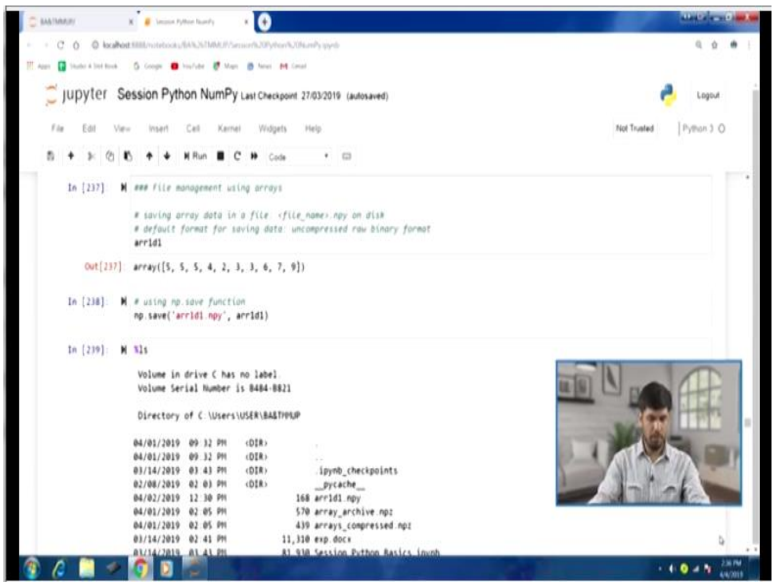
Business Analytics and Text Mining Modeling Using python
Prof. Gaurav Dixit
Department of Management Studies
Indian Institute of Technology Roorkee

Lecture-24
Numerical python-Part I

Welcome to the course business analytics and text mining modeling using python. So, in last few lectures we have been discussing norm NumPy package. So, let us start our discussion from the point where we start in the previous lecture. So, in this particular lecture we are going to start our discussion on file management using arrays. So, little bit on file management we have discussed before as well.

(Refer Slide Time: 00:51)



Now it would be a mainly in the context of arrays, so let us start our discussion here file management using arrays. Now one of the few things that we are required to do is you know typically in file management is saving data in a file. So, in this case we will deal with saving array data in a file and let us see how we can do it. So, we will take an example here this array 1d1 and this we have created before as well.

So, default format that is typically used for saving data is uncompressed raw binary format, so that is the default format that is typically used, now let us take this array 1d1.

(Video Starts: 01:36)

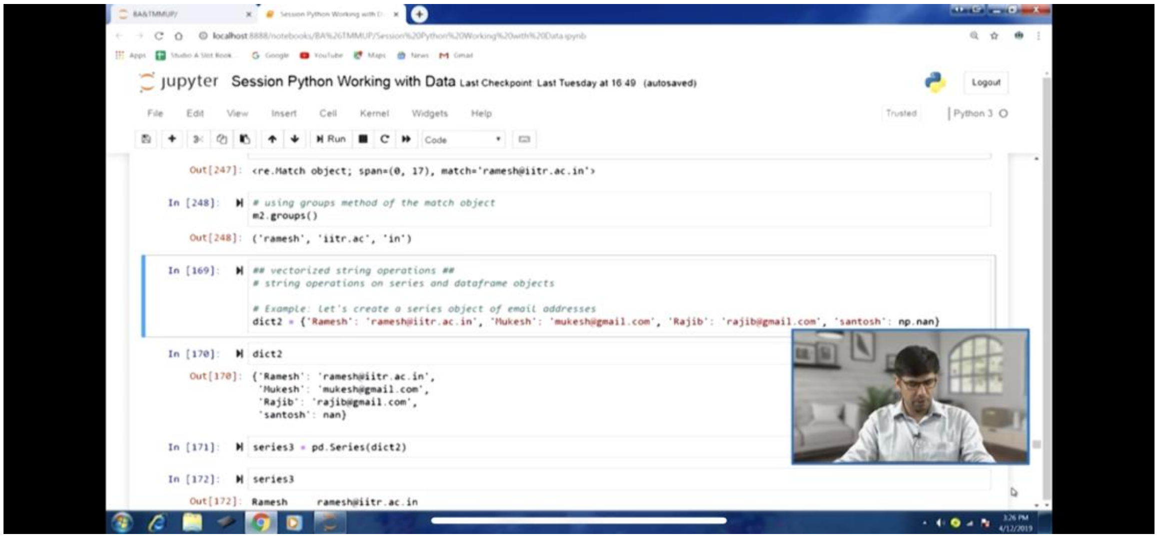
So, let us see what is the data there, so if I run this right now this is not defined, so I will have to do a few things here like we did in the previous lecture as well, I will clear all the outputs here first, all the output that is there in the file and then I will run all the lines of code here once again. So, once everything has been cleared I will go back and run all and once all the lines of codes have been you know run we would have you know all the objects available for us for further execution.

So, let us go through this I think somewhere like in the last lecture we discuss, somewhere the output will stop. So, I will pick that again and what we will do, we will go to file management again and what we will do we will run all the lines above the line that we are going to start in this lecture. So, let me go back to that part, so I will like to run all cells above this line, so I will say run all above, so let us see whether our problem of defining previous array objects is sorted out.

So, let it run and if it is able to successfully run without any errors in between anywhere then we will have the output. So, that is the scroll back, it has stopped at some point here this one again the same point here. So, what I will do, I will run this and once this is done then I will again go back to file management. Now because that problem that we are facing where this we are not able to run all cells above, so because of now this 1 extra line that I have executed.

Now I think we would be able to achieve that where we will have the output of all the previous lines of code and new lines for the remaining lines that we want to cover in this lecture, for those lines we would be executing in this lecture it iself. So, now again I will try that run all and from the point where the output stop, I will have to no choice because of this 1 demonstration that produced error, I will have to I think run from here it iself, run all below no choice.

(Refer Slide Time: 03:05)



Otherwise we will run into one more problem again here, so let me run this and then once this is done we will move on to the file management part, that we wanted to discuss in this lecture in the first place. So, first thing that we wanted to do is about saving data in a file, so in this you know array context we will like to say vary data and as I said default format for saving data is uncompressed raw binary format.

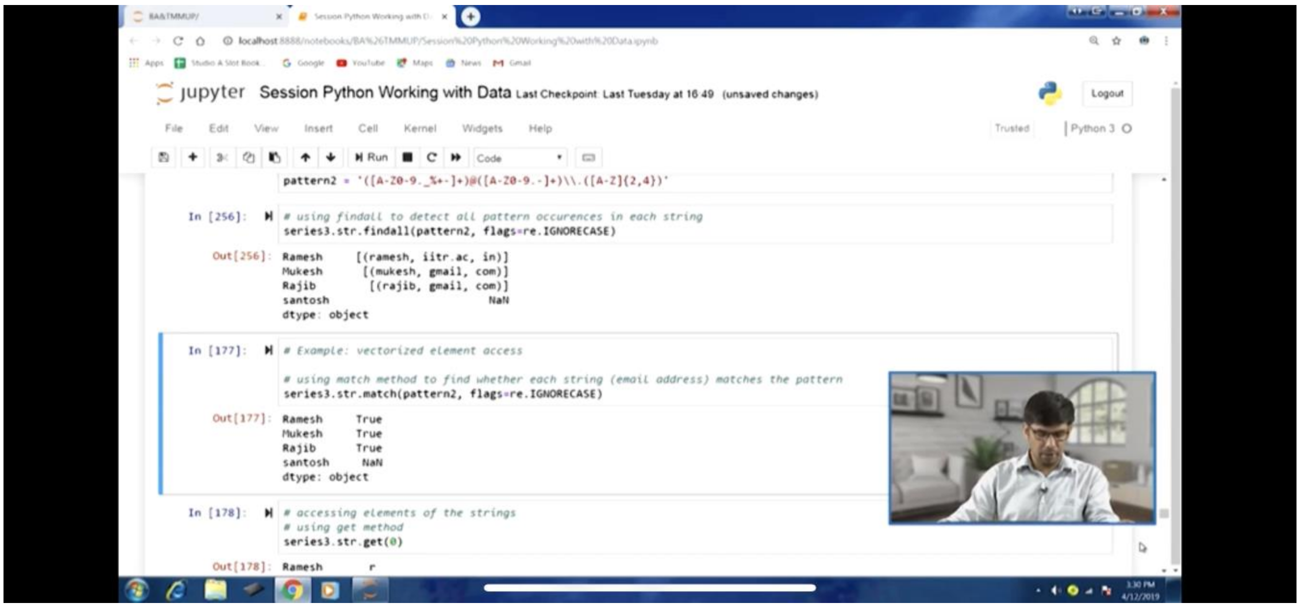
So, let me run this array 1d1 and these are the elements values in this array, this is 1 dimensional array. And if I want to save this file, so I can use this np.save function, so you can see in the parenthesis np.save and within the parenthesis first argument is the name of the file. So, here you can see I have given the name of the file in the using the name of array it iself. So, we are going to save this array 1d1 in this file and the name of the file is also array 1d1.npy.

So, in this case because we are using you know numerical python module this package and the extension is also similar to that .npy and we would be storing, we would be saving this data array 1d1 in this file. So, if I run this then a file with that name would be created and if I run this magic combined %ls, then in the working directory I would be able to locate this file.

So, you can clearly see that this array 1d1 clearly you can see here in this list of files here in this directory, current directory, that particular file has been created. Now if I want to load any array file from disk for that I can use np.load function. So, in to use this function again within the parenthesis we will happily specify the name of the file, so in this case array 1d1.npy, so I can specify this and then run this, so let me run this.

And you can see in the output the data you know once we load it the array that we have stored there, saved there in that file that has been you know displayed in the output. So, this is how we can save and load you know save the data in a particular file and also load the data from that file. So, these are 2 important function np.save and np.load in terms of saving and you know loading data in files.

(Refer Slide Time: 07:27)



So, in files and from files, loading data from files, so all these are important file management functions. Now let us move on, now if you want to save multiple arrays, in the previous example we just saved one array. So, sometimes we might be required in certain scenarios to save multiple arrays, so all those you know data might be important for us for future processing, so we might have that.

So, saving multiple arrays in an un you know compressed archive, so in this case the extension and file name can be archive_name and the extension will change .npz. So, using this extension we can save multiple arrays in a you know uncompressed archive format here. So, for this we can use this np.savez function and we can pass arrays using keyword arguments right.

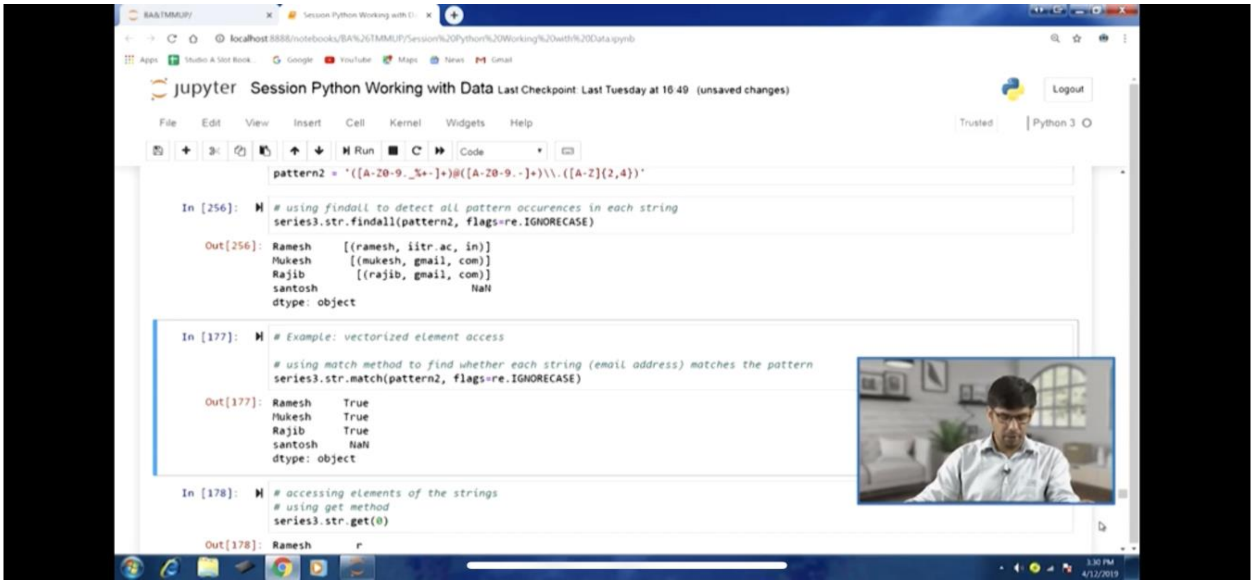
Because later on if we want to retrieve we want to load those arrays then we can use those keyword arguments to actually you know access those elements, access those objects. So, you can see in this example I have this function np.savez and then first argument is the name of the file where we want to archive, where we have to save these multiple arrays, then the listing of the arrays it iself using keyword arguments here.

So, first a=array1d and then array 1d1, so if I want to save these 2 arrays, so I can call this function. So, once this is done I can run this magic come command %ls and you can always check whether the file was created. So, you can see array_archive.npz, this file has been created, now if we want to **s** load the array archive from disk. For this again we can use the np.load function and what we will get in return is a dict like object, that is going to be written.

So, you will understand that, that is why we require keyword arguments there to make this happen. So, if I run this you know np.load and if I pass on the name of the file array_archive.npz will store the return value in arch in the arch you know object. So, let me run this and once this is done, we had stored 2 arrays there using 2 keyword arguments a and b.

So, now I can use these arguments like a you know dict kind of processing, so here you can see arch and within brackets I have you know within single quotes b, that is to access the array associated with this particular keyword arguments, a kind of mapping is created. So, that is why we said a dict like object is return, so if I run this I will get the output for that array.

(Refer Slide Time: 11:06)



Similarly for the other one arch a, for this also you can see output is there and that associated array has been presented in that output. So, in this fashion we can use np.save function, np.load function to store 1 array or more than 1 arrray in different file formats as you have seen till now. Now sometimes you would prefer to save arrays in a compressed archive, so for that we have this function np.savez_compressed, so this function can also be used.

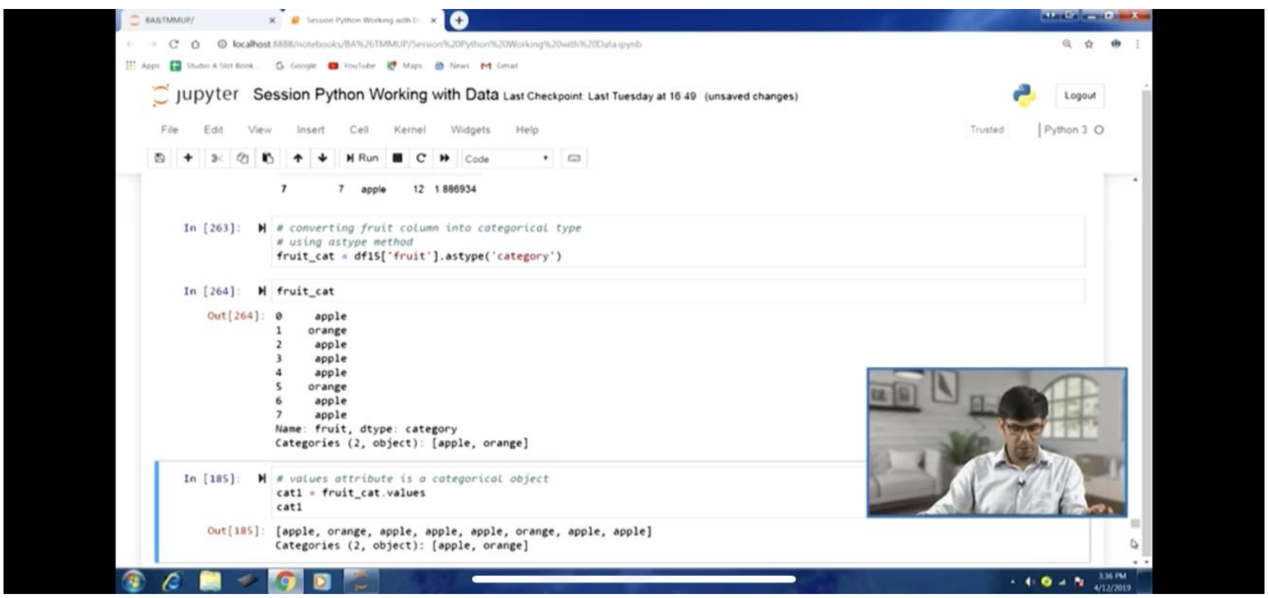
So, this if we want to effectively utilize and if we are you know if you want to efficiently utilize the storage space that we might have. So, for that purpose it might be useful, so when we can call this function np.savez_compressed and within the parentheses the first argument is going to be the name of this file arrays_compress extensions temp.npz and then the air arrays. So, first one a array 1d and then b has array 1d1, so using this work particular function you would be able to save the air arrays in a compressed archive.

Then we can look at the you know we can use the you know magic command %ls and we can find out whether the file was created. So, you can see an arrays_compressed file.npz has been created in this listing. So, this was about these 2 important function npsave and npload functions and different scenarios and how they can be used. Let us move on, now let us talk about the linear algebra operations and how the vectorization aspect of arrays can really be helpful in performing these algebraic operations.

So, let us take this example of element wise multiplication of 2d arrays, it is you know very similar to what you know we have discussed in the matrix multiplication. Now that was in the context of element wise operation, the vectorization aspect. Now in the linear algebraic context we are trying to discuss few things. So few new things would also be discussed here, so if I take array 1d2 this new array as an example here.

So, let me run this array 1d2, so you can see this is a 2 dimensional array 2d array because we want to demonstrate this matrix multiplication here. Now if we look at the shape of this array clearly you can see 2*3. So, if I use the shape attribute array 1d2.shape you can see 2, 3, 2dimensional, first dimension with 2 elements, second dimension with 3 elements because there are 2 list and within each list we have 3 you know numerical values.

(Refer Slide Time: 13:49)



Now let us take another array 1d3, now this array is you know slightly different in terms of dimensions you can see here. So, if I run and the output you can see that this is also a 2dimensional array with slightly different shape, let me run this shape attribute here. So, array 1d3.save and in the output you can clearly see 3, 2, so it is 3*2 array and first dimension with 3 elements because there are 3 list.

And then second dimension is having 2 element because within each of those list we have 2 numeric values. So, in this fashion we can use these 2 arrays later on for matrix multiplication. Now in this time when we are demonstrating matrix multiplication we will use a different approach, we will use a .method and that we had not discussed before. So, this using this .matrix method also we can achieve the same thing.

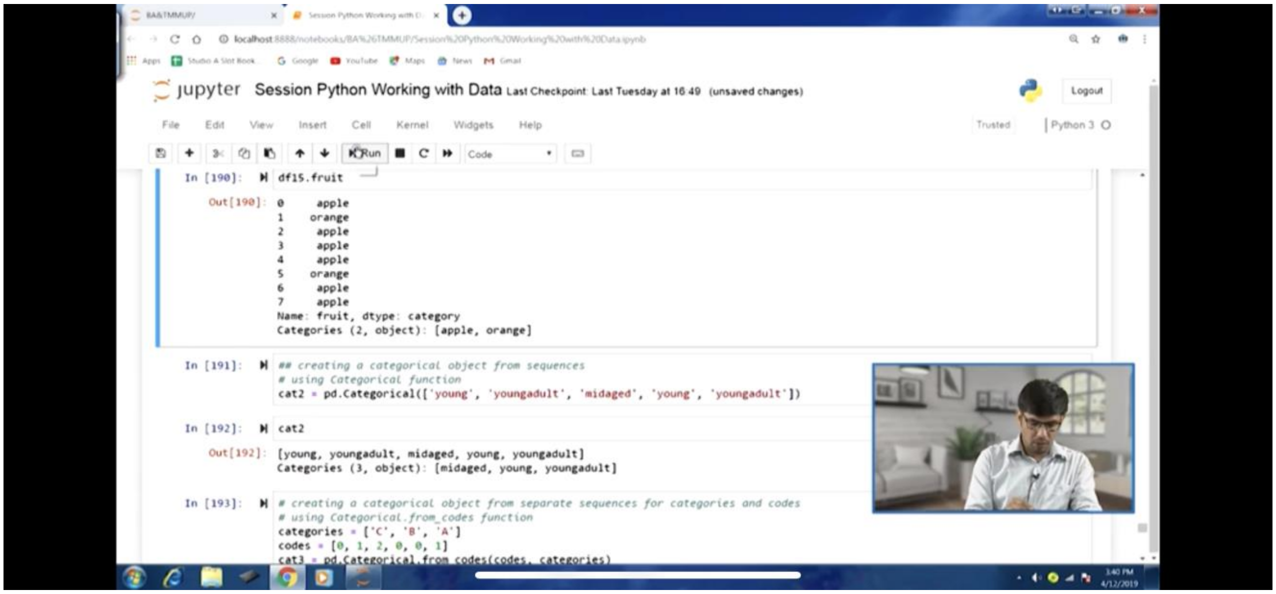
However though the you know terminology looks quite similar to what we typically discuss in the matrix algebra. But remember this is based on the vectorization concept based on the element wise operation concept that is facilitated by you know arrays here in this particular package, NumPy package. So, array 1d2 and the method.dot dot and then we can pass on the second you know 2d array here and this will facilitate a matrix multiplication and you can see the output here.

So, the first one was 2*3 then we had 3*2 the second one, so the resultant is you know 2*true and you can always confirm the values here. Now we also have another this was .method, so this is for you know inplace replacement we want to perform matrix multiplication and you know we want to do it inplace replacement there in the error it iself. Then for that methods are useful otherwise we can always have the functions, so in this case we have this np.function.

So, here we can pass on 2 arguments, first array array1d2, second one array1d3 and we will get a return it will return a another array with the matrix multiplication output. So, if I run this you can see that we have got the same answer, so 2 approaches .method or np.uh dot function. Now, let us take another example, so here you can see in the np.function we are passing first argument array 1d2 and the second argument is actually np.ones.

So, here np.ones this is another way to create an array like we have discussed before just with the 3 elements and right, so 1 dimensional. So, first one you know and this one so 2*3 and 3*1, so in that sense will get you know the output here, so you can see here in the outputs. So, if I run this np. you can see the output here you can get the 1 dimensional arrays also as an output here you know while doing matrix multiplication.

(Refer Slide Time: 21:56)



Now another way to perform matrix multiplication is that we can use this you know ampersand operator here. So, that can also help us in achieving the same thing, so here also array 1d2 at np.ones and 3, so we are creating another array and then multi matrix multiplication here using this ampersand operator here. So, if I run this again you can see the same output and you can compare 4, 3, 7 and 4, 3, 8 same output we have been able to get, so different ways to perform matrix multiplication.

This sparkle section is mainly for the to demonstrate few of the linear algebraic operation. So, now what we will do, we will use some of the linear algebra libraries just to introduce you to some of the functions, some of the methods that are available there. And how they can be used and along with the this you know numerical python package and arrays that we are discussing.

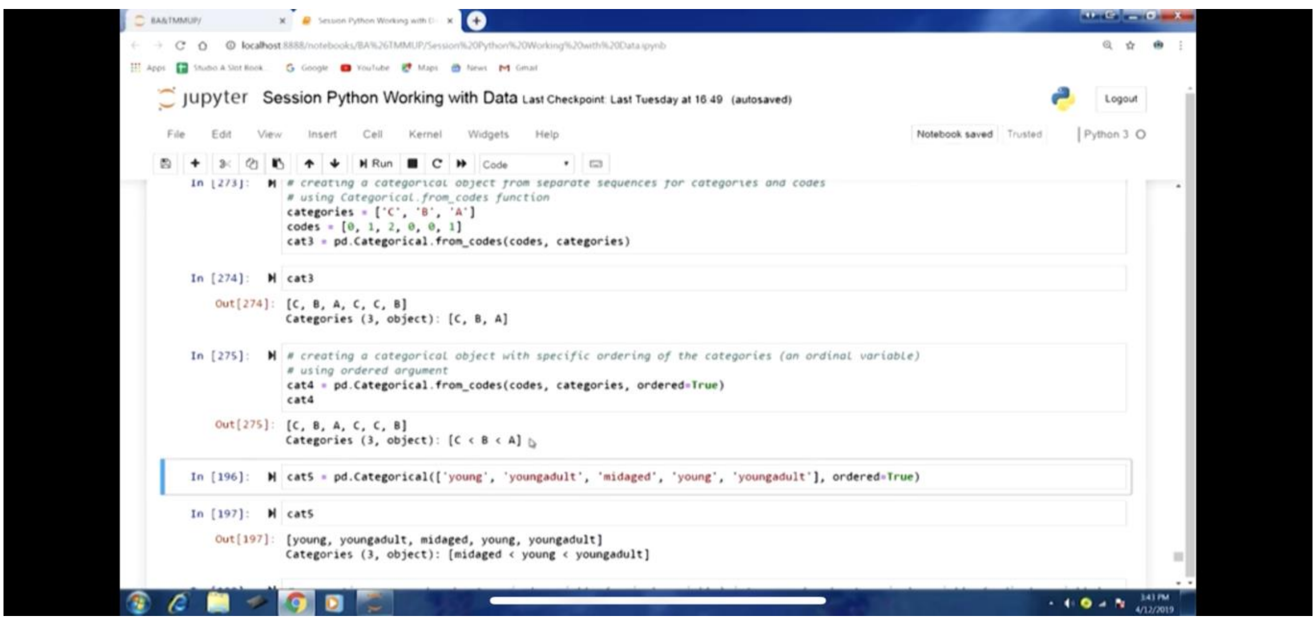
So, for this we need to import this you know NumPy and then linear .lin alg linear algebra. So, from this module we will like to import these 2 at inv and qr these 2 libraries, so we are taking this matrix which is a 2d array here and we are initializing this matrix mat here. So, np.random.random is the function that we are using that we will discuss later on in more detail to initialize this matrix or 2d array, so these are the values as you can see.

Now if you want to compute transpose, so we have introduced this before also .T attribute, so this can really be useful to perform transpose. So, mat.T if I run this you can see that we have got the transpose of a matrix in this case mat. We can also perform matrix multiplication again here in this fashion another demonstration So, you can see that we are performing transpose and then using the .method here, so mat1 you can see we are recording the return value here also in matrix manipulation result as well.

So, this is our mat1, if we want to perform inverse of a matrix, so because we have imported library, so we can use this function. This function is available to us now inv and we can pass on the first argument mat1 here and we will get the inverse. So, if I run this inv mat1 and I will get the output the inverse of a matrix, similarly we can use you know we can pass this inverse of matrix as an argument for the .method and we will get the matrix multiplication there also.

So, matrix multiplication of matrix with it is inverse, so that can be performed in one go. So, you can see all these you know vectorization the element wise as you know operations that we can perform with the arrays. It allows us to execute you know these linear algebraic operation in one go. So, that is the purpose of this demonstration, so you can see here ma1.dot and multiplication of matrix with it is inverse.

(Refer Slide Time: 24:19)



So, you can see the output here, now similarly another library that we had imported qr library, so we can perform the qr decomposition which is another linear algebraic operation. So, for that we have this function qr and we can pass on this matrix mat1 here and in this case we will have we can also segregate these components q and r components here. So, we can record appropriately here, so if I run this q, r and on the right hand side qr function we are passing this matrix.

And we will be able to perform qr decomposition one go and at have a look at the value of q and r here. So, here you can see how easy it is to perform some of the using these you know libraries that are available libraries and functions that are available to us, how easy it is to perform these linear algebraic operations here. Now let us come to the another important concept related to this here is that random number generation.

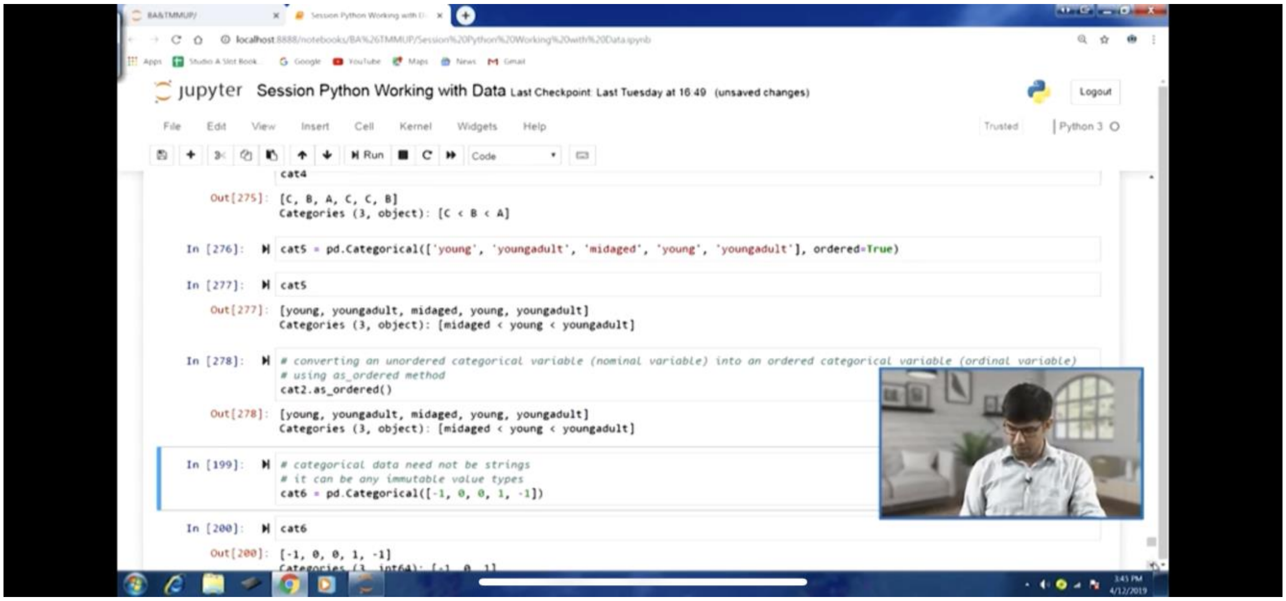
So, in previous many lectures we have been using random.random function quite a lot without discussing much on it. Now in this particular section we would be able to spend some time on it, so what we call it is pseudo random number generation. Because in computer science as of now the truly random number generation that is researchers they have not been able to develop a truly random number generation.

And most of the random generation they what we typically use, they are referred as pseudo random number generation. So, typically this generation random number generation is this generation is performed by an algorithm bit deterministic behavior based on the seed of the random number generator. So, we feed a particular value per particular seed and that seed is use to perform this generation using a particular algorithm.

So, a NumPy.random module actually has a number of functions and to you know perform this kind of a random number generation. So, we will discuss a few of these functions here in this particular section. So, for example we can generate samples from the normal distribution randomly, so for this we have this normal function available in this particular module NumPy.random module.

So, by default it will this normal function by default it will perform standard normal distribution. Because some of the arguments they have the default value for mean and standard deviation which makes it perform standard normal distribution by default. Otherwise we can always specify values for these arguments mean and standard deviation and perform the regular you know normal distribution.

(Refer Slide Time: 24:19)



So, let us take an example of 4*4 array and we are going to create this array and all the values of you know all the elements of this particular array this 2dimensional array are going to be filled with this particular function. So, that is the main idea, so we are going to generate these random numbers and at the same time we are going to create a 2d array and fill this 2d array with these randomly generated numbers.

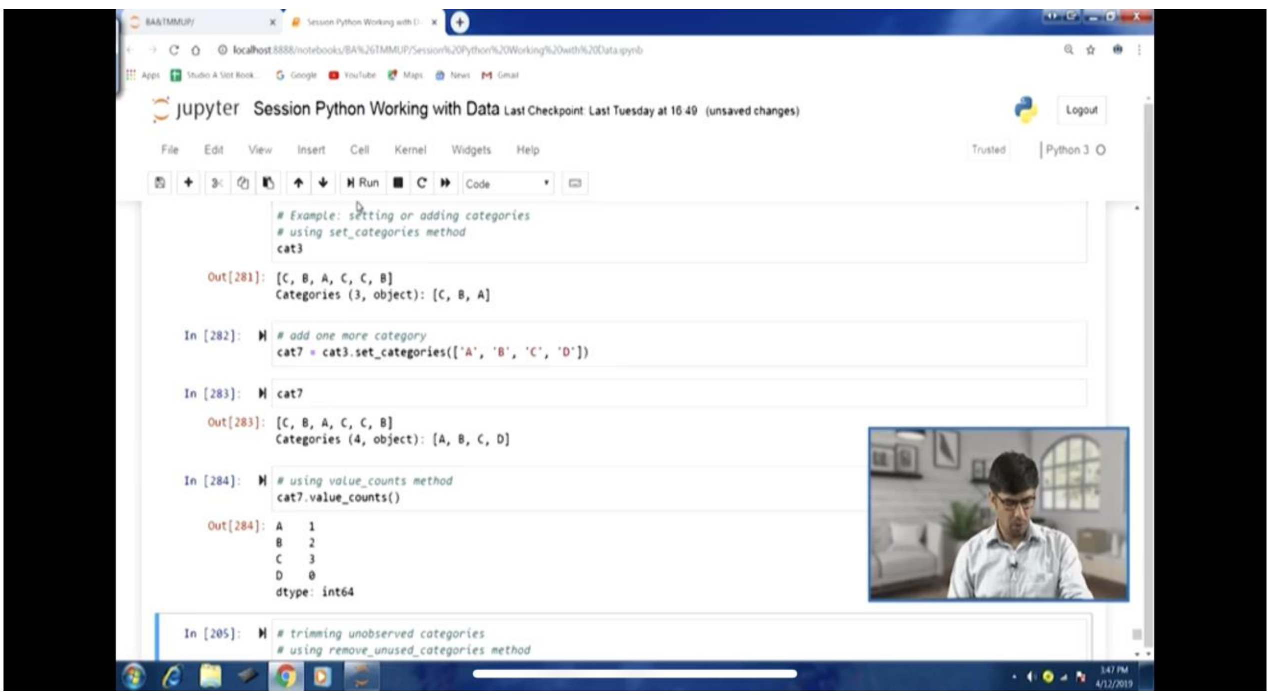
So, np.random.normal, this is the function and we are passing the size argument here and we are specifying the size of 2 dimensional array here 4,4. So, first dimension with 4 elements, second dimension with 4 element, so if I run this. Let us have a look at the this output samp, so this is actually a 2d array and you can see there are 4 list and each of the list is having 4 elements. So, 4*4, so using normal function all these values that you can see they are actually conforming to the standard normal distribution.

Because we did not specify mean and standard deviation, so all these values are conforming to that and that is the indicated by the values itself. If you have certain familiarity with the normal distribution, so let us move on. Apart from using the you know NumPy random you know module we have so you know pythons built-in random module as well. So, however it has certain limitation in the sense that samples that we have they produce only one value at a time.

So, we can import this from random module that is the pythons built-in model, from random import normal variate, this is the function that we would be using. So, let us generate a list of 1000 values, so you can see that you can look at the code that we have written here. We are calling on the left hand side samp1=on the right hand side within brackets. So, you can see we are using a list comprehension here, so we have calling this normal variate 0, 1.

And we are running this loop for you know 1000 times right, so you can see if I run this code we will get 1000 values in the samp1 output. So, if I run this again, so you can see these values here in the output number 450, so 1000 values have been produced here. So, in the list comprehension what is happening is when we call normal variate 0, 1 only one value is being sampled at a time and since we are running it you know in a loop.

(Refer Slide Time: 27:30)



So, 1000 values we are able to generate, so now you can compare that how easy it is to generate random values using NumPy random module right, so here we have to rely on a loop functionality. Now the same thing if you want to perform using you know NumPy's random module, then we can call this np.random.normal function and we can pass on the size argument and just you know 1 value 1000.

So, again in this case we will get an 1 dimensional array of 1000 values, so there we got a list of 1000 values we used a list comprehension when we used python built-in random module. Now when we are using NumPy's random module we are going to get 1d array and let us run this. So, it will very quickly run because this is many times faster than the built-in capabilities of python, so you can see.

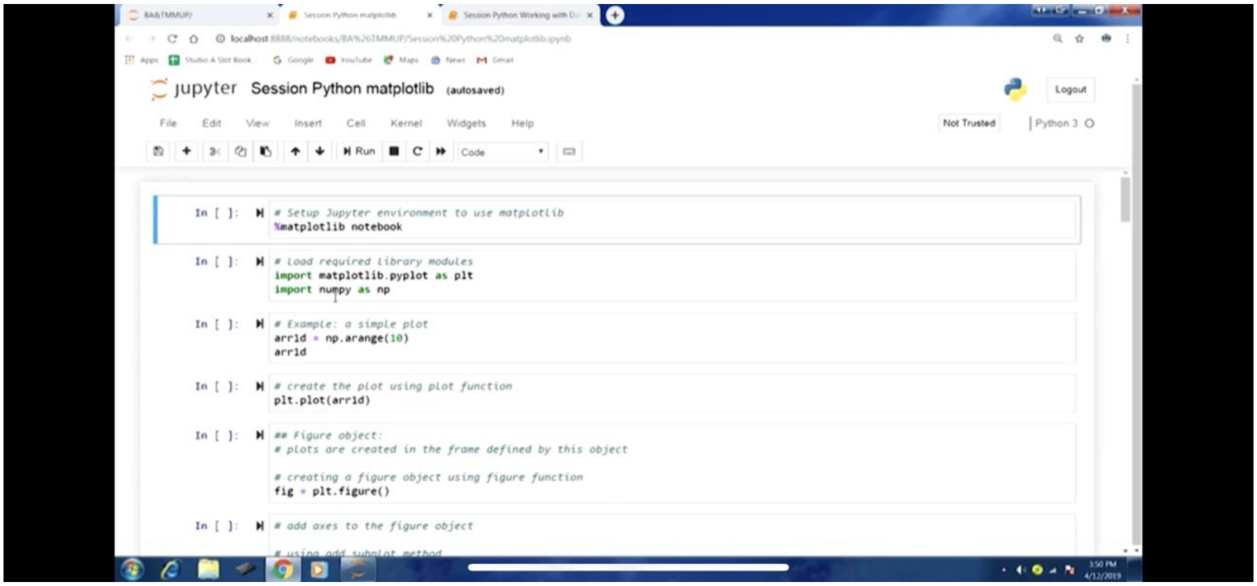
In the outfit number 452, we have actually got and 1d array of consisting of 1000 values, so this is how we can compare the different ways of producing the same thing and how the processing is so much simpler using NumPy. Now sometimes we might be require to change the random number generation seed. Because as we discuss these pseudo random number generation the approach that is taken.

It is based on the seed that is the algorithm is use to generate these numbers. So, we can always change the seed and of course it will have it is own effect on the numbers that are being generated randomly. So, for this we have the seed function, so np.random.seed and we can pass on the seed values which we want to use for this you know random number generation.

So, let us initialize a new seed here and we pd.random.seed and then we can call this function np.random.random this is the function that we had used before. So, this is the function which is equivalent of a standard normal distribution, so we earlier use normal function. Now we are using random function which is directly you know giving us results as per the standard normal distributions.

So, if I run this we will get as you can see in the output over 454, 10values are there. And that are you know conforming to this standard normal distribution and these values have been produced in a 1d array. Now there is another aspect related to this random module is that, the state of the seed and how it is being generated.

(Refer Slide Time: 31:52)

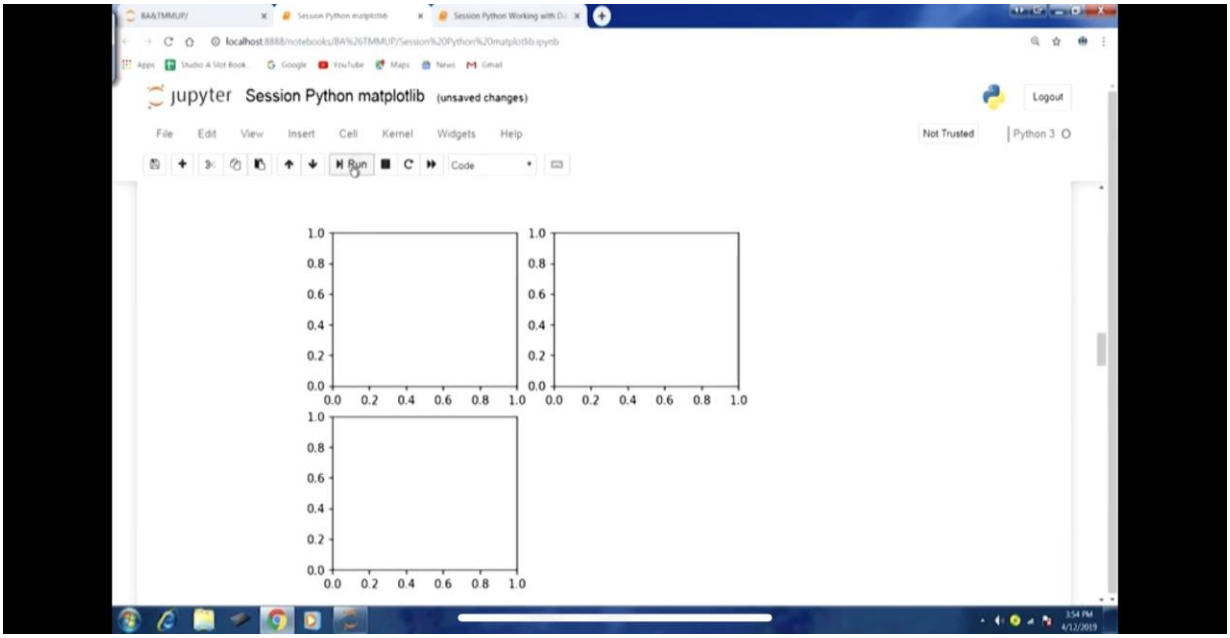


So, certain state so this is state is reflected in certain output form, so we can always call this function np.random.get_state. So, this particular function will give us the details about this state, so we would not go into much details about this state. However why we have called this function, why we are demonstrate this function will you know talk about that particular point. So, NumPy.random function that we have discussed, they typically use the global state and you know if we want to.

So, many idea is that many tasks that we perform in analytics you know sometimes we might be using candidate models. So, if we are able to use the same seed then we will be able to you know draw the same you know points, same in indices using the you know random number generation. So, therefore it would be helpful for us to compare the performance, compare the results of those candidate models.

Because same observations would be use to build the model and therefore and also same observation would be use to test the models and we can compare these candidate models. So, the seed can be really useful however certain scenarios we require to use a generator which is isolated from this global you know state kind of thing, global state mechanism that we have in this module. So, for that we have another function, so we will use that to create a **a** seed which is isolated create and a state which is isolated from the global state.

(Refer Slide Time: 32:00)



(Video Ends: 30:23)

So, we will stop here and we will discuss this aspect in the next lecture, thank you.

Keywords: Linear Algebra, Random function, Demonstration, Flow charts, classifiers etc.