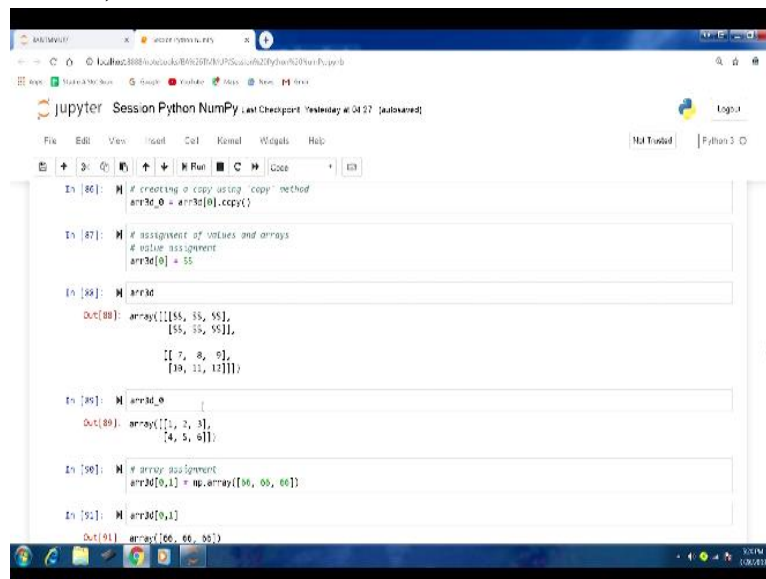


Business Analytics And Text Mining Modeling Using python
Prof. Gaurav Dixit
Department of Management Studies
Indian Institute of Technology Roorkee

Lecture-21
Numerical Python -Part III

Welcome to the course business analytics and text mining modeling using python. So in previous few lectures we have been discussing NumPy and in the previous lecture specifically we were discussing the assignment of values and arrays. So let us quickly discuss this small part, so array 3d that we had used in the previous lecture.

(Refer Slide Time: 00:47)



```
In [86]: # creating a copy using 'copy' method
arr3d_0 = arr3d[0].copy()

In [87]: # assignment of values and arrays
# value assignment
arr3d[0] = 55

In [88]: # arr3d
Out[88]: array([[[55, 55, 55],
                [55, 55, 55]],
               [[ 7,  8,  9],
                [10, 11, 12]])]

In [89]: # arr3d_0
Out[89]: array([[1, 2, 3],
                [4, 5, 6]])

In [90]: # array assignment
arr3d[0,1] = np.array([66, 66, 66])

In [91]: # arr3d[0,1]
Out[91]: array([66, 66, 66])
```

So if we want to assign you know array 3d if we take this array 3d 0 that means the first element in the first dimension. And we are assigning this value 55, so all the elements that are part of this you know first element in the first dimension they are going to be assigned as a scalar value. So you can see in the output number 88 also the same thing is there, you can see the first element in the first dimension which is a list of lists.

So that has been changed to you know all those 6 values first list 3 values, then second list 3 values and this list of list that those have been changed to 55. So this is how the value assignment will take place in a multi-dimensional array. So few more things are there array3_0 that we have recorded previously you can see the values and compare how those values have changed.

(Video Starts: 01:42)

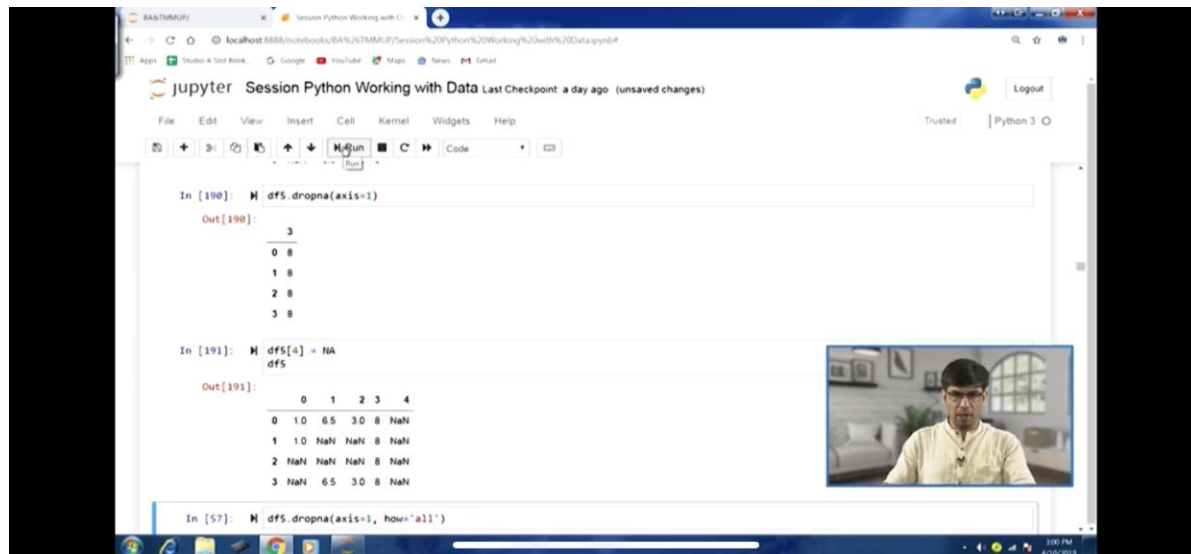
Similarly array assignment so on the right hand side now you see we have a 1d array `np.array 66, 66, 66`. So this is a 1d array which is being assigned to you know an element in the array 3d. Now what is that element in array 3d, so you can see you know 0 indicating that first element in the first dimension, then ,1 1 indicating that second element in the second dimension. So that element which happens to be if I scroll back if you see in the output 88.

So first list is 55, 55, 55. Then the second one is 55, 55 55. So first dimension these together are the first element and second dimension the second of these 2 is the second element. So that is going to be changed with these value 66, 66 66. So this is how you can have a look at it. Now here if I add another code here and if I look at the value array 3d here. So let me type this, so once this assignment take place.

So let us perform this assignment array 3d. So you can see so whatever the values were there what you can see these element this tuple list has been changed because of this assignment which is clearly visible here. So this is how this array assignment can change elements in a you know multi dimensional area. So let us move forward we can always you know we had recorded this 3d_0 before we can restore it you can have a look at the values first element restored now.

Let us move forward, so like we talked about that you know these dimensions of these array 3d 0 and array 3d 1 which we can understand now very clearly that these are 2 dimensional arrays, dimensions of these 2 would be consisting out of 2 1-dimensional arrays. So if I run this array 3d 1 0 and at a 3d 1 1, you can see these are 1d arrays. Now let us talk about slicing aspects of multi-dimensional arrays.

(Refer Slide Time: 05:41)



```
In [190]: dfs.dropna(axis=1)
Out[190]:
      3
0  8
1  8
2  8
3  8

In [191]: dfs[a] = NaN
Out[191]:
      0  1  2  3  4
0  1.0  6.5  3.0  8  NaN
1  1.0  NaN NaN  8  NaN
2  NaN NaN NaN  8  NaN
3  NaN  6.5  3.0  8  NaN

In [57]: dfs.dropna(axis=1, how='all')
```

So let us take the example of array 2-d here, so if I run this array 2d, so you can see these are the elements, this is a 2d array list of lists and 1 2 3 then 4 5 6 then 7 8 9. So if I take just a 1 slice here array 2-d and within brackets :2, so what it means is that if I run this you can see the output 1 2 3 4 5 6, 2 is the stopping point it would not be included. So therefore first 2 elements 0 and 1, they are going to be included in the output.

So that is what the output number 1 0 7 has you know first 2 elements there. Now another if I look at the dimension of this you know the slicing, so again array 2d within brackets :2 and if I look at the end n.ndim object here. So you can see clearly this is a 2d array. Now sometimes we might require multiple slicing, so how that can be performed. So first thing we need to understand that the first index that we will have in the multiple slicing.

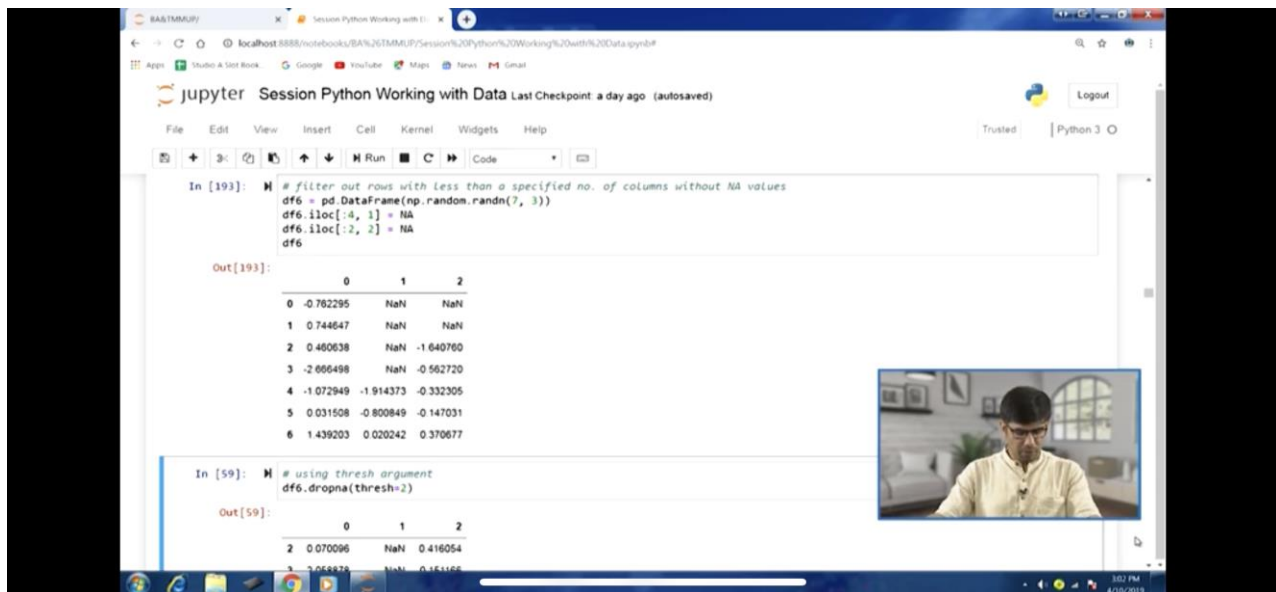
That will indicate the slicing along the first dimension, the second index that will have while performing multiple slices that will indicate the slicing along the second dimension. So if you see the example in 1 0 1 array 2d :2 that is along the first dimension then ,1: that is along the second dimension. So in the along the first dimension we have mentioned the stopping point and along the second dimension we have mentioned the starting point.

So if I run this you can see the output 2, 3 you can see here and then 5, 6. So you can see that you

know in this 0 and 1 along the first dimension and 0 and 1 these 2 lists have been taken. So that is 1 2 3 and 4 5 6. Then second slicing is where it starts with 1 that means second element onwards, the second and third. So that means 2 lists that we had selected after first slicing that is 1 2 3 4 5 6.

Now first one is you know first one is left out, so we are left with 2 3 and 5 6. So you can see how these multiple slicing is working out here, now one of these slicing could be bare slicing. So you can see in the next example at a 2d. The first slicing is bare slicing and the second 1 is where we mentioned :1 that is the stopping point. So if I run this, so first we are slicing means that in the first dimension we are taking all of them.

(Refer Slide Time: 07:38)



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell, labeled 'In [193]', contains the following code:

```
# filter out rows with less than a specified no. of columns without NA values
df6 = pd.DataFrame(np.random.randn(7, 3))
df6.iloc[:, 1] = NA
df6.iloc[:, 2] = NA
df6
```

The output, 'Out[193]', displays a DataFrame with 7 rows and 3 columns (0, 1, 2). The values in columns 1 and 2 are all NaN.

	0	1	2
0	-0.762295	NaN	NaN
1	0.744647	NaN	NaN
2	0.460638	NaN	-1.640780
3	-2.666498	NaN	-0.562720
4	-1.072949	-1.914373	-0.332305
5	0.031508	-0.800849	-0.147031
6	1.439203	0.020242	0.370677

The second cell, labeled 'In [59]', contains the following code:

```
# using thresh argument
df6.dropna(thresh=2)
```

The output, 'Out[59]', displays a DataFrame with 2 rows and 3 columns (0, 1, 2). The values in column 1 are NaN, and the values in column 2 are 0.416054 and 0.161166.

	0	1	2
2	0.070096	NaN	0.416054
3	-2.066498	NaN	0.161166

That means all 3 you know list elements and the second slicing is about that the stopping point is 1 that means just the first you know element is to be taken. So that is why you see just 1 and then 4 and then same in that fashion. So this is how multiple slicing can be performed now slice excavation could also be used for assignment purposes. So in the next example what we are doing is array 2d and we have performed you know multiple slicing :2,1:.

And whatever you know slicing happens, whatever elements are part of this slicing output they are going to be assigned this value 0 which is in the right hand side. Another way to understand

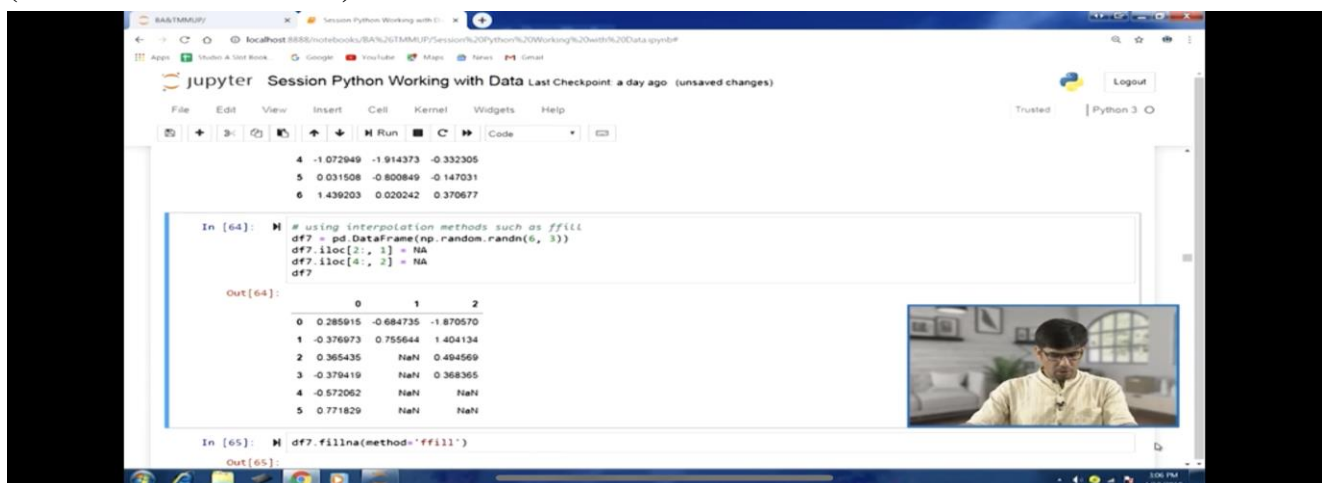
slicing is that you can consider is like if you have 3d array you can consider it like a cube. And you are just you know slicing out some of the parts of that loop that cube and the output is going to be the remaining part.

So that kind of operation in essence we are performing when we do this kind of multiple slicing. So let us coming back to this assignment, so after performing this multiple slicing the output the sliced output all the elements in that slice output they are going to be assigned this value 0 that is there on the right hand side. So if I run this and I look at the values array 2d so you can see those 4 elements that we had sliced those are now the value is 0 0 0 0 that you can see in the output 1 1 2.

Array 3d if let us look at the values again, now this 3-dimensional array this particular example also we can take to perform slicing. So let us start with the single slice 1 slice, so just :1 right, so just the you know first element here 1 2 3 4 5 6 in the first dimension. So this is what we get, now multiple slicing, so first dimension and second dimension. So in the first dimension the first element and in the second dimension also in the first element.

So as an output we will just get 1 2 3 the very first list in the array 3d. Now let us move forward, now if we look at the you can see the all these outputs that you can see which are related to array 3d starting from 1 1 3, 1 1 4 and 1 1 5. You can see in all these outputs we have you know triple brackets there, that triple brackets is always indicating that all these outputs are actually 3d arrays themselves.

(Refer Slide Time: 11:25)



The screenshot shows a Jupyter Notebook window titled "Session Python Working with Data". The notebook contains a code cell (In [64]) and its output (Out [64]). The code cell contains the following Python code:

```
In [64]: # using interpolation methods such as ffill
df7 = pd.DataFrame(np.random.randn(6, 3))
df7.iloc[2:, 1] = NA
df7.iloc[4:, 2] = NA
df7
```

The output (Out [64]) displays a 6x3 DataFrame with columns 0, 1, and 2. The values are as follows:

	0	1	2
0	0.285915	-0.684735	-1.870570
1	-0.376973	0.755644	1.404134
2	0.365435	NaN	0.494569
3	-0.379419	NaN	0.368365
4	-0.572062	NaN	NaN
5	0.771829	NaN	NaN

Below the output, there is a code cell (In [65]) and its output (Out [65]). The code cell contains the following Python code:

```
In [65]: df7.fillna(method='ffill')
```

The output (Out [65]) displays the same 6x3 DataFrame as before, but with the missing values filled using the forward fill method (ffill). The values are as follows:

	0	1	2
0	0.285915	-0.684735	-1.870570
1	-0.376973	0.755644	1.404134
2	0.365435	NaN	0.494569
3	-0.379419	NaN	0.368365
4	-0.572062	NaN	NaN
5	0.771829	NaN	NaN

The output (Out [65]) displays the same 6x3 DataFrame as before, but with the missing values filled using the forward fill method (ffill). The values are as follows:

	0	1	2
0	0.285915	-0.684735	-1.870570
1	-0.376973	0.755644	1.404134
2	0.365435	NaN	0.494569
3	-0.379419	NaN	0.368365
4	-0.572062	NaN	NaN
5	0.771829	NaN	NaN

So the same thing we can check using the `ndim` object. So 1 0 8 array 3d we are performing multiple slicing here `:1,:1` and we are checking `.ndim` here and you can say 3. So slicing any 3-dimensional array the output will also be a 3 dimensional array itself, let us move forward, now let us perform triple slicing here first slicing along the first dimension `:1`, second slicing along the second dimension `:1`, third slicing along the third dimension `:1`.

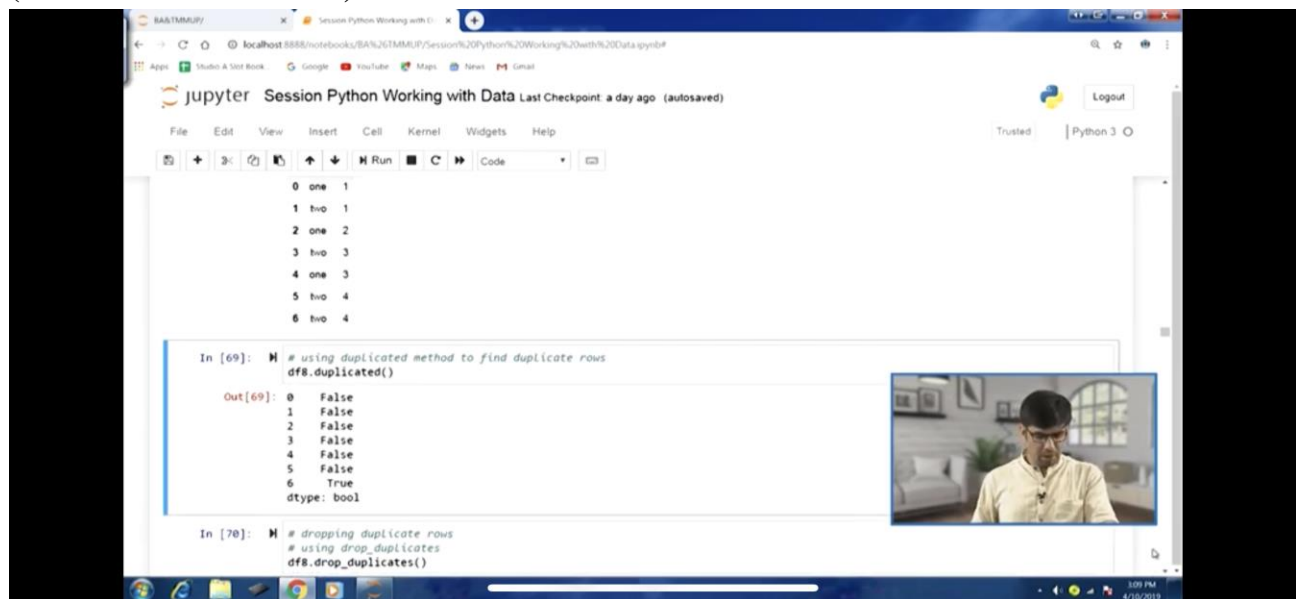
So if I run this you can see in the output it is a 3 dimensional array with just 1 value, 1 element, 1 you know scalar value there, what happened here in this case first slicing `:1` that means just the first element that means just the first list of lists, then the second slicing is also in the along the second dimension this first element that means within the first list of list first list. Then the third slicing is `:1` that means just you know first element which happens to be the first scalar value within the nested list.

So that means just 1 value is going to be there in the sliced output, so that you can see. Now we can also check the `.ndim` object for this output also, though it is having just the 1 element. This is still 3d, so 3 dimensional so if I run the `n.ndim` here then still I will get this output 3 here, you can also check the shape object here again. So if I run this 1 1 1 line number 1 1 1 and you can see the output 1 1 9 you can see this shape output this tuple 1,1,1.

So as you can see the first dimension for one only one element second dimension only one element, third dimension only one element, because there was in total just 1 element there. So shape output is reflecting that, however number of elements in this tuple are 3. So it remains a 3 dimensional array. Now can we perform for the 4th slicing we do not have a 4th dimension in array 3d it was 3d dimensional array.

But can be performed, so if we try to perform 4th slicing let us say array 3d `:1` then `:1` then `:1` and then 4th 1 as well `:1`. So if I run this I will get an error here, you can see in the error output you can see no 4th dimension, no slicing. So that is the output that we have got. Now if I perform a slicing as you can understand this stopping point is not included you know. So therefore if I say `:0` that means nothing is being selected.

(Refer Slide Time: 15:01)



The screenshot shows a Jupyter Notebook window titled "Session Python Working with Data". The notebook contains a DataFrame with 7 rows and 2 columns. The first column has values "one" and "two", and the second column has values 1, 2, 3, 4. The rows are indexed 0 to 6. The output of the `df.duplicated()` method is shown, indicating that row 6 is a duplicate of row 5.

```
In [69]: # using duplicated method to find duplicate rows
df.duplicated()

Out[69]: 0    False
         1    False
         2    False
         3    False
         4    False
         5    False
         6     True
         dtype: bool

In [70]: # dropping duplicate rows
# using drop_duplicates
df.drop_duplicates()
```

So the output will also reflect that, so in line 1 1 3, if I run this it will become line 1 1 1 2 1. Now you can see in the output 1 2 1 you know that is nil there. So you can see that, now let us talk about another aspect related to you know indexing particularly. So here we will talk about using boolean's as index values in arrays rather or in other words we can say using boolean arrays for indexing.

So let us take an example, so what we will do we will create a you know random normally distributed data in a 2d format. So we are calling array 2d1 as in the right hand side we are calling this function `random.random` and we are specifying the you know dimensions here. So it is a 2 dimensional, so just 2 values for those 2 dimension and file dimension having 5 elements, second dimension having 3 elements.

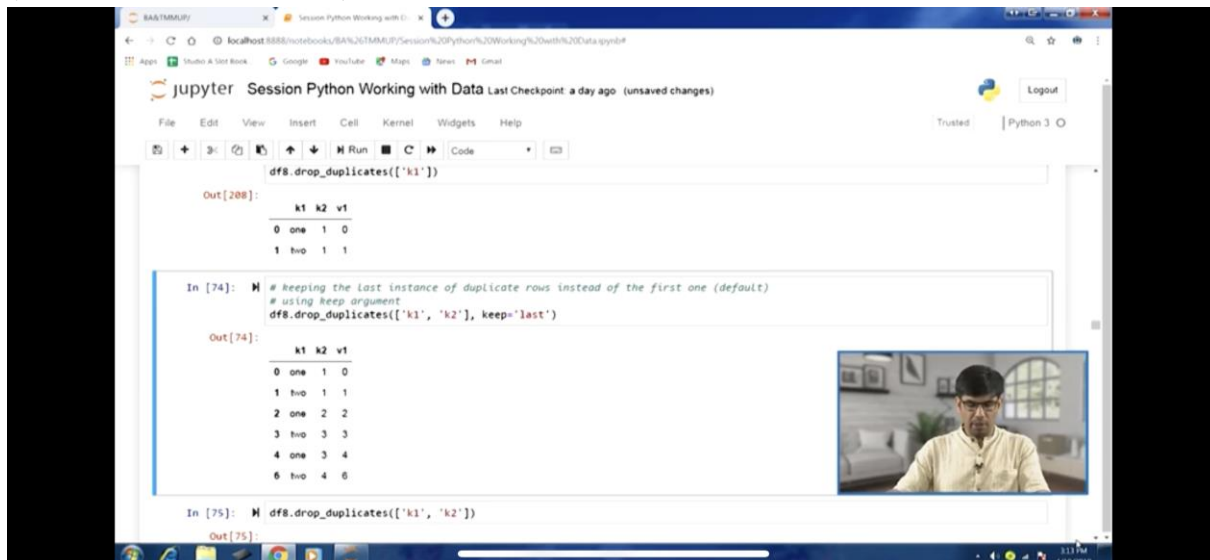
So if I run this I will get in the output 1 to 2 you can see the values here. So randomly and normally distributed values in this 5 cross 3 2d array, once we have created this array for demonstration let us look at the shape objects, it says 5 3, now let us create a boolean array. So we are getting a boolean array with called terms and on the right hand side we are calling this function `np.array`.

And the list that we are passing is comprising of these string values alpha beta gamma alpha sigma. So you can see that alpha is coming twice here, we will use this to demonstrate you know this boolean arrays indexing. Now if we look at the shape of this you know terms array, so you can see 1d array consisting of string values 5 elements. Now what is the example that we want to execute.

We want to select indices for the first dimension of array 2d1 which correspond with alpha element values in terms. So alpha you can see their comprising 5 you know values there 5 string values alpha is coming twice. So wherever alpha is coming and corresponding to those indices in the first dimension of array 2d1 you know we would like to select them. So for this to happen number of element in terms.

And number of elements in the first dimension of array 2-d 1 this would be equal which is already the case here. So we had planned that way. So 5 cross 3 is the dimensions so 5 elements are there in the first dimension in this array 2d and we have 5 values also in the terms this 1d array. So if I say terms is equal to equal to equal to alpha then if I run this you can see I am getting the output in the output number 125 as you know true false false true false.

(Refer Slide Time: 18:29)



```
df8.drop_duplicates(['k1'])
```

```
Out[208]:
```

	k1	k2	v1
0	one	1	0
1	two	1	1

```
In [74]: # Keeping the last instance of duplicate rows instead of the first one (default)
# using keep argument
df8.drop_duplicates(['k1', 'k2'], keep='last')
```

```
Out[74]:
```

	k1	k2	v1
0	one	1	0
1	two	1	1
2	one	2	2
3	two	3	3
4	one	3	4
5	two	4	5

```
In [75]: df8.drop_duplicates(['k1', 'k2'])
```

```
Out[75]:
```

So 2 true we can clearly see here because those were alphas. Now I can pass this boolean expression here within brackets for array 2d1 for indexing purpose. So 1 1 9 array 2d1 and you

can see terms equal to equal to alpha. So this will return an output just like in the output 1 to 5 and those are going to be used as indices and those then that is going to be you know depicted in the output.

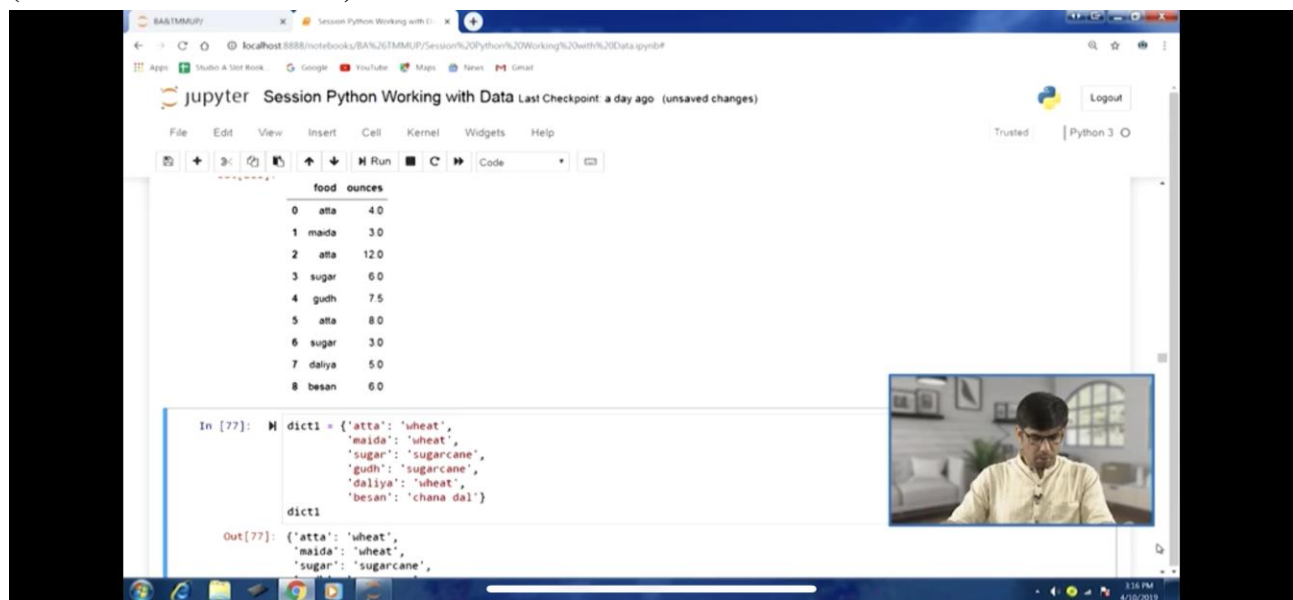
So if I run this you can see the array that we have caught. So you can see corresponding to you know alpha the values have been selected here, we can compare this with the output of this array 2d, so you can see, so first and 4th element, so if you can see first and 4th element only those have been selected. So you can compare the value the last 3 digit 5 0 7 and 9 5 7 in the output number 126.

And here you can see you know 5 0 7 and 9 5 7 you can see there. So those have been selected here clearly. Now along with this boolean you know indexing we can also combine this with the slicing. So in the next example what we are taking here is you know array 2d1. So you can see terms equal to equal to alpha in the first for the first you know dimension and in the second dimension we are slicing 2:.

So you can see here try on this and if you compare the output number 127 with 126. So you can see when we say 2 columns so starting from the second. So essentially that means the last you know last column there. So those 2 values are there in the output number 127. In addition to what boolean you know indexing that we did we can also select a pick a particular element in the second dimension.

So in this example you can see in next example array 2d 1 terms equal to equal to alpha and then just the you know index to select a particular value. So that index is 1 in this case that means a second element in the second dimension. So if I run this you can see output number 128 and you can compare this with the output number 126. So you can see clearly there the second element in the second column has been picked from there,

(Refer Slide Time: 21:27)



Let us move forward now there might be situation where you would like to select everything except a given value. So for that also we might have boolean expression like this terms not equal to alpha. So these kind of expression will give this output, let us run this so we will get output something like 129 where false true true false true. So except the indices except alpha you would like to have everything else in the output.

So if I pass this on for the indexing in array 2d 1, so this is what we will get here, so except those 2 you know those 2 rows we have got all remaining 3. So this is another way this is another scenario that can be covered using boolean expression. Now another you know tilde operator can also be used to invert a conditional statement. So sometimes this might also be useful. This is also alternative to what we just did in you know output number 130 where we used not equal to.

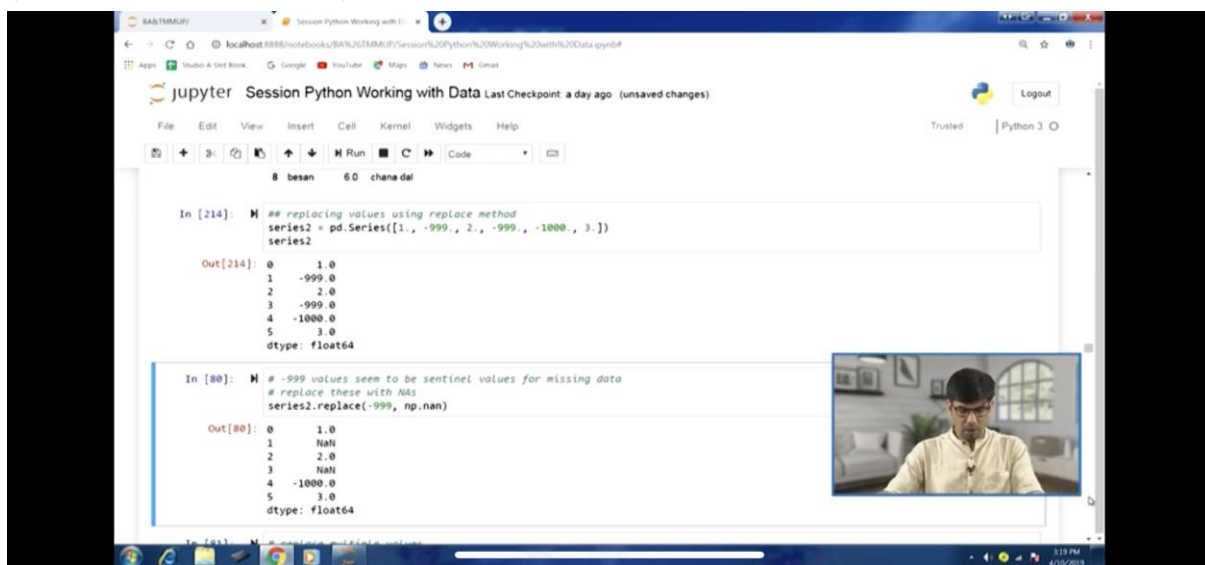
So tilde operator inverting this equal to equal to output and if I run this and you can see output number 131 and you know out were number 130, if you compare they both are same. So this is another way to achieve the same thing however tilde operator itself can be really useful in terms of inverting a particular you know output. Now there might be situation where we would like to combine you know certain you know conditions returning Booleans.

So let us understand that also. So let us take this terms equal to equal to alpha all terms equal to equal to sigma. So we can combine these 2 conditional statements here. And if I run this we will get a returned will have this array as the returned output true false false true true and we can pass this on in our 2d array and that can be used to select certain indices. So array 2d1 and this combining multiple condition this boolean expression we can pass it on in the brackets operator.

And you can see the output number 133. So those you know rows have been selected here. Now let us look at the value array 2d1 and another thing then we can perform is sometimes we would like to check whether we like to find out the values in a particular 2d array which are less than 0. So that is there in 128. So if I run this array 2d1 less than 0. So within this boolean array that we have got in the out for number 135.

Clearly it will indicate which are the values which are less than 0. So now this can really be used as indices to select indices. So you can see in the next line of code array 2d 1 within brackets we have this array 2d 1 less than 0 and if I run this the you know only the elements which are less than 0 they have been selected here in the output. Now this kind of thing can also facilitate assignment.

(Refer Slide Time: 24:10)



```

In [214]: ## replacing values using replace method
series2 = pd.Series([1, -999, 2, -999, -1000, 3])
series2

Out[214]: 0    1.0
          1   -999.0
          2    2.0
          3   -999.0
          4  -1000.0
          5    3.0
          dtype: float64

In [80]: # -999 values seem to be sentinel values for missing data
# replace these with NAs
series2.replace(-999, np.nan)

Out[80]: 0    1.0
          1   NaN
          2    2.0
          3   NaN
          4  -1000.0
          5    3.0
          dtype: float64

```

So sometimes first we would like to identify the values which are less than zero and then maybe we would like transform them to 0. So that can also be facilitated in 1 go, so you can see in the next example array 2d 1 and within brackets we have array 2d 1 less than 0 and on the right hand

side we have 0. So this can be easily performed here. So if I run array 2d 1 here you can see in the output number 138 we can see we have got this matrix here.

And you can see many values which were earlier less than 0 now they have been changed to geo they have been modified. So this is you know another useful scenario that can be useful in analytics. Now let us take another example, so in this one we are looking to identify terms which are not equal to beta and then in one go we are assigning the value 2 to all those indices. So let us run this, and let us look at the values you can see in the output number 144.

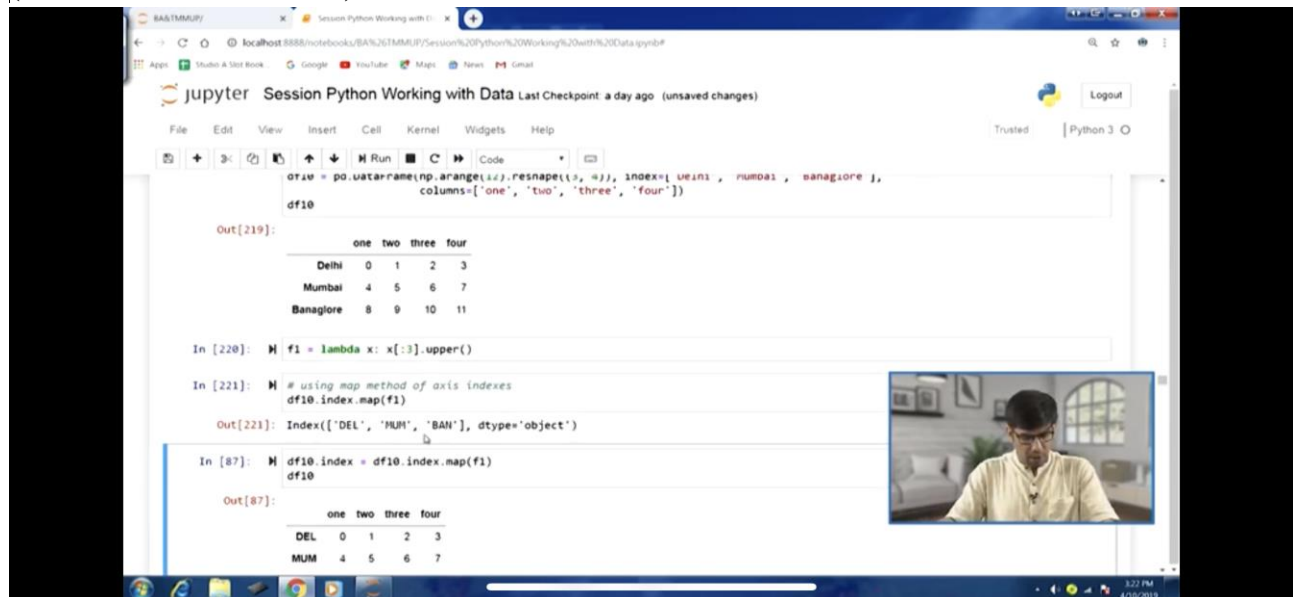
There are many values many elements which were not beta and all of them have been changed and now they carry the value of 2. So this is another way to use boolean expression, select indices and also combine it with other kind of slicing and also assignment as well. So it could be really used for in certain scenarios. Now let us talk about another aspect of this indexing. Now in this we are going to use integer list or arrays for indexing.

So let us create another 2d array to demonstrate this. So let us create this array 2d 2. Now in this we are using the empty function. Let us run this and these are the values in this empty function you can see in this empty function actually the previous array that has been you know assigned to this. Now we can assign individual values using loop. So we can run a loop because this if you look at this 2d array that we have just created this is a 5 cross 3.

So for you know each row we can just use a scalar value integer value and therefore we can change modify the values of array 2d 2. So if I run this loop and let us have a look at the output you can see first row is all 0s, then second row all 1s, then all 2s, then all 3s and then all 4s. So this are we can modify.

Now the important thing that we wanted to discuss, now for indexing we can select elements that means 1d arrays here from the first dimension of array 2d 2 using a list of integer values as indices. So you can see in this I am passing on a list of having a 4, 2, 0. So this particular list is going to be used as indices here and that will give us the output. So let us run this. So now if you compare the output number 145 with output number 144 you can see that the first you know list of integer value that we had passed on first value there is 4.

(Refer Slide Time: 27:16)



```
df10 = pd.DataFrame(np.arange(12).reshape((3, 4)), index=['delhi', 'mumbai', 'bangalore'],
                    columns=['one', 'two', 'three', 'four'])
df10
```

```
Out[219]:
```

	one	two	three	four
Delhi	0	1	2	3
Mumbai	4	5	6	7
Bangalore	8	9	10	11

```
In [220]: f1 = lambda x: x[:3].upper()
In [221]: # using map method of axis indexes
df10.index.map(f1)
Out[221]: Index(['DEL', 'MUM', 'BAN'], dtype='object')
In [87]: df10.index = df10.index.map(f1)
df10
```

```
Out[87]:
```

	one	two	three	four
DEL	0	1	2	3
MUM	4	5	6	7

So that essentially means 5th element in array 2d 2 as you can see in output number 144, all 4s so that is now coming first, then the second value in the list was 2 and you look at the output 144 which essentially means third element all 2s. So that is the second element in the output 145 similarly for 0 all 0s. So you can see in a sense using these list of integer values we have done a reordering of those elements in that array 2d 2 list.

So a kind of reordering, so we have still got a 2d array here and but the reordering of those rows have happened here. So this is how these you know list of integer values can really be useful. Now instead of a list of integers values we can also use an integer array as an index. So in the next example what we have array 2d 2 now you can see within brackets I am passing an integer array for indexing.

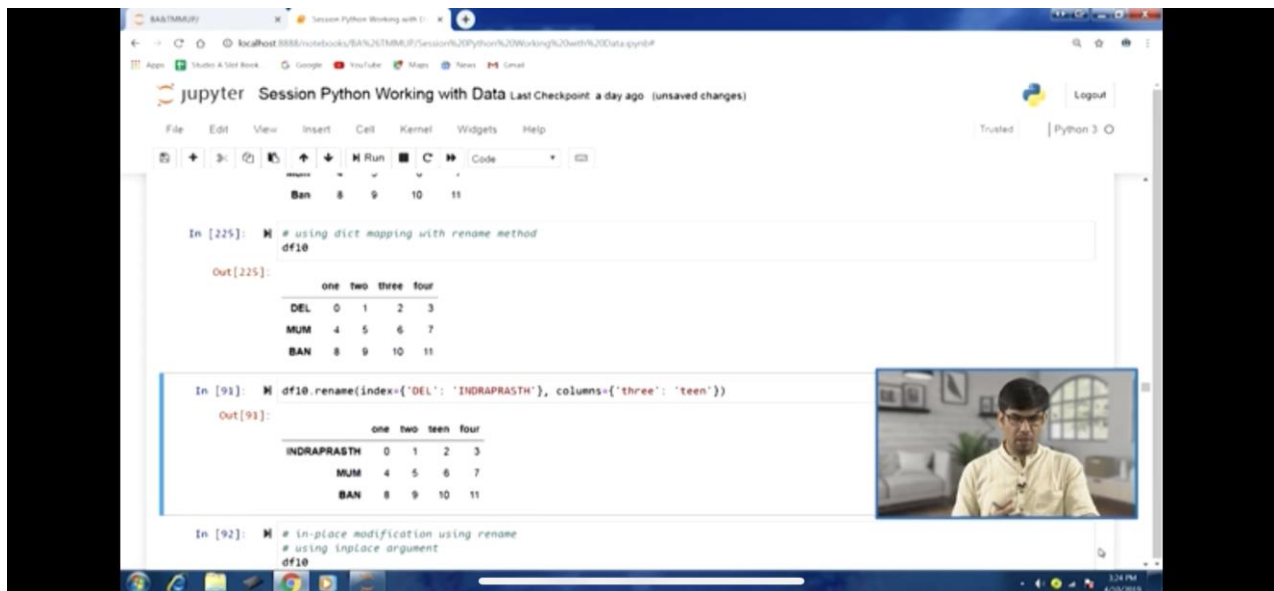
So you can see np.array and within brackets again same thing 4 2 0, so you can see the output is going to be same. So whether we pass a list of integer values or whether we pass an integer array for indexing both are going to give us same kind of output. Now let us talk about the negative indices when we are using this list of integer values. So you can see the output will change now instead of 4 2 0 we are passing -4 -2 0.

So if I run this here and output number 147 you can see now the counting is from the end. So if you compare this from the output number 144, so you can see that if you do the counting from the end first you know once will come then for -2 the 3s will come and for 0s it is there. So you know in this fashion it will happen. Another example where we are going to use you know some you know list for indexing.

So before that let us create this another 2d array, so here we are using another method v shape method. So this is typically this particularly v shape method is typically used to fix dimension. So here if you see this what I have in this line of code array 2d 3 and in right-hand side I am using a range function. So value is 32. So the values would be from 0 to 31. So I am going to create a value an array with value 0 from 0 to 31 and with 8 cross 4 dimension.

So that reshapes method that allows us to pass on that dimension the desired dimension that we want. So if I run this and let us look at this output 149 you can see we have got an 8 cross 4 you know array this is a 2d array and the values are from 0 to 31. So in this fashion we this is another way to create an array or you know v shape. Now let us talk about the next aspect which is using multiple lists of indices.

(Refer Slide Time:29:09)



```
In [225]: # using dict mapping with rename method
df10

Out[225]:
```

	one	two	three	four
DEL	0	1	2	3
MUM	4	5	6	7
BAN	8	9	10	11

```
In [91]: df10.rename(index={'DEL': 'INDRAPRASHTH'}, columns={'three': 'teen'})

Out[91]:
```

	one	two	teen	four
INDRAPRASHTH	0	1	2	3
MUM	4	5	6	7
BAN	8	9	10	11

```
In [92]: # in-place modification using rename
# using inplace argument
df10
```

So till now we talked about the list of indices with just 1 listing that we had used it is also

possible to you use multiple lists for you know indexing which is also referred as fancy indexing. So in this case it is important that number of index values that are there in the list in both the lists this would be equal and you know first list is for you know first dimension and the second list is for second dimension.

So the operation the kind the way this operation takes place it is very similar to a specifying a cell in Excel. So array 2d 3 and you can see v in the brackets we are passing 2 list here one consisting of values 1 3 5 which are going to be in indices and the second is 0, 2, 3. If I run this it is something like this like along the first dimension we have taken the you know second element the index value is 1.

That means second element and in the second dimension we have taken index value 0 that means first element. So that means that you know that elements happens to be 4 and similarly the next matching. So in a sense matching is happening here the next one will become corresponding 2 3 and 2s it will be 14 and the next one will become 23. So a matching kind of thing has happened here. This is called fancy indexing multiple lists are being used here.

Similarly you know we can take another example here array 2d 3 this is passed pass 1 you know list of you know values that for indexing purpose. So 1 3 5 you can see that second row and 4th row and 6th row they have been selected. Now we can also use it in another example where we are you doing the multiple indexing so first one is 1 3 5 that means the output number 151 we can refer to.

Then in the second one second brackets we have you know multiple lists there one is bare slicing in a sense the second one is list of integers. o again a matching will happen there, so you know along the first dimension we are taking all the elements in the second dimension we are having 0 2, 3. So if I run this you will get this kind of output. So this is you know another way to actually perform this multiple listing.

(Video Ends: 32:00)

So we would like to stop at this point and in the next lecture we will start our discussion on transpose of an array, so let us stop here, thank you.

Keywords: Multiple Indexing, Classification, Boolean Expression, Multi-dimensional array.