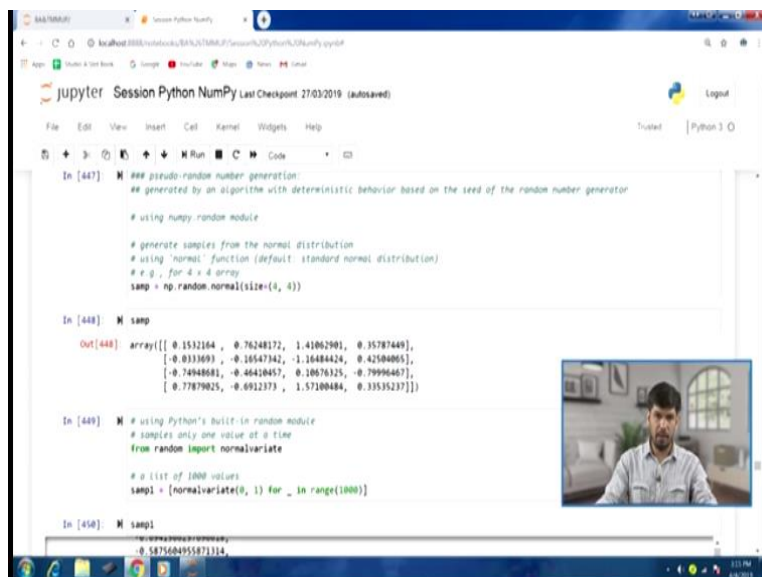


Business Analytics and Text Mining Modeling Using python
Prof. Gaurav Dixit
Department of Management Studies
Indian Institute of Technology Roorkee

Lecture-25
Numerical python-Part VII

Welcome to the course business analytics and text mining modeling using python. So, in previous few lectures we have been discussing numerical you know python package, NumPy package. And in the previous lecture we started our discussion on pseudo random number generation. So, let us discuss few points that we were able to discuss in the previous lecture and then we will pick up from there, so we talked about pseudo random number generation.

(Refer Slide Time: 00:52)



```
## pseudo-random number generation
# generated by an algorithm with deterministic behavior based on the seed of the random number generator

# using numpy.random module

# generate samples from the normal distribution
# using 'normal' function (default standard normal distribution)
# e.g., for 4 x 4 array
samp = np.random.normal(size=(4, 4))

Out[448]: array([[ 0.151104,  0.76248172,  1.41801901,  0.35787449],
 [ -0.833609,  -0.16547342, -1.16484424,  0.42504865],
 [ -0.74948681, -0.46420457,  0.18676325, -0.79996467],
 [ 0.77879025, -0.6912373,  1.57100484,  0.33535237]])

# using Python's built-in random module
# samples only one value of 0/1
from random import normalvariate

# a list of 1000 values
samp1 = [normalvariate(0, 1) for _ in range(1000)]

Out[449]: 0.8375604955477114
```

And we said that you know these numbers they are generated by an algorithm with deterministic behavior based on the seed of the random number generator. For this we used NumPy.random module which we have been using of the functions from this module, we have been using in the previous lectures as well, for example random.random function. So, these modules you know can help us for example in generating samples from the normal distribution.

So, for this we can use the normal function, so NumPy.random.normal function, so default values as we discussed it can be used to regenerate a standard normal distribution as well. So, we talked about we did few exercises here use np.random.normal to you know generate these

numbers and create this 2 dimensional array.

(Video Starts: 01:50)

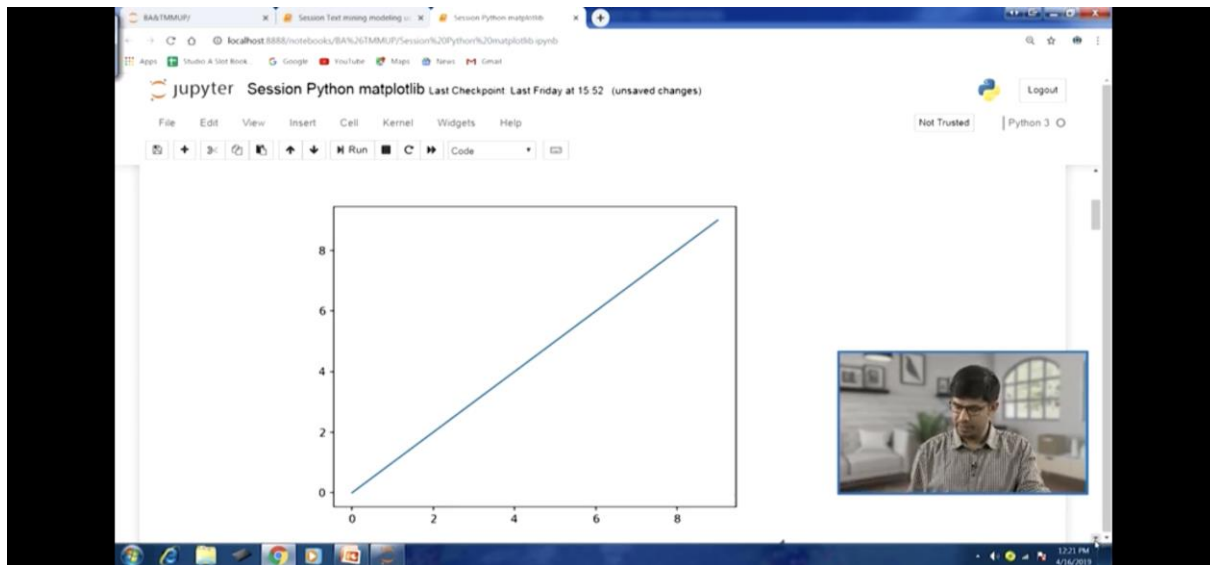
Then we talked about the python's built-in random module also and the limitation of this module also, how we will have to use a list comprehension which involves a loop construct to actually achieve you know something like a 1 dimensional array. And how easily that can be done using `np.random.normal` function in one go. So, we looked at those examples as well then we talked about the changing random number generation seed.

And because this kind of functionality is really useful in the analytics context as we said. Because sometimes for different candidate models we would like to have the same observation in different partition and compare the results. So, seed value will really facilitate you know the random selection of the observation into different partition. Then we talked about the random function which is actually confirm to the standard normal distribution and we looked at that also.

Then we talked about the state aspect of these particular you know random number generation that there is something called global state which is typically use to generate these random numbers. However if situation demands that we would like to we would not like to use the global state for the random number you know for these random functions. And rather we would like to manage or alter the state manually and therefore we would like to have our random number generator isolated from the global state.

So, that can also be done, so for this we can use random state function, so to create a random number generator `rng` and use it for further cause we can use this `np.random.random state` function. So, you can see in this upper particular line of code **we** on the left hand side we have `rng`, a new random number generator and isolated random number generator isolated from the global state that we want to create.

(Refer Slide Time: 02:54)



And for this we are using `np.random.random` state function, we are passing on this value 1, 2, 3, 4, 5 and if I run this we will have an isolated you know a state. And if I perform the you know same kind of operation where I am trying to do out 10 values confirming to standard normal distribution using the random function. Now you see specifically focus on the code line that have written here `rng.random`, earlier it was `np.random.random` now it is `rng.random`.

Because we have created a you know isolated state for this generator, so therefore every time we want to use this you know isolated state will have to use this particular this particular object `rng`. So, `rng.random` and then 10, so if I run this we will get the 10 values, so the only reference is this particular state is isolated from the global state. Now we can also look at the state, so earlier we had used `get_state` function, so here also we can call `rng.get_state`.

So, this output will reflect the state of this particular random number generator, the earlier you know previously when we had called you know `np.random.get_state`, that was you know the output was reflecting the global state situation. So, if I run this we will have these values which indicate the state of this particular generator but we would not go into details of this rather we will focus on another aspect here.

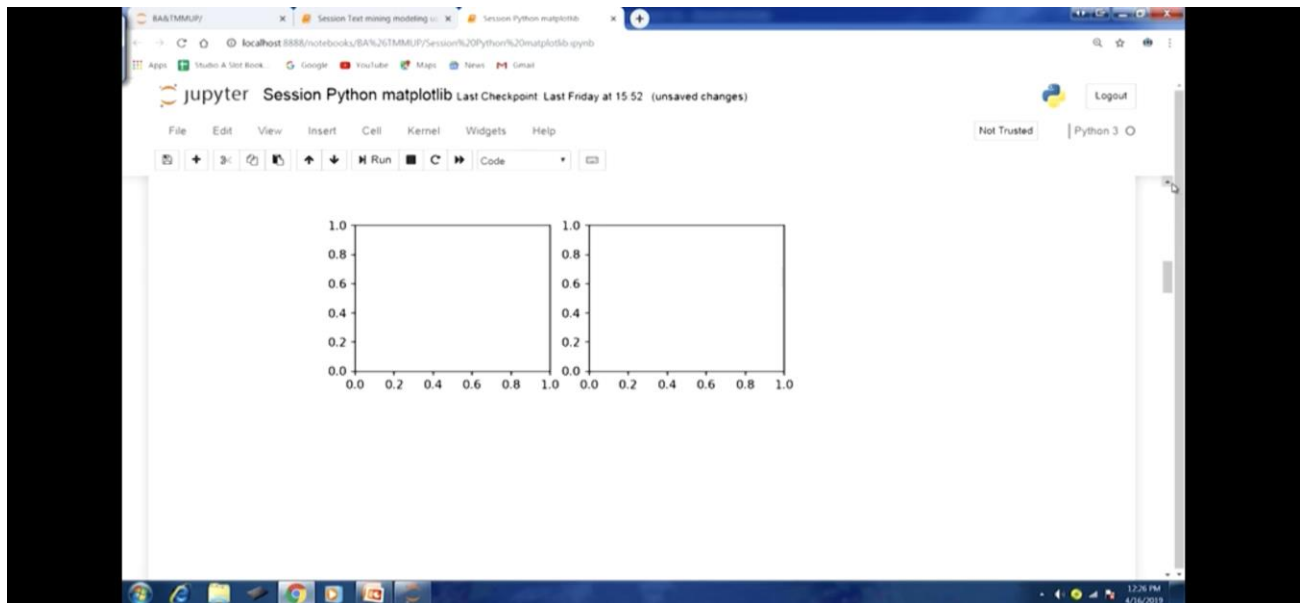
Rather it is an example which is you know sometimes really useful in the context of analytics, random walks. So, you know this is an interesting concept which is you know used in you know

modeling. For example many people believe that you know fluctuation in stock market prices they are actually random walk. So, essentially when we say random walk that means the next to values are going to be only dependent on the previous value.

So, therefore there is not much sense in making prediction of any series which is a random walk. So, here we are talking about when we are talking about stock prices we are talking about the time series, data and if that is you know a random walk. Then maybe it would not help us much in terms of making prediction about those prices. However how we can create random walks you know that we are going to focus on now.

So, in this context we can define a simple random walk like starting at 0 where the steps 1 and -1 occurring with equal probability. So, this is also a simplistic way of defining a random walk or we can call it a simple random walk. So, how we can you know create this simple random walk using python built-in capabilities, so let us focus on this. So, we have a random module and their also python built-in capabilities, so we will import this import random.

(Refer Slide Time: 10:22)



And then we have you know x, this is something that we will use for simple random walk. So, walk= we have creating a list we are initializing a list with these x values which is right now just with the 1 element 0. So, we will be working on the value of x and we will be appending this list

you know this walk list here. So, number of steps 1000, we will take 1000 steps and therefore 1000 values will be appended to the this list object that is walk and for that we will be working on x.

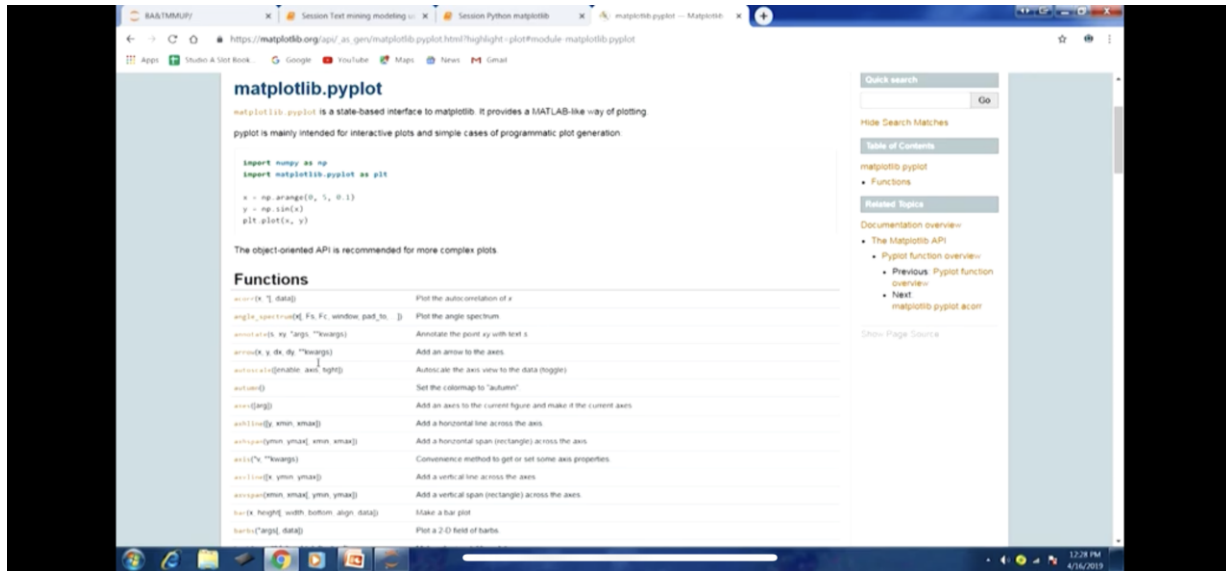
So, we are running a for loop here for i in this range, so 1000 values will run this and then step 1 step=1 if random.read. So, this is the function that we are calling here random.rand int, so between 0 and 1 else -1. So, we will pick a value from here using this function and based on that you know will either give step as value 1 or -1. Then will be you know using that value to you know update x and then add that x is going to be appended in the list that walk object that we have, so in this fashion we will be creating this.

So, let me run this and you can see if I look at the walk, so you can see the values 0, 1, 2, 3 these are the values of walk here. And if I want to generate a plot here, then this aspect we have used before also though we are yet to discuss the visualization the plotting aspect in more details later on, we will cover it later on. So, right now we can use this magic command %matplotlib inline and then import this local library module matplotlib.pyplot as plt.

And then we will have access to this function plt.plot and we can pass on and plot these 100 values from walk. So, if I run this, so you can see in the output the walk values they have been plotted here and we can see here in the output. So, this is very much similar to what we see in stock prices, so stock prices if you because the fluctuation that we see is in a pattern is going in any direction, if we look at day to day changes.

So, therefore that is why many people believe that stock prices they are actually random walks. And also many people believe that and statistically it you know they prove that it is very difficult to make prediction of you know series which follow a random walk. So, now that was using you know the built in random module, now we can use np.random module to perform the same thing.

(Refer Slide Time: 12:43)



Now you would see you know that it would be much easier here, we would not have to run a loop to achieve that thing. So, you can see first thing to notice that no loop required, so the first thing we are calling np.random.rand int function. So, here again the value would be 0 and 2, so the value actually would be using this function we would actually be you know getting either 0 or 1.

And for 1000 steps we will run this you know this particular function and therefore 1000 values will have. Then we are calling np.where function where the previous output draws that we had just that we will get using ran in function they will be compared with 0 if we greater than 0 than 1 otherwise my -1. So, you will have an array 1 and -1 and then we will have the walk1 where we can actually computer this cumulative sum of these values.

So, let us run this, so let me run this and let us have a look at the draws the first output. So, you can see these are the draws here and then walk1 you can see these are the values. And then we can plot this again and you can see these kind of values and this kind of plot we have obtain here. So, this is how using different you know modules 1 using the NumPy random module and 1 using the build-in module and you know how we can generate these plots.

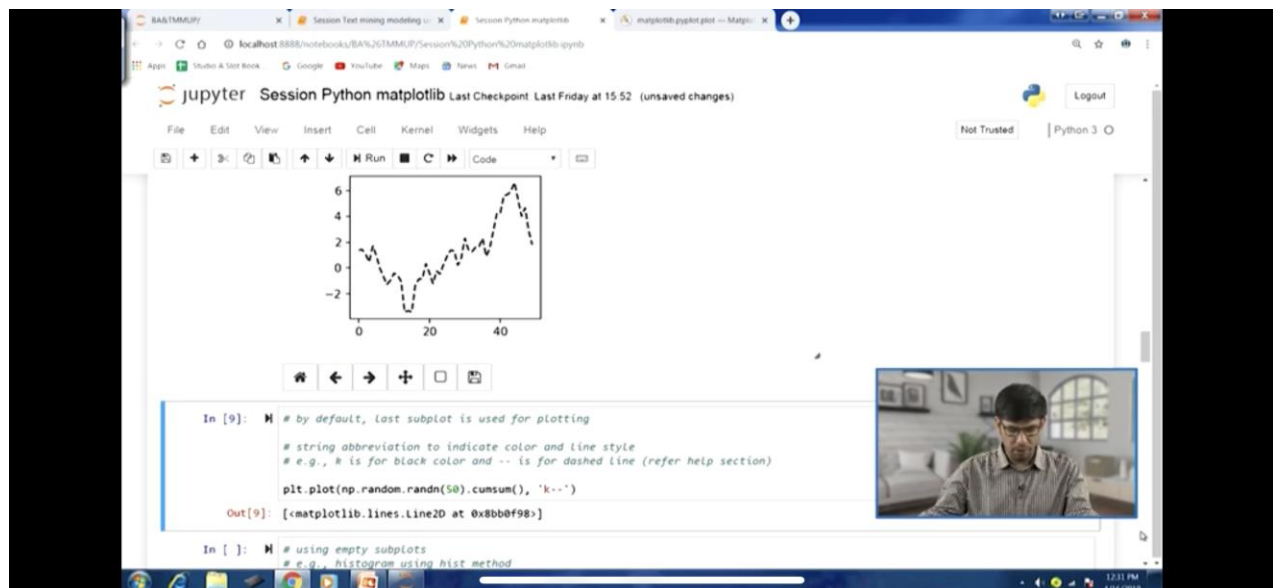
And you can see you know that NumPy random module makes it slightly easier and faster and

convenient to you know actually create these random walks. Now few statistical operation can also be perform on these walk objects. So, let us talk about minimum value, so along this walks trajectory we can call this win method here. So, walk1.min and we will be able to compute the minimum value in this path similarly maximum value along the walk trajectory.

So, we can use this method maximum, so walk1.max and we will get the maximum output, so here you can see 27. Now if we want slightly complex statistics you know to be generated from this trajectory is walks trajectory. So, that can also be done for example first crossing time, so what this is about this is a step at which the random walk reaches a particular value, so let us say that value is 20.

So, we would like to find out this step the first step at which this value was actually achieved. So, this is what we are referring as first crossing time. First time when this random walk actually cross that particular value, so this can be done using argmax function you know however we will have to do certain processing before we reach to the final output. so what argmax function does, it returns the first index of the maximum value in an array.

(Refer Slide Time: 16:16)



So, let us say 1d array given it will have a maximum value but it might appear many times in that array. So, what argmax function will give us, it will return the first instance of you know that

maximum value. So, let us apply this walk1 let us apply this argmax method on walk1. So, walk1.argmax and 890 is the you know first index where this maximum value 27 has been reached.

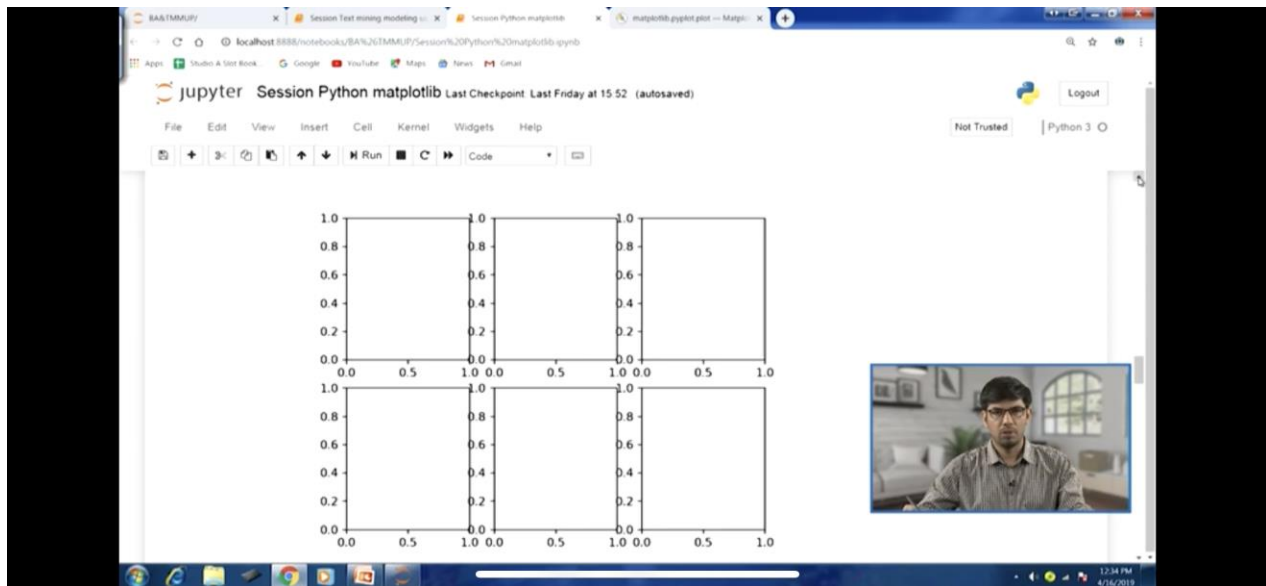
We can check it back we can confirm it back you know by accessing this index, so if I try walk1 and within the brackets if I pass on 890. So, you will see we will get 27, so out of 1000 indices that we have is the 890th you know index where we actually reach this particular maximum value 27. Now we wanted to actually implement the first crossing time, so for that we will have to use this boolean array.

So, in case of a boolean array true is the maximum value because boolean array would be comprising only true and false. So, in a numeric sense true is the higher value typically represented by 1 and false is typically represented by 0. So, therefore if I even a you know argmax on a boolean array, the implementation will you know allow us that in that sense that true would be treated as the maximum value.

And we will get the first index of you know true value there using argmax function. So, what we are going to perform is to achieve the first crossing time statistics will use argmax function on a particular you know expression, boolean expression to achieve this desired result. So, if you look at the expression that we have in this line of code what we are doing is np.abs.

So, we are calling the absolute function, we are passing on this list walk1 and you know from here you know we are comparing this we have the conditioner `(())` (16:08) and we are comparing it with the 10 greater than or equal to 10. So, if we look at the absolute values of this walk1 you know list. If all the elements where the value is greater than or equal to 10, they would be done true.

(Refer Slide Time: 19:21)



So, now out of this will be a boolean array, now we are applying `argmax` on this particular boolean array. `argmax` as you can see in the code. So, therefore the first instance of true value will be written and that would actually be the same as the first crossing time. So, if I run this we get 29 and if I access `walk1[29]` if you know to just to confirm you can see the output is 10. So, at this particular 29th index, first time this particular value of 10 this absolute 10 level was reached.

So, this is how slightly complex statistics can also be easily computed you know using numerical you know python package. Now let us take another example, now this time we would be implementing multiple random walks. So, let us take for an example 50 walks of 1000 steps each, so let us say `walks=50` then number of steps in each of those walks 1000, then we will perform the random draw.

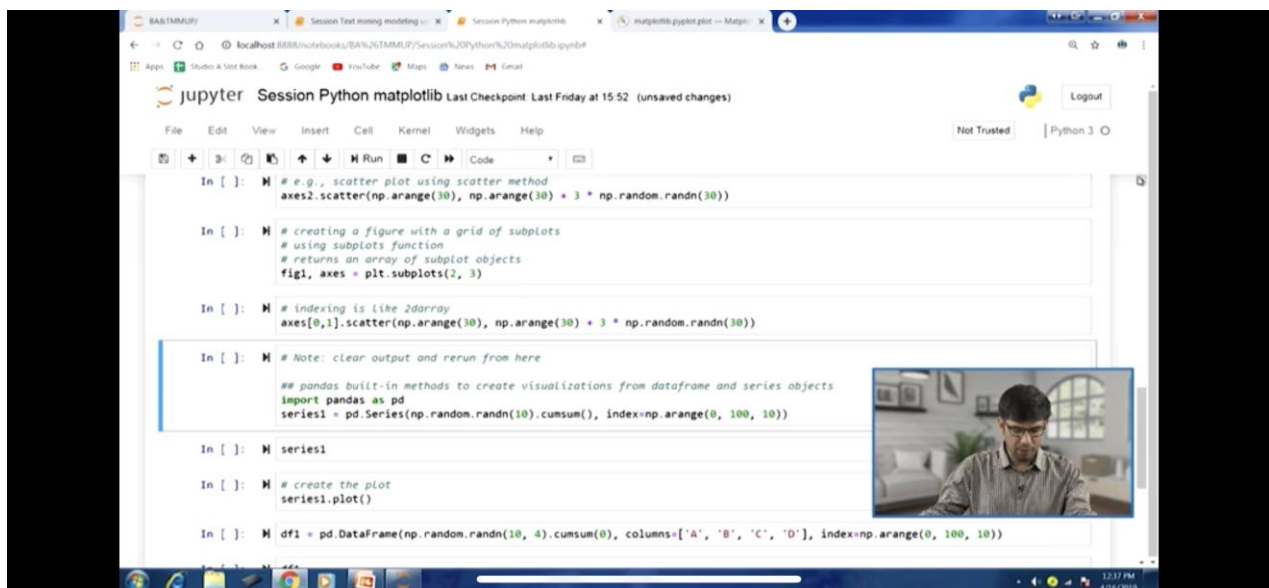
So, we are calling `np.random.randint(0, 2, size=(walks, nsteps))` and so this will `randint` will be written as 0 or 1. So, we will have this you know 2d array filled with these values and then we will use `np.where` function to actually you know we will compare `draws1` with 0 or we will make it 1 or -1 to actually implement the random walk. And then we will take the you know cumulative sum to actually achieve that.

So, so this would be along the second dimension that is why you see that `steps2.cumsum1`.

Because for you know every walk because we are having a fifty walks for every walk we will be performing this separately, so cumsum on the x is 1, that is you know second dimension. So, if I done this let us have a look at these steps2, so you can see here this is a 2d array here filled with the ones and -1 values as you would expect and then walks2.

Here you will have the cumulative sum, so you can see this is also 2d array and you can see clearly if we perform the cumulative sum you know pluses and minuses and then you will have these values. Now if I just confirm from the dimension here, if I access the shape attribute here. So, 50 elements in the first dimension that is because 50 list we have that means 50 box we have and in the second dimension we have 1000 elements.

(Refer Slide Time: 20:59)



The screenshot shows a Jupyter Notebook window titled "Session Python matplotlib". The code in the notebook is as follows:

```
In [ ]: # e.g., scatter plot using scatter method
axes2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))

In [ ]: # creating a figure with a grid of subplots
# using subplots function
# returns an array of subplot objects
fig1, axes = plt.subplots(2, 3)

In [ ]: # indexing is like 2darray
axes[0,1].scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))

In [ ]: # Note: clear output and rerun from here

# pandas built-in methods to create visualizations from dataframe and series objects
import pandas as pd
series1 = pd.Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10))

In [ ]: # series1

In [ ]: # create the plot
series1.plot()

In [ ]: # df1 = pd.DataFrame(np.random.randn(10, 4).cumsum(0), columns=['A', 'B', 'C', 'D'], index=np.arange(0, 100, 10))
```

Because each of those list, each of those walks will have 1000 steps 1000 values there, so that is why this dimension. Now from walks2 and if we call the min method here will get the minimum value across all walks that we have, so that comes out to be -86. And similarly we can also find out the maximum value that we will be across you know all these you know elements, so walks2.max this is 103.

Now if we want to implement the similar kind of slightly complex statistics there when we just had 1 walk we implemented first crossing time. Now here we are having 50 random walks, so

instead of first crossing time let us find out of these 50 walks which walk actually had the minimum crossing time. So, that is something and interesting statistics that we would like to you know find out.

So, minimum crossing time for any of the walks to reach a particular value, so let us take you now absolute 30 level. So, again you know, so first we will find out which of the walks reach the value. So, for this we can see we are trying this in similar kind of you know a code here `np.abs of walks2` we are looking at just the absolute value. So, greater than or equal to 30.anyone to dot any along the we are passing the axis as 1s.

So, that means along second dimension because we would like to work on different walks and any of these dimensions whether they have reached because not all of these random walks you know might be reaching this absolute 30 label. So, first we need to find out which one of them is actually reaching that, so using this expression we are just doing that. So, we will have a boolean output and you can see you know 1d boolean array (()) (21:15).

So, we will have you can see in this list we have 50 values 50 walks and not we can clearly see there are many false values. So, not all walks have actually reached this absolute 30 label, now number of walks you know hitting 30 if we want to find out that count. So, we can use `hits.sum` function and we will be able to find out how many actually reach that label. So, if I run this you can see output is 37 sort of 50 around 37 walks actually were able to hit this absolute 30 label.

So, now our idea is to find out the label minimum crossing you know point. So, for this again we will use the `argmax` function, so you can see here the code is quite similar to previous lines that we have used. `np.abs walks2`. Now we are just working on the hits the walks which have actually reached the 30th level and greater than or equal to 30.`argmax` and so this will again be along second dimension because we are working on walks.

So, here again we will get the you know the output would be you know boolean you know array 1d array and from there `argmax` will find out the you know first instance where you know that happened. And therefore in the first hits we will have you know those indices, so far all of those

37 you know walks that these 30th level will have the indices where it happened, so let me run this and first hits.

Now if you look at the output of first hits, so I think it would be 37 values and if you look at the you know difference. So, for different walks different you know index is there 2 reach this label, now let us find out let us use the min method to find out the minimum crossing time here. So, we will call it mct and first hits.min, so if I run this and you know you see that 135, so if you look at the output number 483, 484 you can see 135 is the index with minimum index level where we please the absolute 30 level.

And if you look at the output 483, so you can see it is actually the 4th you know value in this output, that is actually 135. Now if you want to you know confirm some of these things, so what we will do we will run few more things here. So, what we are going to find out now in the output number 483 the index where this minimum value was achieved. So, for this we are writing this kind of code list of first hits equal to mct.

So, the elements where this mct happens because there can be more than 1 instances as you can manually see in the output number 483, there I can at least clearly see 2 values with 135 you know index. So, then we will apply index true, so we will find out which one is the you know first index there this is another way to find out this thing, index method. So, if I run this line of code 3 is the output, so you can see in the output of number 483 it is actually 3rd index.

Now that we can check manually also but through code also we can come from the same. So, all this is 0 index, so it is actually 0, 1, 2 and 3 4th value. Now if we want to find out the index of true value you know, so out of 37 you know true value that we earlier worked on. So, index of the true value you know among these you know 37 true values here, so so we can find out that again for this we can use this code.

(Refer Slide Time: 23:33)



So, we are going to count true and we are running a loop to you know find out the true value which is mct_int+1 th among true values in hits array. So, in hits array we had 37 values, so you know which index was you know this 4th true value. So, we are essentially looking for 4th true value in the hits array, for this we will have to count the true values here and we are running this loop and you can see for i in range length of hits.

If $count_true$ less than mct then you know and if it is true then we will update the count if otherwise will continue. And then if the condition of a you know counting_true is you know violated it is not less than or equal to mct int. Then we will break out of this loop, so let me run this and let me check the value of I , so it comes out to be 8. Now the appropriate index would be less than would be $i-1$.

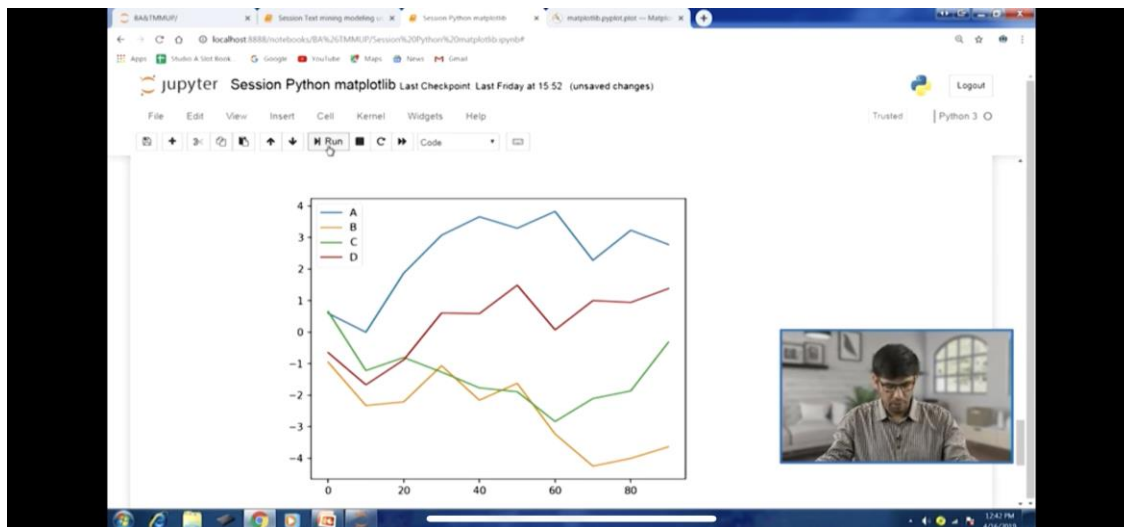
So, in the walks2 you know added that we had will use $i-1$ and mct to find out whether at particular that you know index that minimum crossing time 135 whether you know we really had reached the 30 level. So, if I run this you can see the output 30, so $i-1$ that an mct in that particular walk you know in that. That means it happens to be you know 7th index that means in the 8th walk we reach the 30th level at the minimum crossing time that was 135th.

So, this is how these kind of interesting statistics can really be generated, so with this we are able to you know complete our discussion on numerical python aspect. That is relevant you know in the analytics context and you know. Now we will start our discussion on another important package which is pandas. So, for this we have created another script here another file, so I will run this.

Now we will briefly start our discussion on this particular package and then we will continue in the next lecture. So, we now have a good enough view of numerical python package, we have looked at python basics, we have looked at built-in capabilities that are there, we have looked at the numerical python package. Now we are go into the pandas, pandas is the real package which will you know facilitate much of our you know important tasks in the analytics that we have to perform.

So, this package actually is use to work with the heterogeneous or tabular data. So, typically as you might be familiar with other you know NPTEL courses, business analytics and data mining modeling using R part 1 and part 2. That we are typically dealing with matrix format data, tabular kind of data and their each column is going to represent a different kind of variable, so those are the kind of data sets that we typically deal with.

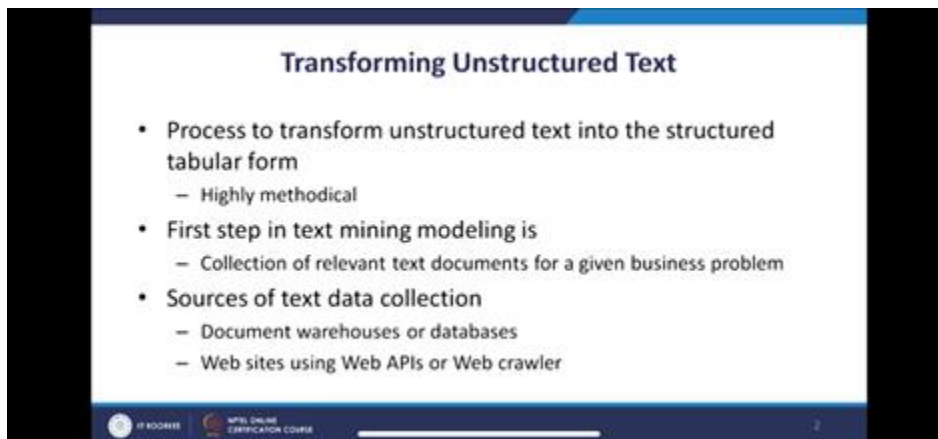
(Refer Slide Time: 25:40)



So, it is the pandas package which typically helps us in facilitating in working with those kind of

data. So, pandas package use to work with heterogeneous data, tabular data, facilitates faster and easier data processing. So, that now, so from this discussion onwards, this part of the python the pandas package will we are going closer to the you know our analytics aspects where we will see how python is going to be really useful in the analytics in general and text mining modeling in particular.

(Refer Slide Time: 26:00)



So, more focus is required from you know pandas onwards, so to import the particular package we can write this import pandas as pd. So, first thing is pandas is the full name and we would be using this abbreviation here. So, first run this and then frequently used libraries would be series and data frame. So, will be talking about these 2 important data structures series and data frame here.

So, you would see that data frame the name is quite similar to the data frame that we have in R platform. So, you will notice slightly you know similarities in terms of data frame structure that we have in the pandas in python and what we have learned in R platform in those 2 previous courses. So, for this will be importing these library series and data frame, so let me run this. Now the next thing is our discussion on these 2 important data structures series and data frame.

(Video Ends: 30:29)

So, we will stop at this point and we will pick up our discussion on these data structure in the next lecture, thank you.

Keywords: NumPy, pseudo random function, data frame, data structure, Panda, loops, charts etc.