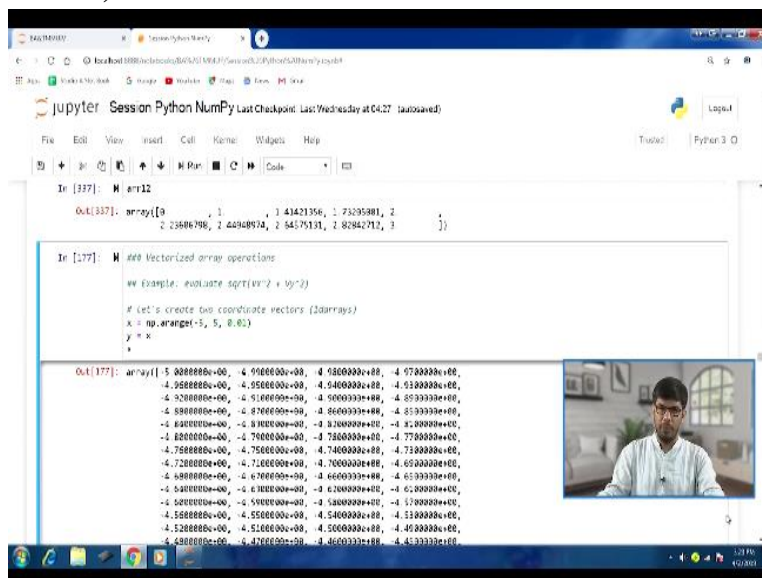


Business Analytics And Text Mining Modeling Using python
Prof. Gaurav Dixit
Department of Management Studies
Indian Institute of Technology Roorkee

Lecture-23
Numerical Python - Part V

Welcome to the course business analytics and text mining modeling using python. So in previous few lectures we have been discussing NumPy. So let us start our discussion from the point you know we stopped in the previous lecture.

(Refer Slide Time: 00:40)

The screenshot shows a Jupyter Notebook interface with a browser window. The notebook title is "Jupyter Session Python NumPy Last Checkpoint: Last Wednesday at 04:27 (autosaved)". The code in the cell is as follows:

```
In [337]: arr12
Out[337]: array([0, 1, 1, 41421356, 1, 73295881, 2,
                23886798, 2, 44908912, 2, 54575131, 2, 82942712, 3])

In [177]: ## Vectorized array operations
## Example: evaluate sqrt(vx^2 + vy^2)
# Let's create two coordinate vectors (1d arrays)
x = np.arange(-5, 5, 0.01)
y = x
#
```

The output of the second cell is a large 1D array of floating-point numbers, representing the square root of the sum of squares of the two vectors. A small video inset of a man is visible in the bottom right corner of the notebook window.

So we will talk about the vectorized array operations, so it is about the element wise operations it is about the vectorization aspect that facilitates faster execution and avoids you know slightly more lines of code using loops and other things that we typically can do using built-in and basics capabilities in python. So let us take this example suppose you know we want to perform this, we want to evaluate this you know particular expression where we would like to determine the square root of these 2 you know vectors which are essentially 1d arrays here.

So vx square you know +v2 vy square you would like to take a square root here for of these you know summation of these 2 terms these 2 vectors which are essentially. So we can consider this you know these 2 you know as 2 coordinate vectors 1d arrays. So let us take, let us initialize x as using a range function -5 to 5 and increment step would be 0.01. So essentially will be creating

1000 values here x having 1000 elements 1000 values and y also will keep it same because the idea is just to demonstrate this aspect.

(Video Starts: 02:02)

So let me run this and you can see in the output with the 8 and you can see this x, this 1d array this vector or coordinate vector has been produced and y will also be same. So you can see this is quite long 1d array with you know 1000 values right. So you can see this now here, now if you want to look at the shape of this because I said 1000 values. So let us confirm this x.shape you can see 1000 now you know elements there in this 1d array.

Same thing we can do with the y also y.1000 values just to confirm what we wanted in the first place. So now what we will do is we will create the coordinate matrices the 2d errors from these coordinate vectors. So matrices as we have indicating these are you know similar to 2d arrays. So these coordinate matrices will be 2d arrays and the coordinate vectors 1d array. So problem using these coordinate vectors will produce the coordinate matrices here.

So let us say vx,vy on the left hand side of the this line of code and you are using this function meshgrid function where we can pass on these coordinate vectors x,y and it will just produce a simple matrix kind of format using these 2 vectors. So let me run this and the functionality of this purpose of function would be more clear. So vx,vy in this fashion if I run this and let me have a look at the value vx here and you can see what we have in the vx so essentially if you see what we had in x.

So that has actually become a row here, this is a 2d array and that has become a you know first row here and these rows are being repeated. So in a sense you know and the same vector has been used to created this mess this grid kind of thing using the same value. So all the rows are having the same values and we have just you know regenerated you know this 2d array using that 1d array by repeating 1d array multiple times as per the size of the second dimension.

So this is vx same thing you can confirm you think vx.shape. So you can see 1000, 1000. So this is the you know you can this is the size of these 2 dimension of this 2d array. Similarly for vy

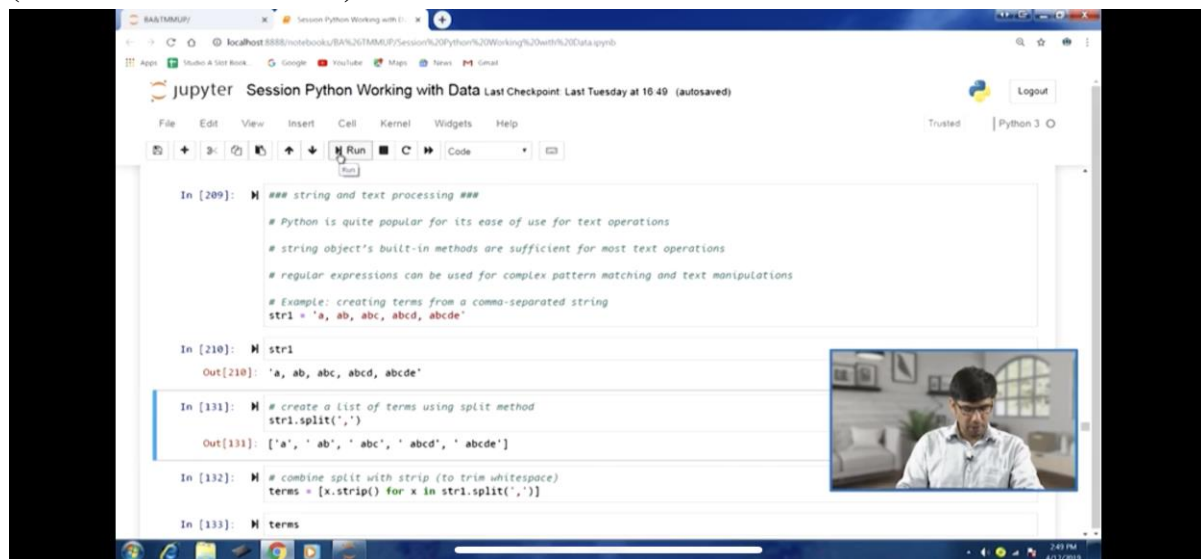
you also if I run `vy` you can see here slightly different you can see here the column wise. So whatever vector that we had that has now been you know generated in a column wise fashion you can see.

So a different you know matrix, so using this `meshgrid` function we can you know produce you we can use 1d array and produce 2d arrays but slightly different output 1 row wise the another one column wise. Now we can recheck the shape of this also you can see 1000 1000 and now we can use these square to function `sqrt` function to take this square root of the these 2 terms `vx` to the power 2 and `vy` to the power 2 and summation of these 2 terms.

And we can take this square root here, so let me run this, so we will get this output. And again let us see the output here. So you can see that in the output number 3 4 7 for you know each of so this is an element-wise you know kind of element wise operations, so you know each element from `vx` and corresponding element from `vy` has been taken is quiet and then summed up and then this square root has been taken.

So in this fashion you can remember the values were around 5, so let us say you know it would be around 5 to the power 2, so around 25 and `vy` around. So it will be around 50 and however you take a square root so the values will be around 7 something. So you can see most of the values that we have got in this output number 347, they are 70s values around 7. So in this fashion we have been able to you know perform this operation.

(Refer Slide Time: 02:32)



```
In [209]: # string and text processing ###
          # Python is quite popular for its ease of use for text operations
          # string object's built-in methods are sufficient for most text operations
          # regular expressions can be used for complex pattern matching and text manipulations
          # Example: creating terms from a comma-separated string
          str1 = 'a, ab, abc, abcd, abcde'

In [210]: # str1
          Out[210]: 'a, ab, abc, abcd, abcde'

In [131]: # create a list of terms using split method
          str1.split(',')
          Out[131]: ['a', ' ab', ' abc', ' abcd', ' abcde']

In [132]: # combine split with strip (to trim whitespace)
          terms = [x.strip() for x in str1.split(',')]

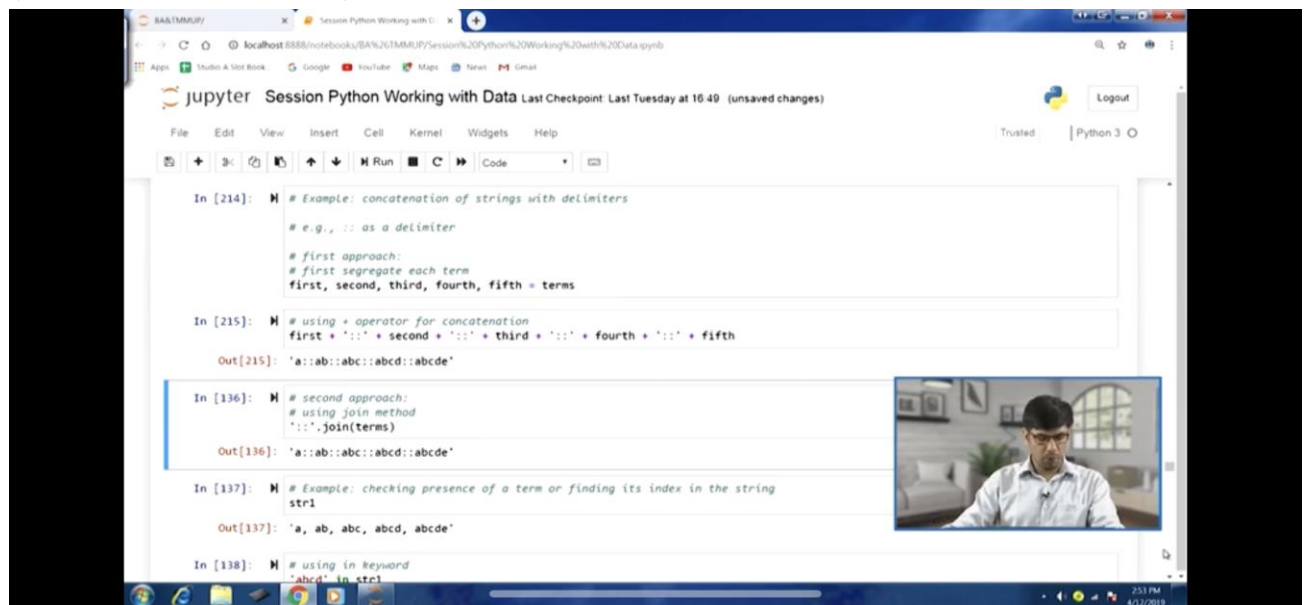
In [133]: # terms
```

So this is a vectorized operation and this is one example that we have got, we look at the shape of v then this is 1000/1000. So we had those 1000 values there 1000 the same shape there in v_x same shape there in v_y and the output also we have got the corresponding you know size here. So you can see the element wise operation has taken place the vectorized kind of operation has taken place.

Now let us take another example where you know we might be supposed to evaluate conditional statements. So let us create 2 integer arrays and 1 boolean array we would be using these arrays to you know to take an example for conditional statement. So let me slice these arrays here, so x_1 you are using `np.array` and you can see 5 elements in this now y_1 and then b_1 so just some values here and let me define them and slice them.

Now we will perform an operation like this list comprehension the next thing, so what we are doing in this list comprehension we are picking x_1 if b_1 is true and else pick y_1 . So in this fashion we would be you know creating this you know list comprehension this list of values. So we are calling this `res1` and let me run this and let us see the output you can see the values here 1.1 2.2 and so you can now you know check that back with the example you know values that we had in x_1 y_1 and come from the output also.

(Refer Slide Time: 06:33)

A screenshot of a Jupyter Notebook interface. The browser address bar shows a localhost URL. The notebook title is "Session Python Working with Data". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for running, saving, and other actions. The code area shows several input cells with Python code and their corresponding outputs. The code demonstrates string concatenation using delimiters, the '+' operator, and the join() method. It also shows how to check for the presence of a substring in a string using the 'in' keyword. A small video inset in the bottom right corner shows a person speaking.

```
In [214]: # Example: concatenation of strings with delimiters
# e.g., :: as a delimiter
# first approach:
# first segregate each term
first, second, third, fourth, fifth = terms

In [215]: # using + operator for concatenation
first + '::' + second + '::' + third + '::' + fourth + '::' + fifth

Out[215]: 'a::ab::abc::abcd::abcde'

In [136]: # second approach:
# using join method
'::'.join(terms)

Out[136]: 'a::ab::abc::abcd::abcde'

In [137]: # Example: checking presence of a term or finding its index in the string
str1

Out[137]: 'a, ab, abc, abcd, abcde'

In [138]: # using in keyword
'abcd' in str1
```

Now what we will do is will introduce another you know another function which is very useful np.where function, this is a vectorized version of ternary expression that we have been using. So this output that we have just produced using list comprehension can also be produced using this np.where which would work like a you know element which will perform element wise operations.

And vectorization of the you know operation will happen and the same result will be able to achieve the same result. So let us understand what happens in this function np.where. So first argument is a conditional statement and depending on the output of this conditional statement which is going to be a boolean output you know. So true or false, so true then second argument will have the associated results.

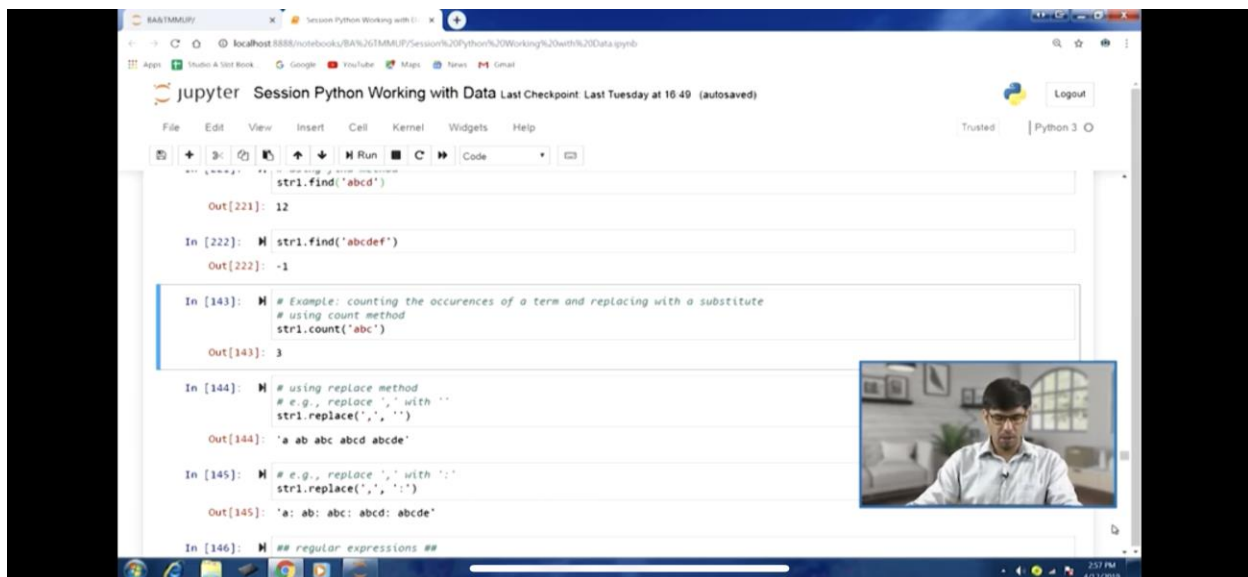
And the you know false then the third argument will produce the associated results. This is how np.where is going to work. So we look at the functionality wise quite similar to list comprehension however this is a vectorized version of the same right. So this can work with you know n dimensional arrays those kind of you know extended data structures in a sense. So what we are doing here is we are recording reg1.

And on the right hand side we have `np.where` and `b1`, `x1`, `y1`. So here a vectorized in a vectorized fashion this will happen. So if I run this and you can see `reg1` we have the same output now **to** to demonstrate the power of this we will use it again later on in a you know you know 2d arrays or 3d arrays using 2d arrays or 3d arrays. Now let us focus on another aspect we will create a new array where we would be conditionally transforming another array.

So we will use `where` function again and conditionally transform another array. So let us create let you know create this array array 13 for to demonstrate this example array 13 these are the values 4 cross 4 it is a 2d array here and if we type this the conditional statement array 13 greater than 0 or not. So if I run this we will get a boolean array here boolean 2d array here you can see false true all those you know values here.

Now we can use `np.where` function so what we will do in this will set positive values to 2 and negative values to -2. So you can see `np.where` the first conditional statement same that we had just on array 13 greater than 0. So it will give us the boolean output and you can see it is 2d no more we are dealing with 1d.

(Refer Slide Time: 10:22)



```
str1.find("abcd")
Out[221]: 12

In [222]: str1.find("abcdef")
Out[222]: -1

In [143]: # Example: counting the occurrences of a term and replacing with a substitute
# using count method
str1.count("abc")
Out[143]: 3

In [144]: # using replace method
# e.g., replace ',' with ''
str1.replace(',', '')
Out[144]: 'a ab abc abcd abcde'

In [145]: # e.g., replace '.' with ':'
str1.replace('.', ':')
Out[145]: 'a: ab: abc: abcd: abcde'

In [146]: ## regular expressions ##
```

However when we talk about the vectorizer operation using NumPy functions we can perform element wise operations and therefore it is a kind of vectorized operation we can deal with you know n dimensional data. So in this case it is 2 dimension. So array 13 greater than 0 and for the

true positive values the output is going to be 2 for the negative values the output is going to be -2.

So if I run this you can see output 357 and the array output you can see all the values have been changed to either 2 or -2 depending on the boolean output right. So in this fashion you can see that using a conditional statement we have been able to transform the array. So this could be really useful in you know analytics context when we are forming certain data processing you know certain cleaning or some other operations manipulating data. So this could be really useful.

Now let us take another example, now this time we will set positive values to 2 and leave others as is. So in this case in the np.where function the first argument is array 13 greater than 0. So any values is positive is true, then we will assign 2, otherwise we will keep the original values. So you can see array 13 we have indicated the original array to tell the function that will keep the value as is.

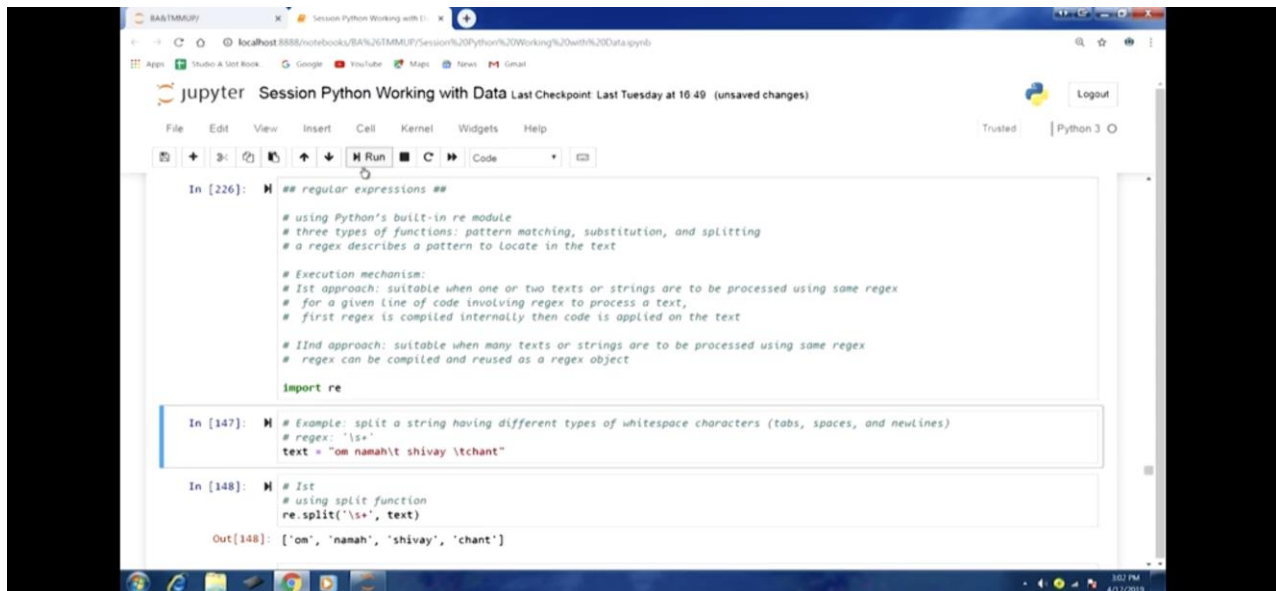
So if I run this and you can see the values changing here any value in this 2d array 13 for anything which was greater than 0 that has been changed or to anything which was otherwise it has remained it has kept its original value, you can compare this output 358 with the you know output 355 you can see the first value itself is unchanged. So that is how you know you should always confirm whether you are getting the output that you are you have to charge.

So it is always better practice to compare the outputs. Now let us take another example which is focusing on the mathematical and statistical operations. So you would now start getting the idea how this python and these packages are really useful in performing these operations because mathematical and statistical operation from the core of any analytics you know course. So you would start getting this idea from here onwards.

So you can see now we are taking this example array 2d 6. We are initializing this using random function, so let me run this these are the values. Now if I want to compute the mean value across all elements you know, so we can use mean method here so array 2d6 taught me so for all the values all the elements that are part of this 2-dimensional array. So if I call this mean method so I

will get this mean value here as an output.

(Refer Slide Time: 15:16)



The screenshot shows a Jupyter Notebook titled "Session Python Working with Data". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for running, saving, and other actions. The code in the notebook is as follows:

```
In [226]: ## regular expressions ##  
  
# using Python's built-in re module  
# three types of functions: pattern matching, substitution, and splitting  
# a regex describes a pattern to locate in the text  
  
# Execution mechanism:  
# 1st approach: suitable when one or two texts or strings are to be processed using same regex  
# for a given line of code involving regex to process a text,  
# first regex is compiled internally then code is applied on the text  
  
# 2nd approach: suitable when many texts or strings are to be processed using same regex  
# regex can be compiled and reused as a regex object  
  
import re  
  
In [147]: # Example: split a string having different types of whitespace characters (tabs, spaces, and newlines)  
# regex: '\s+'  
text = "om namah\t shivay \tchant"  
  
In [148]: # 1st  
# using split function  
re.split('\s+', text)  
  
Out[148]: ['om', 'namah', 'shivay', 'chant']
```

Now we can achieve the same output using the mean function NumPy mean function here. So np.mean if I pass on this array here. So the same output will be able to achieve, however this is this similarity ends here where we are just you know using all the values. However if we want to you know let us take another example where it involves all the elements, so we will compute some across all elements using some method.

So again sum and we have got the value now if what if we want to take values along first column. So in this fashion we can take array 2d6 and the bare slicing and 0, so it will get the first column here. So you can see now we can use again the mean method here and we can pass on the axis if we want to compute mean along this first dimension and the column wise in. So if I run this I will get the output here all right so along the first dimension. Now if I if I am looking for values along first row.

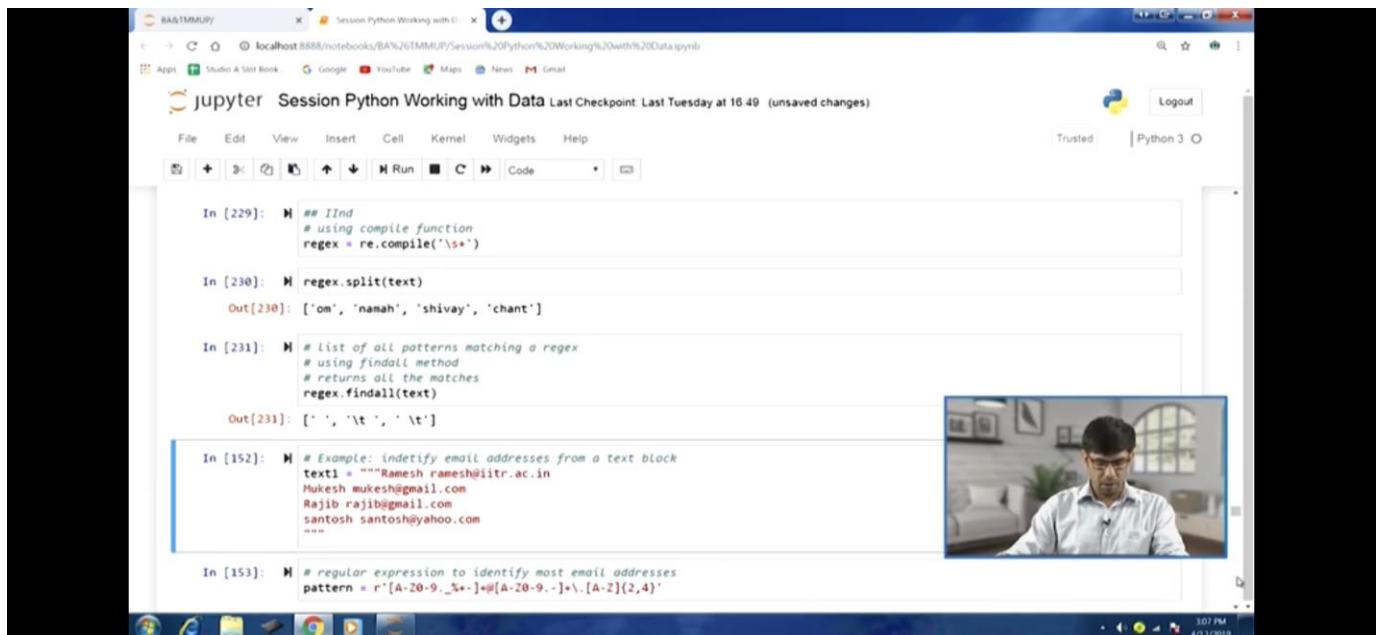
So I can access like this array to d6 and within bracket 0. So we will get these values here and if we want to compute mean along associated elements of second dimension then you can see in the mean method we are changing the axis argument here x=1. Now it will help us compute the you know mean along the you know second dimension here row wise thing.

So say if I run this I will get these mean values for you know along the second dimension here and we can use array cumsum method also to compute the cumsum values. So if I just you know run this array 2d4 here these are the values if I run array 2d4.cumsum you can see how the cumulative sum is being produced here that you can see. Now we can also use the cumsum method by indicating the axis along any axis column wise then axis 0.

So that can also be done and the same method cumsum we can pass axis 1 so it will perform row wise thing cumulative sum row wise similarly come product is cumulative product. So again we can pass on the axis and we can compute the cumulative product here. So array 2d4.cumprod and axis 1 and you can get the output. So in this fashion we would be able to compute the values along any dimensions.

So different statistics that we just demonstrated here. Now sometimes we would be required to work with boolean arrays. So how that can be done, so let us focus on few examples here. So let me create this array 1d in this 1-dimensional array here. And let me create an boolean array out of this if I run array 1d greater than 0 you can see we got the boolean output here and we can you know if we count positive values we can do in this fashion array 1d greater than 0.sum.

(Refer Slide Time: 19:38)



```
In [229]: ## Iind
# using compile function
regex = re.compile('\s+')

In [230]: regex.split(text)

Out[230]: ['om', 'namah', 'shivay', 'chant']

In [231]: # list of all patterns matching a regex
# using findall method
# returns all the matches
regex.findall(text)

Out[231]: [' ', '\t ', '\t']

In [152]: # Example: indentify email addresses from a text block
text1 = """Ramesh ramesh@iitr.ac.in
Mukesh mukesh@gmail.com
Rajib rajib@gmail.com
santosh santosh@yahoo.com
"""

In [153]: # regular expression to identify most email addresses
pattern = r'[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}'
```

So this is you know certain statistics that we are computing just for the demonstration purpose. Now sometimes you might be required to check whether one or more lose our true. So that we can again we can use this any method that will help us find out you know in the boolean array that we might have whether there is any true value that is present. So let us take b1 so this is the output now if we use this method b1.any.

So this any method will give us this output here true, now we can if we want to check whether all the values are true or not then we have another methods at all, so we can call b1.all and we will get the output here. So you can see all these you know arrays that we have been talking about some of these are 2-dimensional and these methods are working with these 2-dimensional arrays and performing those mathematical and statistical operation and giving us the output.

So you can see how it becomes easy to use these methods array methods and perform these you know n dimensional multi dimension element wise multi-dimensional you know operations.

So now let us take another example here, now these 2 methods that we just talked about any and all methods though we demonstrated using boolean arrays they can also be used with the non boolean arrays.

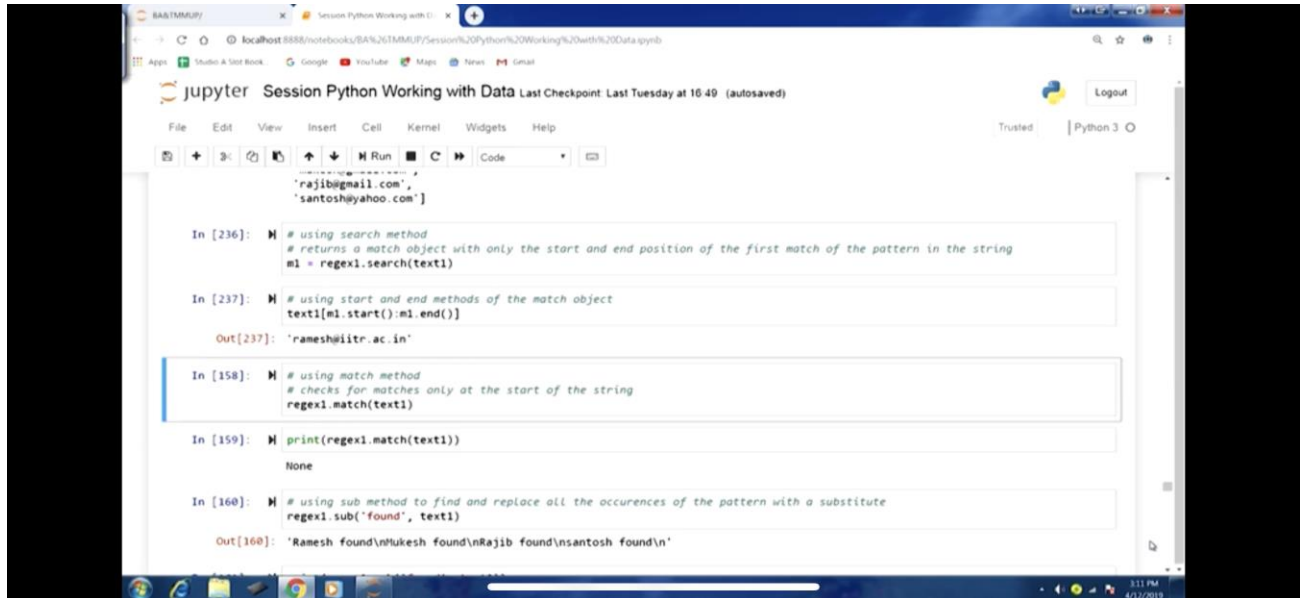
So let us take array 1 3 and if we apply array 13. any here, so if there are any nonzero values that are present in this array 13 which is a 2d array. So output would be if the non zero values are present the output would be true. So in this case again of course there are nonzero values. So true is the output, whether all values are nonzero values here in this case, so we can use all method here.

So you can see because that is the case, so again we will get true output so these methods any and all method can also work with non boolean arrays. Now let us talk about another aspect which is you know very important if you data processing context in the analytics context that is sorting. So we might have a sequence of values that we would like to sort. So let us understand how we can perform this here.

So let us take this is example array 1d, so these are the values. So sort these values to perform in

place kind of sorting we can use the sort method so we can call array 1d.sort and we will produce in-place sorting. So in the array 1d itself from the output is going to be stored. So let me run this array 1d.sort and you can see the output the array has been sorted and the original array has been replaced with the sorted copy of the you know itself.

(Refer Slide Time: 24:01)

A screenshot of a Jupyter Notebook interface. The browser address bar shows 'localhost:8888/notebooks/SAT%20MMUP/Session%20Python%20Working%20with%20Data.ipynb'. The notebook title is 'Session Python Working with Data' with a 'Last Checkpoint: Last Tuesday at 16:49 (autosaved)' note. The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar has icons for new, open, save, undo, redo, run, and code. The code area contains several cells. The first cell defines a list: `['rajib@gmail.com', 'santosh@yahoo.com']`. The second cell (In [236]) uses `regex1.search(text1)` to find a match in 'ramesh@iitr.ac.in', returning a match object. The third cell (In [237]) uses `text1[m1.start():m1.end()]` to extract the email address, resulting in 'ramesh@iitr.ac.in'. The fourth cell (In [158]) uses `regex1.match(text1)` to check for a match at the start of the string, returning `None`. The fifth cell (In [159]) prints the result of `regex1.match(text1)`, which is `None`. The sixth cell (In [160]) uses `regex1.sub('found', text1)` to replace all occurrences of the pattern with 'found', resulting in 'Ramesh found\nRakesh found\nRajib found\nsantosh found\n'.

So this is how we can perform sorting. Now sometimes we might want to do perform the you know sorting along the dimension, so that can also be done using this you know sort method. So let us again look at array 30 and if we want to do sorting along a particular axis let us say axis 1 that is 2nd dimension. So array13.sort and within parenthesis axis1 and you would see in the output the sorting has been done you know along the second dimension here.

So you can clearly see that you can see the all the rows you know they have been sorted. Similarly you know we have np.sort function also, so that will also perform this kind of sorting. So it will actually return a sorted copy rather than doing in-place sorting. So let us take this example array 2d6 here, this is the output. And we can actually perform the sorting here np. sort array 26 now we are passing this as an argument.

And if I run this you can see all the values they have been sorted here clearly. So this function can also be really useful to perform this kind of operation. Now we maybe have earlier discussed

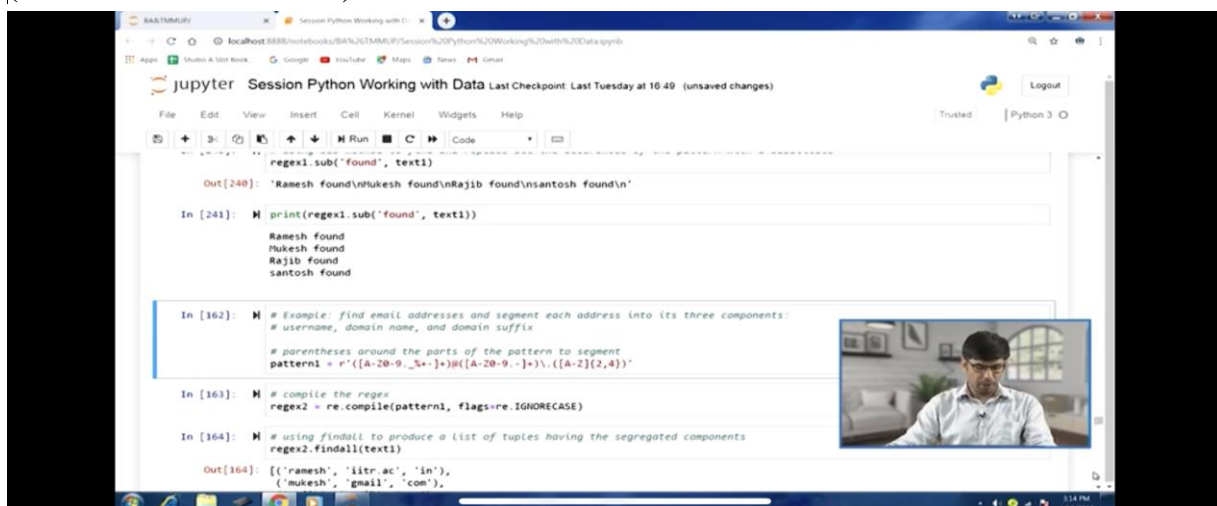
about the quantiles and the sort function methods and function that we have just discussed they can be really useful to compute quantiles. So let us take a few examples. So let us compute 50% quantiles.

So what we will have to, what code we will have to write to perform this. So let us take this array 1d and you know within the brackets you can see that what we are indicating here is that int type and 0.5 for the 50% percentile multiplied with length of this array 1d. So in this fashion you know we will get this value here. So if I run this and you would see that I have got this output - you know 0.12 here.

Now you know run array 1d and now you compare the output number 3 8 7 and 3 8 2, now you can see where this value -0.12 is lying in this array 1d. So you can clearly see this is you know 6 value here. So definitely this is 50% quartile here you know half of the values are behind this value. So in this fashion you can see very easily because the sorted array the array 1d we had already you know sorted.

So on sorted array we had applied this, please note this that array 1d when we type this line of code 387 array 1d has already been sorted. So on this sorted array we have perform this kind of indexing here and access this particular value 50% quantiles value. So sorted array can actually can be useful to quickly compute the quantiles as well. Now let us move further, now we will talk about the certain operations set operations on let us start with 1d arrays.

(Refer Slide Time: 26:42)



```
regex1.sub("found", text1)

Out[240]: 'Ramesh found\nHukesh found\nRajib found\nSantosh found\n'

In [241]: print(regex1.sub("found", text1))

Ramesh found
Hukesh found
Rajib found
Santosh found

In [162]: # Example: find email addresses and segment each address into its three components:
# username, domain name, and domain suffix
# parentheses around the parts of the pattern to segment
pattern1 = r'([A-Z0-9_!@#$%^&*~]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})'

In [163]: # compile the regex
regex2 = re.compile(pattern1, flags=re.IGNORECASE)

In [164]: # using findall to produce a list of tuples having the segregated components
regex2.findall(text1)

Out[164]: [('ramesh', 'iitr.ac', 'in'),
('mukesh', 'gmail', 'com')]
```

So let us take this example terms here and let us have a look at the values, so in this we have this array we have used before also so alpha beta gamma alpha sigma. These are the values that are part of this 1d array. So what we will do is we will discuss few set operation that can be performed. So let us if we want to find out the unique values and we would like to sort them out.

So we will like to produce a sorted 1d arrays and also with the unique values. So for this we have this unique function, so we can use this np.unique and we will have to pass on this particular object terms and if I run this np.unique terms you can see in the output 390 you can see the alpha, then beta, gamma and sigma as you can see these string values this is string array this it has been sorted alphabetically.

And only the unique elements have been retained, you can see in 389 alpha was coming twice right, now that has been only it is coming only once. And the first one first element first appearance has been retained right, the second onwards appearances they have been removed and we are just left with the unique values and sorted array. Now a similar thing we can work with the numerical arrays as well.

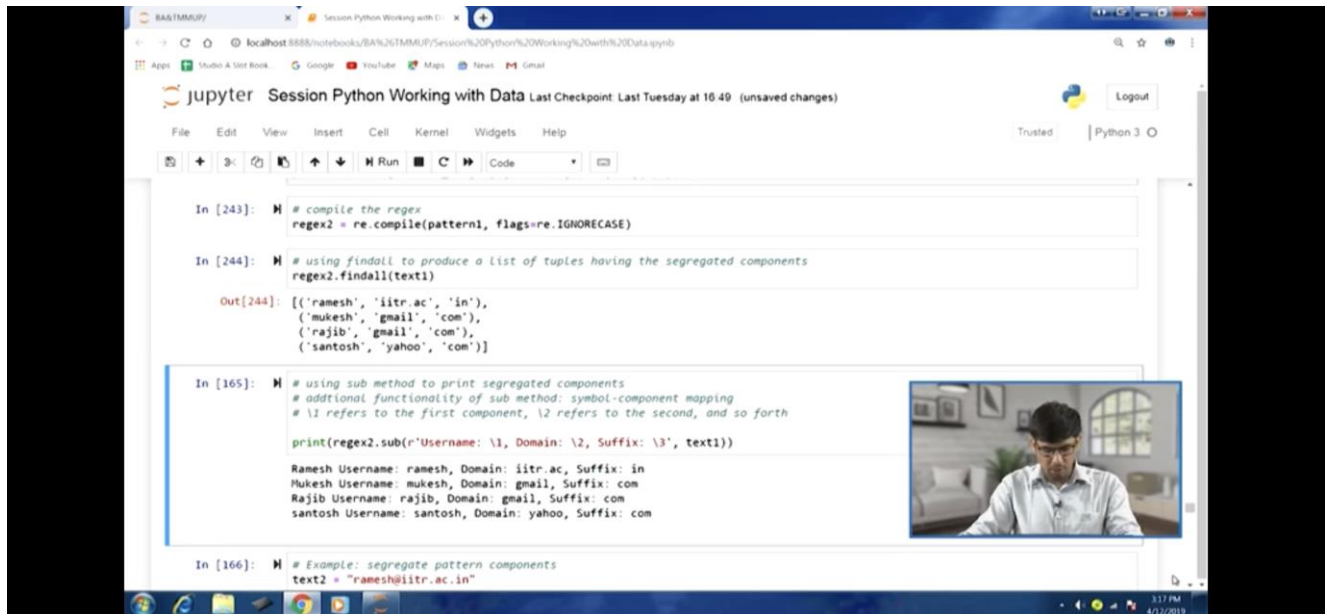
So let me create this array 1d 1. So let me run this and you can see in this array we have many values which are being repeated. So first element 5 then again 5 then 5 then 4 then 2 then 3 3, so you can see few elements are being repeated a few times. So again we can if you want to find out as you know sorted array with unique values we can use this np.unique function and we can will be able to run this.

So this kind of functionality is really useful in analytics context to give you an example for example you have a categorical variable and you know categorical variable will have certain categories certain labels. So all the observations that all the cases, all the observations that you might have in your tabular data and for that categorical variable. So those labels are going to be repeated for different observations.

So we just want to if you are if we do not have the metadata with us, if we do not have the

information about whether how many categories are present in a particular variable. So it sometimes it becomes quite difficult, so using this function `np.unique` function will be immediately able to run this function on that column on that vector and we will be able to quickly find out the unique and elements.

(Refer Slide Time: 29:30)

A screenshot of a Jupyter Notebook interface. The browser address bar shows a local host URL. The notebook title is "Session Python Working with Data". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for running, saving, and other actions. The code area contains three input cells. The first cell (In [243]) compiles a regex pattern. The second cell (In [244]) uses `findall` to extract components from a list of email addresses. The output (Out[244]) shows a list of tuples. The third cell (In [165]) uses the `sub` method to format the extracted components into a readable string. The output shows formatted strings for four email addresses. A small video inset in the bottom right corner shows a person speaking. The status bar at the bottom indicates the time is 3:17 PM on 4/12/2019.

```
In [243]: # compile the regex
regex2 = re.compile(pattern1, flags=re.IGNORECASE)

In [244]: # using findall to produce a list of tuples having the segregated components
regex2.findall(text1)

Out[244]: [('ramesh', 'iitr.ac', 'in'),
           ('mukesh', 'gmail', 'com'),
           ('rajib', 'gmail', 'com'),
           ('santosh', 'yahoo', 'com')]

In [165]: # using sub method to print segregated components
# additional functionality of sub method: symbol-component mapping
# \1 refers to the first component, \2 refers to the second, and so forth

print(regex2.sub(r'Username: \1, Domain: \2, Suffix: \3', text1))

Ramesh Username: ramesh, Domain: iitr.ac, Suffix: in
Mukesh Username: mukesh, Domain: gmail, Suffix: com
Rajib Username: rajib, Domain: gmail, Suffix: com
santosh Username: santosh, Domain: yahoo, Suffix: com

In [166]: # Example: segregate pattern components
text2 = "ramesh@iitr.ac.in"
```

And also in a sorted fashion. So sometimes it is going to be really useful in the analytics context, that is again will in the next example what we will do is we will use sorted we will use sorted and set function, these 2 function to return sorted 1d arrays with unique values. So already we have talked about `np.unique`. Now these are a few other you know functions that can be used to perform the same thing.

So let us again have a look at the terms array here and you can see the output alpha beta gamma alpha sigma and I will use set terms or set terms will also produce the similar kind of output and a sorted array with unique values. So if I run set terms here you can see we get these you know same output here in this you know as you can see here. Now similarly sorted function also we can use here, so set is giving us the unique output.

Then sorted we can combine the set and sorted function and sorted function can be used to sort it out. So in this case it was already sorted in a sense, so however we can run this method when this

function sorted and within parentheses set terms this will have the unique terms and that those will be sorted. So let me run this, so you can see in the output 395 we have got the sorted. So first we will use set and then sorted and we will be able to achieve the same output that we got using the np.unique function.

Now sometimes in certain scenarios we might be required to check the presence or absence of elements of one array and other. So a matching kind of thing and other presents are absence kind of thing we would be required to find out. So that also let us see how we can do this , so let us take this is example array1 d1 here and if I run this we get this array here and you can see some values are repeated here. Now the function that can be used to perform this is np.in1d.

So this function can actually be used to perform this in the first argument what we are doing is we are passing array1d 1 and the you know second argument we have this list of values 0, 1, 2 3. So we would like to find out whether any element in the first argument that is the you know first array whether any element in that first you know first argument first array whether it is matching with the elements of second list that is it is actually a list here.

So we will get a boolean output if it is matching then true if it is not matching and false. So the output would be of the same size as the first argument array you know passed in the first argument. So if I done this you will get in the output 397 you can see first value in the output 396 is 5, which is not part of the second argument list. So false similarly for 5 again false false, 4 is also not present false then true.

So that is actually the value is 2 which is part of the you know second argument list. So therefore it comes out to be true. So presence or absence of elements of 1 array or list and another can actually be done quite easily using this particular function. Now next aspect about you know arrays that you would like to discuss is about file management. So file management we have before also how this can be for home using arrays that is the next thing that I would like to discuss. So this will discuss in the next lecture.

(Video Ends: 31:21)

At this point I would like to stop here, thank you.

Keywords: Numerical Python, Vectorization, Multi-Dimensional array, Categorical variable, integer, float, string etc.