside the excel sheet will look like. We have also seen how to read the text formatted data using both read_table command and read_csv command. So, on the whole we have seen how to read different formats of data into Spyder.

Thank you.