

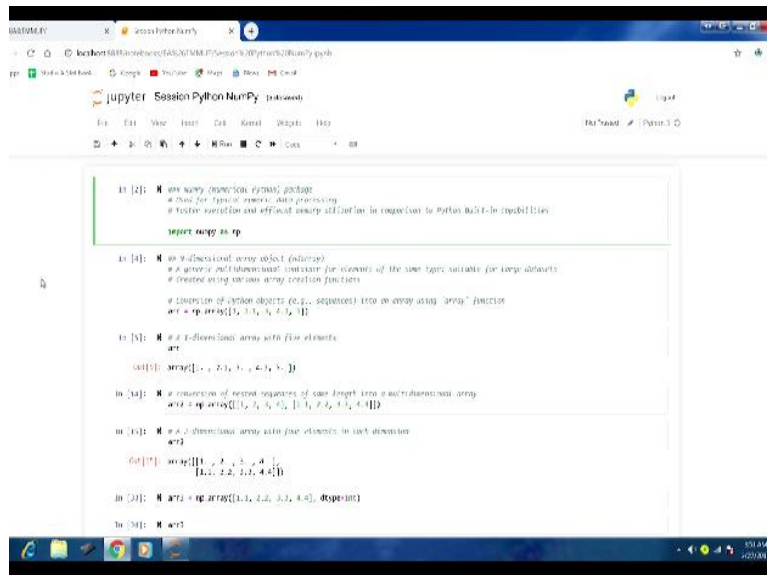
**Business Analytics And Text Mining Modeling Using python**  
**Prof. Gaurav Dixit**  
**Department of Management Studies**  
**Indian Institute of Technology Roorkee**

**Lecture-19**  
**Text Collection And Transformation-Part I**

Welcome to the course business analytics and text mining modeling using python. So, in previous few lectures we have been able to complete the built in capabilities aspects as also. So now we will move into the another important package that is NumPy that is numerical python package. This package is especially useful for generic numerical data processing. And though in analytics context we are you know, typically required to work with the tablet kind of data for that will also in the coming lectures will discuss will learn about the another package that is panda.

However, this is also very much important because we will be dealing with the larger data sets and while we are dealing with the largest data the certain processing, certain data processing, certain data transformation, certain operations that we will have to perform for those purposes this particular package can be really useful in the analytics context. So, let us start our discussion on this particular package NumPy.

**(Refer Slide Time: 01:35)**



```
In [2]: # one numpy (numerical python) package
# this is for typical numeric data processing
# faster operations and efficient memory utilization in comparison to python built-in capabilities
import numpy as np

In [4]: # an n-dimensional array object (ndarray)
# a generic and efficient container for vectors of the same type, suitable for large datasets
# created using various array creation functions
# conversion of python objects (e.g., sequences) into an array using 'array()' function
arr = np.arange(10, 1, -1, dtype='f')

In [5]: # a 1-dimensional array with five elements
arr
Out[5]: array([ 9.,  8.,  7.,  6.,  5.])

In [14]: # a conversion of nested sequences of same length into a multidimensional array
arr = np.arange(10, 1, -1, dtype='f').reshape(2, 5)

In [15]: # a 2-dimensional array with five elements in each dimension
arr
Out[15]: array([[ 9.,  8.,  7.,  6.,  5.],
               [ 4.,  3.,  2.,  1.,  0.]])

In [21]: # arr2 = np.arange(10, 1, -1, dtype='f').reshape(2, 5)
```

So, this is as you can see in the comments, this is typically used for you know, typically medical numeric data processing. And the big advantage of this particular package over the pattern building capability is in terms of you know execution. So, faster, the code can be executed much faster and also the memory utilization is you know, a more efficient memory utilization will happen you know, using the algorithms based on this package in the functions.

And code you know, based on this particular package, so, it has certain advantages or python built in, you know, capabilities built in you know, functionalities that we have. So, therefore, suitable for you know working with larger data sets, so, if any processing that is ready to be done on a larger data set, so probably functions from this package would be better than writing code using the built in you know, python capabilities functionalities.

So, as usual, first thing would be to import to load this package into the python environment. So, this is encode NumPy we typically we use np.

**(Video Starts: 02:53)**

So, let us run this and then 1 of the important aspects of this package is this N dimensional array object ndarray. So, this particular object is really useful for data processing, because it can you know, very efficiently store large data sets and very efficiently can be used to perform operations you know on those data points.

So, let us talk about N dimensional array object. So, as you can see here, again, a generic multi dimensional container for elements of the same type, this is an array so just like you know list object that we talked about in the python built in, you know, capability session. So, there, this arrays 1 dimensional array, specifically 1 dimensional arrays going to be quite similar to list objects that we have discussed before

(Refer Slide Time: 05:30)

The screenshot shows a Jupyter Notebook session titled "Session Python pandas". The notebook contains two code cells. The first cell (In [165]) contains the following code:

```
# filling values in the operation involving unmatched indices
# using add method

# fill NaNs in dfs with 0s
df4.add(df5, fill_value=0) # doesn't work since df4 still has NaN values
```

The output (Out[165]) shows a DataFrame with columns 'Delhi', 'Mumbai', and 'Bangalore' and rows 'a', 'b', 'c', and 'd':

|   | Delhi | Mumbai | Bangalore |
|---|-------|--------|-----------|
| a | 0.0   | 1.0    | 2.0       |
| b | NaN   | NaN    | NaN       |
| c | 3.0   | 4.0    | 10.0      |
| d | 12.0  | 14.0   | 16.0      |

The second cell (In [166]) contains the following code:

```
df4[1:2]=2
df4
```

The output (Out[166]) shows the same DataFrame as above, but with the values in rows 'b' updated to 2.0 for all three columns.

|   | Delhi | Mumbai | Bangalore |
|---|-------|--------|-----------|
| a | 0.0   | 1.0    | 2.0       |
| b | 2.0   | 2.0    | 2.0       |
| c | 3.0   | 4.0    | 10.0      |
| d | 12.0  | 14.0   | 16.0      |

A small video inset in the bottom right corner shows a person speaking.

So, arrays are a multi-dimensional container. So, you know, we will see through examples how this can be worked with, so suitable for large data sets, and it acts as a container. Now, there are various array creation functions that can be used to create these objects array objects. So let us start with the first one, the most common one is the array function. So conversion of Python objects, mainly sequences, and even their list object into an array can be done using array function.

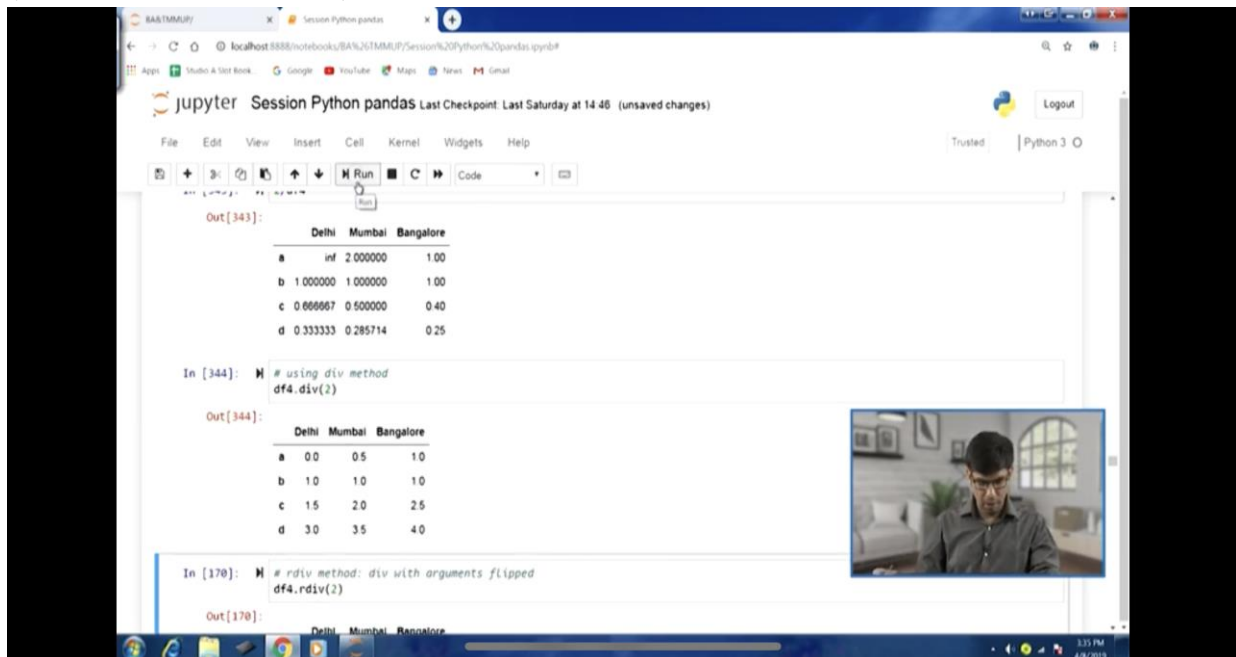
So here you can see I have given one example. So here, `np.array`, this is the function that we are using and within the parenthesis the only argument that we have here is the data that we are passing, which is actually a list object. So in this list, we have these 5 elements, and this is being passed as the argument here in the `np.array` function, and this function will convert it into a NumPy array object. So if I run this I will get an array object.

So, this is a 1-dimensional array with 5 elements. So, if you know try to print this array object here, you can see in the output from a 3D array 1 2.2, 1. and 2.1 3.4.3 5. So, you can see most of the elements in this are being installed using the floating point data type. So, you can see 1-dimensional array with 5 elements. So, very similar to you know, the list that we have learned in previous sessions. Now, let us take another example.

So, in this case, we are taking we can convert nested sequences of same length into a multi dimensional array. So, the sequence with same length sequence we can take and then we can create this multi dimensional array. So, here again you can see we are using the same function np.array and here this is a list of lists. So, comprising of this you know list is comprising of 2 list, and you can see 4 elements in each of these lists 1 2 3 4 four elements in the first list, then 1.1 2.2 3.3 and 4.4 you know, 4 elements in the second list.

So, this particular data, this particular list can actually be used to create an 2 dimensional array, so let us run this. And you can see if we run this axis array again and we can see that in 2 dimensional array with 4 elements in each dimension has been created. And you can see array and you can see then you know parenthesis 1.2. 3.4., these are the elements of the first dimension then 1.1 2.2 3.3 and 4.4.

**(Refer Slide Time: 11:54)**



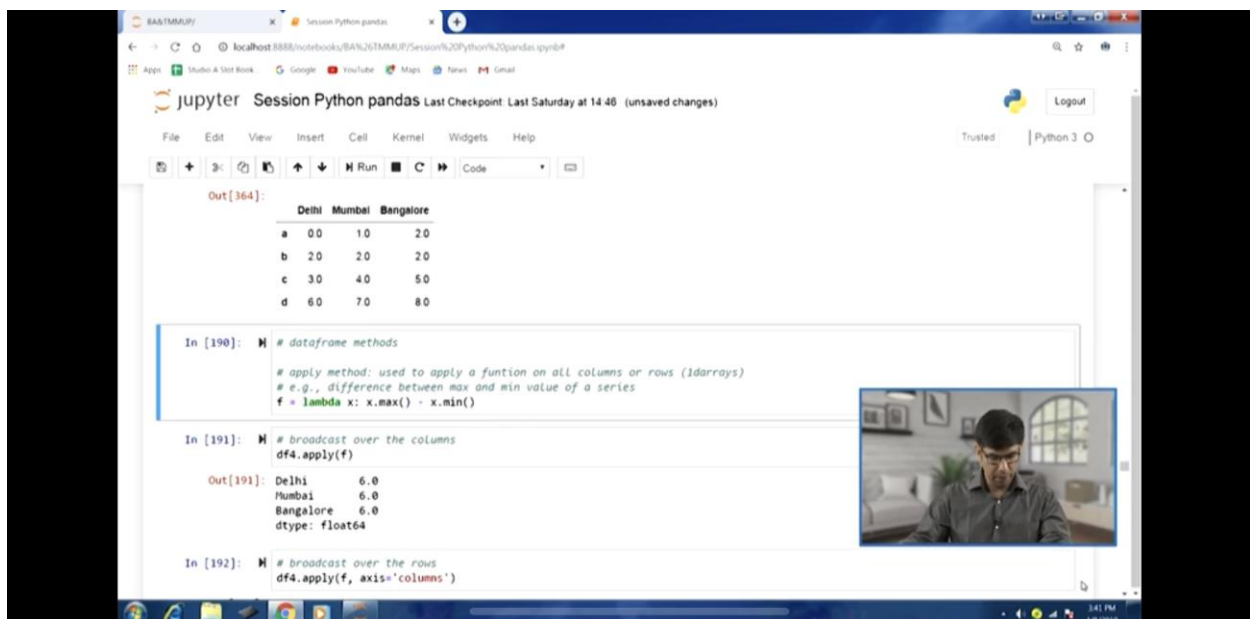
These are the elements of the second dimension. So in this fashion, we can create a multi dimensional in this case 2 dimensional array. Now, let us take another example. So, here you can see we are again using the same function np.array and you know, the argument the first argument is on data input data that we want to convert into an array. So 4 elements list with 4 elements.

However, in the second argument we are also specifying in this example, the data types. So, here we are specifying the int integer data type here. So, the array even though the input data and all the elements of this particular data list data, they are floating point numbers, but since we are trying to create an array comprising of you know interior, you know elements by specifying this d type, you know, keyword argument here.

So, appropriately then this array is going to be created. So, let us learn this. Now, because the list elements were floating point number and we are trying to convert into an integer array. So, the decimal part is going to be truncated. So, if we look at the output for array3 here, so you can see output number 7 array 1 2 3 4 and the original data was 1.1 2.2 3.3 and 4.4. So, because of the contest convergent, because of the you know this conversion into interior array from a you know floating kind of list this you know decimal part has been truncated here.

So, this is how we can use array function to create you know to rather convert python build in objects into an array you know NumPy object NumPy array object. So, this is 1 way, there are other array creation functions. So, you know, which can also be used to create array objects. So, we will see them through you know, few examples. Now, before we talked about those functions, let us discuss some of the properties of array objects.

**(Refer Slide Time: 17:24)**



So, first talk about number of dimensions. So, given an array, how do we you know, find out the number of dimensions that are there. So, for this we have this ndim metadata object or you know, ndim attribute that can be used to find out the number of dimensions of an array. So, here array object we have already created, so, array.ndim if I run this, I will get the number of dimension. So, in this case, the array object that we have are located it just had 1 dimension as you can see in the output number 3, you know just you know, 1 dimension with 4 elements.

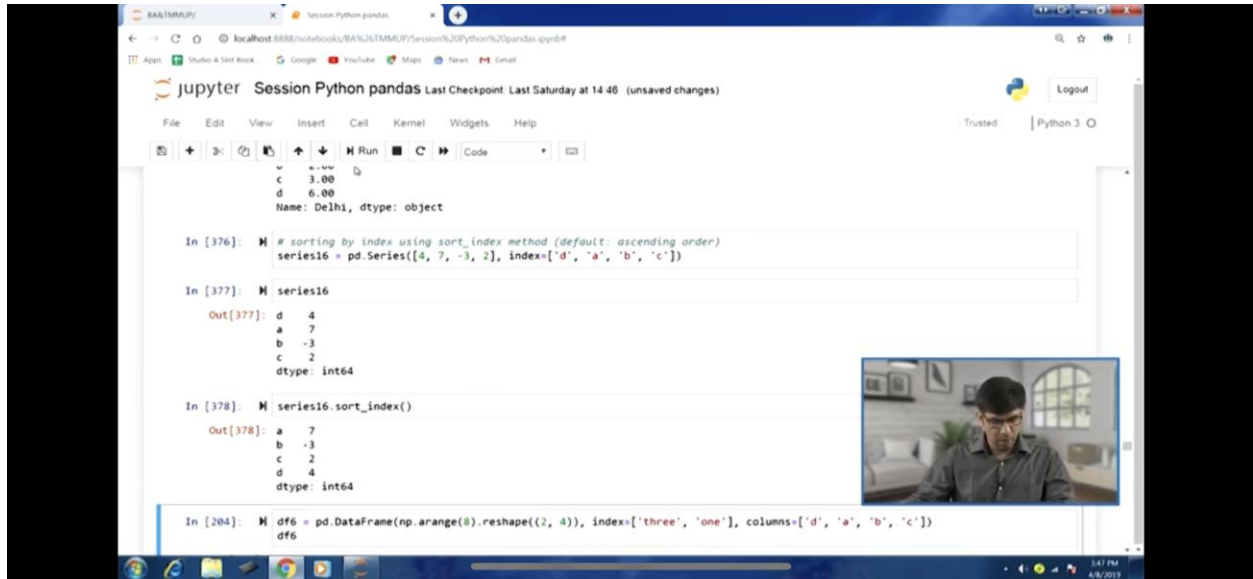
Now, the secondary object that we had created array 2.ndim here we have, if I run this line then number of dimensions comes out to be 2 in the output 9. So, you can see and as you can see in the output 5 there were 2 dimensions. So, number of dimensions we can find out using this metadata object. Similarly, you know, if you want to find out the full detail in the sense the number of diamonds.

And the you know size of those dimensions, and you know dimension then, you know, we can use another metadata object and other you know attribute that is shape. So, array.shape will give us the actual shape of an array object or rather you know number of dimensions and the sizes of those dimensions. So, let us run array.shape now, as we know that this is 1 dimensional array. So, therefore, if you look at the output 10, it is just 1 value is there in this output.

And that is depicting the number of elements that are there in the 1 dimensional array. Now, if I take another example array2.shape, then in this case, you would see a tuple is return, where 2 elements are there and the first element 2 is indicating the number of dimensions. And the second element you know 4 that is indicating the number of elements in each of those dimensions. So 2,4 so that is telling us about the shape of this particular array2.

So, in this fashion we can learn about some of the attributes, some of the properties of array objects, so whether it is about the number of dimensions or the sizes of those dimensions. Another important metadata object about all property of an array object is the data type that is being stored in a particular array object. So, how do we find out that. So, we can use dtype metadata object in this case.

(Refer Slide Time: 25:08)



```
Out[376]: c    3.00
          d    6.00
          Name: Delhi, dtype: object

In [376]: # sorting by index using sort_index method (default: ascending order)
series16 = pd.Series([4, 7, -3, 2], index=['d', 'a', 'b', 'c'])

In [377]: # series16
Out[377]: d    4
          a    7
          b   -3
          c    2
          dtype: int64

In [378]: # series16.sort_index()
Out[378]: a    7
          b   -3
          c    2
          d    4
          dtype: int64

In [204]: # df6 = pd.DataFrame(np.arange(8).reshape((2, 4)), index=['three', 'one'], columns=['d', 'a', 'b', 'c'])
df6
```

So, what it does this dtype metadata object will indicate the type name whether it is float integer or string or something else, and the number of bits per element that have been used to store you know those elements. So, array.p you know, execute this array.dtype, then we will come to know about the data that has been used to store elements of this array object arr. So, if I run this, you can see in the output number 12, we get dtype float64.

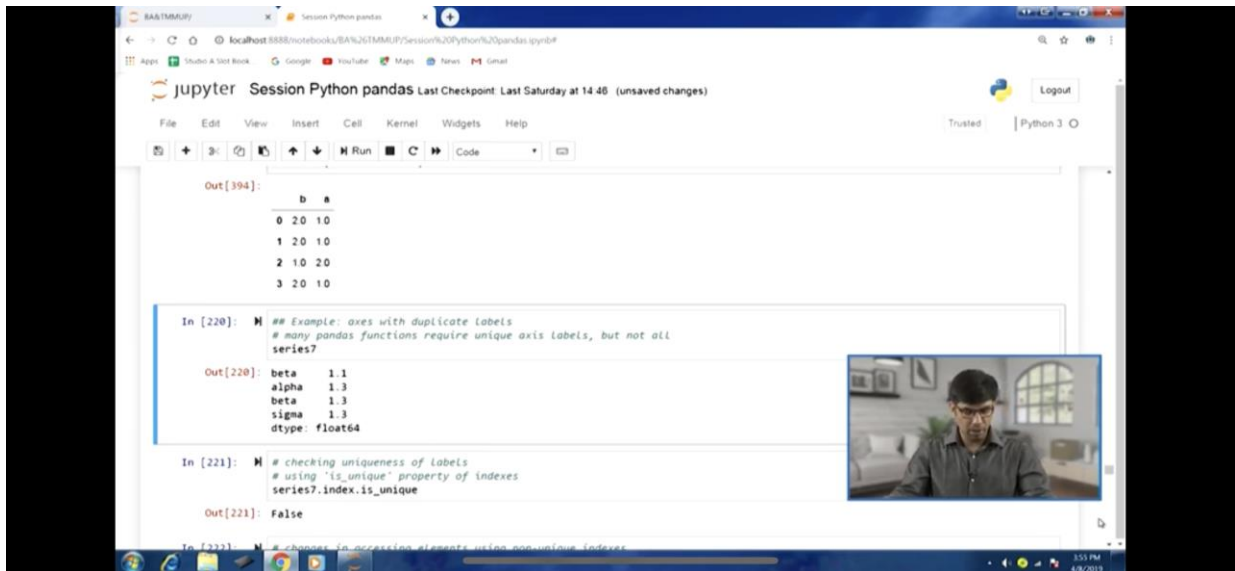
So, float is indicating the type name the data type and the 64 is indicating the number of bits that have been used to store in each element of you know this particular data type for this array object. Now, similarly, we can also run array2.dtype. So, in this case you can see dtype of float 64. So, array2 also happens to be same kind of data type. Now similarly, array3 also we can check here.

So, here you can see again, array3 was data type for the array3 object was integer 32. So, type 1 integer and 32 bits for element were used to store those elements of array object. Now, let us move on to the next aspect. So, now, let us talk about few more array creation functions that can use to create array objects. So let us talk about the zeros function. So, we can use this function again in a sense to initialize an array object.

So if I you know, execute this code np.zeros and in the only argument that passing this example is the 5. So, that is actually to indicate the number of elements in that you know 1 dimensional array that we want to create, and the array would be created and all the elements of you know this newly created array will be initialize with the value zero. So, if I learn this zeros function here np.zeros, 5. So, you can see output number 15.

And we have got this 1 dimensional array with 5 elements, and each of those elements are you know, 0.0.0.. So, from this, you can also understand that the default value for this array creation function, and the default datatype for these array creation function is the floating point. So that is why the output is coming it is being displayed as 0. something. Now if you are interested in finding out more details about these functions, different arguments.

**(Refer Slide Time: 28:57)**



And how they are to be used, that you can always do by going through the help section. So, if I click help, now, you can see python reference ipython reference NumPy reference. So, for earlier 2 sessions that we have covered the 1 on Python basics, the other 1 on the pattern built in capabilities, we were mainly using the python reference here. Now, we have started our discussion on NumPy packets.

So, we will be using the NumPy reference now, for any details about any functions matters that we would like to find out. So, if I click the NumPy reference here will get a you know NumPy documentation here and click search function, we can always go and you know, whatever, you



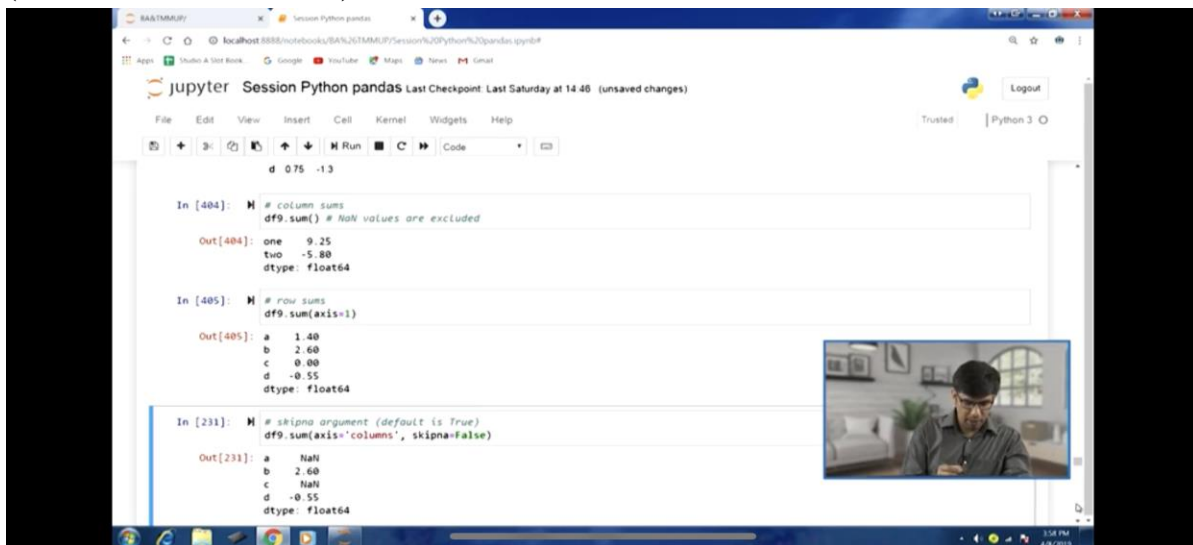
know, function that we are interested in, we can always go type that in the search. So, if we want to find out more details about the zeros function, you can just type zeros.

And you can see NumPy.zeros, this is the function, this is the python function that we are looking for. So here you can see the details, the first argument is the shape. So we are supposed to specify the shape of the function, then the dtype which you can see it is you know, keyword argument and the default values 4, then we have another you know argument order. Now let us go through the parameters these arguments, so shape int or tuple of ints.

So example could be you know, we can specify this 2,3 you know in this fashion, we can specify the shape like 2 dimensional array with 3 elements in each dimension, or you know, if it is 1 dimensional array, we can specify just the number of elements. So example that we just have run through, we had a specified you know 5. So you can see the same example here, here you can see np.zeros5 array and this is the same output we also you know got there.

So anytime when you are looking to find out more details about a certain function, you can always go to the help section, find the relevant references link, and just search it out using the big search box. So let us go back. Now, in this example, we had just passed on the just 1 number that was to indicate the number of elements for 1 dimensional array, and everything was initialized with zeros.

**(Refer Slide Time: 31:10)**



Now we can use the same function `np.zeros`. Now this time we can we are passing on, you know tuple with 2 elements 3,3. So essentially, we are trying to create a 3 dimensional array with each dimension having 3 elements. So if I run this, so you can see in the output number 16 array you can see 3 dimensions, 3 rows of data are there. So we can also think about these arrays as the, you know, rows and you know columns.

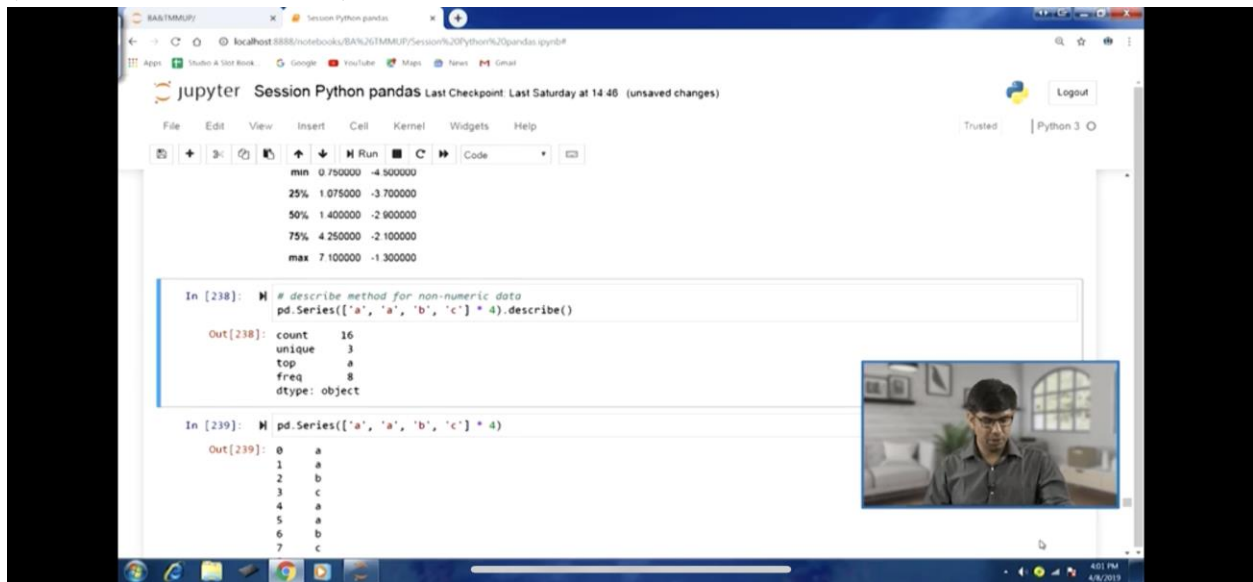
And later on, we will talk about how to access these elements of the array, just like the list object, which were zero indexed, the arrays are also zero index. So we will talk about those aspect also. So here you can look at it like rows and columns. So each row can be thought of as representing 1 dimension. So there are 3 rows in this output number 16. So 3 dimension. So the same thing we indicated in the first argument. And in the second argument, also the values 3, that is to indicate the number of elements in each of those dimension.

So if you look at the number of columns, so that would be equal to the number of elements that we like to have in each of those dimensions. So the number of rows are like number of dimensions, and number of columns are like number of elements in each of those dimensions. So, you know, in this fashion, we can understand the output or any array object. Now, let us take another example for `np.zeros`.

Now in this case, the first argument as you can recall the difference also here, you can see shape and dtype. So now we are going to use the second argument also the dtype. So, this in the first argument, we are passing a couple with the elements 2,3,3. So, this is to indicate that you know we are looking to create 2 cross 3 cross 3 arrays and the data type for the elements that are to be stored is integer that is the second argument.

So, if I run this here `np.zeros`, you can see the output. Now, here you can see that the first element you know first argument was 2 and so in a sense we can consider that it is comprising also the array this multi dimensional array with you know 2 cross 3 cross 3 cross 2 is consisting of 2 arrays of dimension 3 cross 3 cross 2, then again the next level can be considered again as you know 3 arrays of you know 3 cross 2 sub arrays.

(Refer Slide Time: 35:54)



So, in that fashion, we can think about these outputs. So, let us go back to the previous example. So, you can see here in this case 2,3,3 dtype n. So, you can see all the elements are zero failed, and you know, first you know array 3 cross 3 first sub array rather 3 cross 3 and second sub array 3 cross 3. So, in this fashion, we can understand, and the next example, that we just talked about output number 18 also in the same fashion as I discussed, you know, first element will have another sub array with 3 cross 3 cross 2.

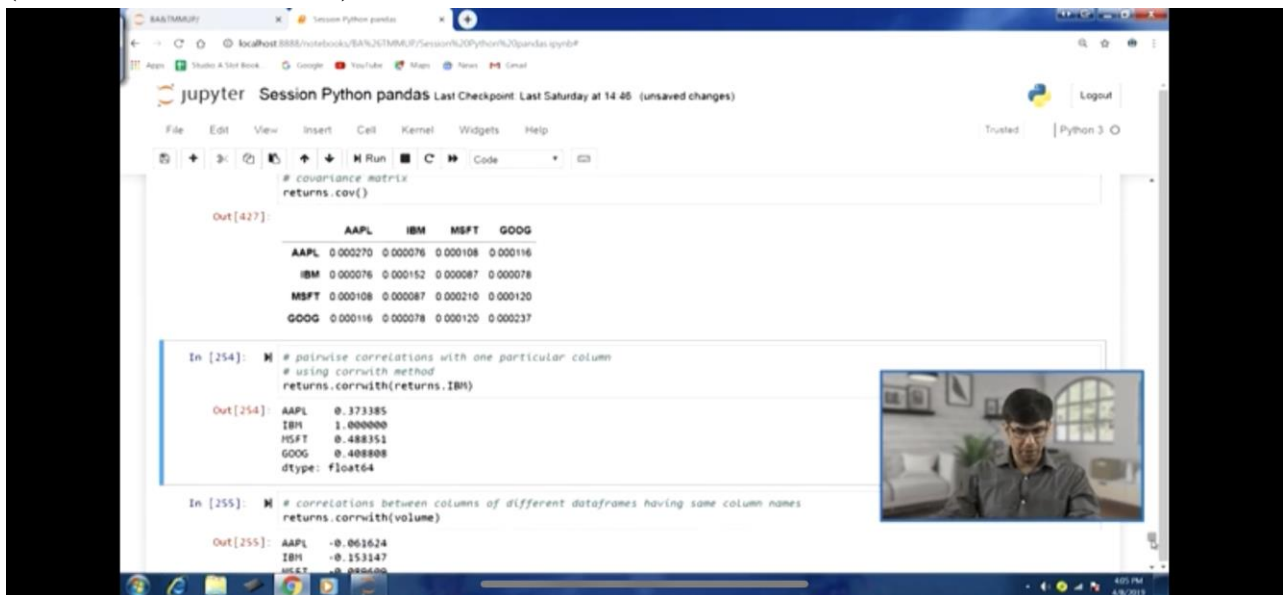
And second element will be the second 1 3 cross 3 cross 2 and the data type is integer. So, we really need to see how these arrays are to be understood, all the elements are you know, having the same data type, and when we are using these zeros functions, all of them are being filled with zeros and how the dimension aspect is to be understood rows and columns aspect that I just talked about. So, how that is to be understood has to be very clear.

Now, let us take another example, now in this will be using another array equation function that is once. So, it is very similar to the function zeros. So, in the function zeros the values were filled with zeros in this case the values are going to be filled with 1s. So, if I take this example np.ones and the only argument that we have in passing the value as 8. So, we will be creating a 1 dimensional array with 8 elements.

So, if I run this, you can see in the audience number 19, we got an array with 9 elements and each element is having the value 1 point. So, because the default is float, as we saw in the reference also. So, let us take another example for this function. So, now, this time we are passing on np.ones we are passing a tuple, which is indicating the kind of array that we want to create, which is 2 cross 4 cross 4.

So, that means, you know first, you know, first sub array would be 4 cross 4 cross array. And the second sub array would be again a 4 cross 4 array. So, in this fashion, we can understand the sub array elements and then go on further. So, if I run this, as you can see the output you can see all the values are filled with ones and we can clearly understand the dimensions, which were passed on as arguments 2 cross 4 cross 4.

**(Refer Slide Time: 38:32)**



Similarly in this function also, we can specify the second argument that is dtype to specify the data type to be used. So you can see in the output if I run this, here, you can see in the output the decimal part is 1 decimal part is you know truncated, because we change the you know data type for each element. Now, let us talk about the another array creation function that is a range function.

So we are already familiar with the range function that we have used in python built in capabilities you know for different sequences, here also a similar kind of function is available to us, which is a range function. So let us understand this. So here in this example, I'm calling this function np.arange, and the value that I am passing is 6. So here, if I run this in the output, you can see array 0 1 2 3 4 5 6 six element, you know.

So, this is creating this a range function is creating the elements in a range fashion. So, the only argument that we have passed on is the stop point, and by default has taken the start point as 0. So from 0 to you know, up to the number of elements that are to be there, 0 to 5 have been created. Just like you know, the functionality is very similar to the range function that we have understood earlier.

So, if we go back to the python reference, and type range here, and if search you can see, here again, start stop, if you look at the arguments here and start stop. So you can see started optional is top is the only mandatory argument here you know, the other arguments are optional, and the d type is the key words so again, it is optional. So in this fashion, you can see a range can be really useful sometimes to quickly create you know, arrays.

Now another example for a range function, now this time I am you know, specifying both the start and a stop point. So, you can see 5,20. So the, you know, values will start around 5 and, you know, as we have learned in the range function, the you know, stop point value will not be included. So you can see from the output if I run this output number 23, 5 to 19 this kind of sequence has been created here.

This kind of numerical series from 5 to 19 has been created, and this array object has been created. Now, let us talk about another aspect of NumPy that is the supported data type. So what kind of data types are supported in this package, so, floating point complex, integer, Boolean, string, general python object, so for all kinds of data types we have support in this package.

So we will learn this through a you know few examples. So, given these different data types, how we can convert 1 object from 1 data type into another data type. So we are talking about array here, so how we can cross an array from 1 data type into another. So for this will be using as type method. So let us take the example of array3 that we had allocated. So let us run this, this is the array3 1 dimensional array with 4 elements. And let us look at the data type for this array.

So, array3.dtype and you can see the data type is int32. Now, if we want to convert it into some other data type. So, array4 we are trying to create a new array object, and in this you know, we are using array3.as type, we are converting array3 costing array3 using another data type that is np.int 64. So instead of using a 32 bit, you know integer we would like to use a 64 bit integer here.

So, smaller into bigger int, this is the kind of costing that we are doing here. So, if I run this, and if I check array4 dtype you can see int64. However, if I look at the values, so, you know, in the output also, you can see you know in this, you know, dtype has been changed. Similarly, you know, integer to float, so here again, we can use as type, and in the argument we can pass on np.float32 if we want to use a you know, 32 bit floating data type.

So, let us create array4 using array3 and costing array3 and if I look at array4 and you can see the output earlier all the values were integer 1234. Now, we can see a decimal point also there, because the costing has happened and it has been converted into a floating point, let us move on float to int. So, whenever this kind of casting is done the decimal part is going to be truncated.

So there is going to be loss of data in this case. So we have to be aware of this. So let us take this example will cost array4 into a 32 with integer. So let us run this array 5 that is created and you can see that a decimal part is gone. Sometimes a string values might be containing the number so in the single quotes or double quotes. So let us take this example array6. So, let us first create this array using array function np.array.

So, this is the you know string object that we have and values let us output the value. So, you can see 3 elements this array with 3 elements of string type, and you can see the data type is also you know, mention slightly differently from the way we have seen till now integer floats kind of time this is a string type. And now we can use this example to convert this string you know type array into a float type.

So, you can again use the as type and here you can see in the next in line 54 rather 35 if I run this you can see as type float. So, I have not mention np.float here. So, you can always you know, because the what NumPy does is it is running arrays. So, in the sense python types are going to be converted into the equivalent NumPy you know data types. So, therefore, there is going to be no problem here.

And you can see in the output 35 array have got 3 element here you know without any issues. Now, let us take a few more examples. Now, this time in this array have you know the first element is really long floating point number many decimal points running up to many decimal points. So, let us run this and you can see this is so now here you can see because this is also using the string data type.

So you can see the data time is changing, the earlier example that here we have the smaller floating point number within you know single quarter 1.1 and all that -5.5 and 9 we had U4 because 4 bytes were used here in this case you can see 20 bytes have been used that is because of the number of characters that have been tagged into the first element in this array. So depending on the you know the way the string values are being initialize.

So that was the storey fixed stories going to be allocated for this. So you can see that is being reflected in the data type you know object as well. Now, if we you know, convert this a string array object that we have just created to float1. So if I run this next 1, so you can see array7.as type np.float and in the output, you can see that in the array7 output, we had a really long you know, floating point number within double quotes.

So, that was accurately displayed there in output 36. However, when we casted into float you can see that the value has been truncated, right. So, that is a kind of data loss. So, there is no warning for this. So one has to be really cautious about you know these kind of castings because there might result in some kind of data loss. Now, if I check the data type for this, you know, float array that he had just created.

If I run this you can see float 64 2 because we are just using 60 bits for each element that means you know, about 8 bytes, and the actual you know, array with the string values it use you know 20 bytes. So therefore, the changes clearly noticeable here. Similarly, let us take a few more examples, here you can see np.array and in this example, you can see another you know, along this value of floating point value here.

If I run this, and again, you know, if I want to cast this into the string type, so, you can see how the casting to string type is happening. So, you can see here, so, this is in a sense is happening in this binary mode, b is append everywhere. So you can see this. Now, there is another way to use data type. So, you know, we might have created objects array objects earlier also. And we would like to keep the same data type which we had used for the earlier array objects that we might have created.

So that can also be done. So let us take the example of array3, so array3 .dtype data type in this case is int32. And array4 dtype the data type in this array object is both 32. So again, I can use as a function and you know, I can cast array3 using the array4 data type. So, in the argument that I have, I can pass on array4.dtype. So this is another way to convert. So you can see that in the output number 43 the data type is float32.

**(Video Ends: 34:25)**

So, we will likely stop here and then we will continue our discussion on NumPy in the next lecture, well we will talk about the short and type quotes as well that can use to perform this kind of casting or conversion, thank you.

**Keywords: NumPy, Array, Dataframe, float, Integer, type casting, text mining, metadata.**