NumPy Full Python Course - Data Science Fundamentals

52 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=4C_MWNYDBHQ
https://www.youtube.com/watch?v=4c_mwnYdbhQ

SUMMARY

This video provides a comprehensive crash course on NumPy, a fundamental Python library for numerical computing, particularly useful in data science and machine learning. The tutorial covers installation, basic array operations, mathematical functions, and advanced features like reshaping and random number generation, emphasizing the performance advantages of NumPy over standard Python lists.

- *NumPy is essential for data science and machine learning, serving as the foundation for libraries like Pandas and TensorFlow.*
- *Installation is done via `pip install numpy`, and the library is typically imported with the alias `np`.*
- *NumPy arrays are more efficient than Python lists, supporting multi-dimensional arrays and optimized for linear algebra operations.*
- *Key features include array creation, slicing, and accessing array attributes like shape, size, and data type.*
- *Mathematical operations on NumPy arrays apply element-wise, unlike Python lists, which do not support direct arithmetic operations.*
- *NumPy provides functions for generating arrays with default values (zeros, ones), reshaping arrays, and flattening them.*
- *The library supports random number generation and statistical functions for data analysis.*
- *Arrays can be saved and loaded in both NumPy's binary format and CSV format for easy data management.*

what is going on guys welcome back in today's video i'm going to provide you with a full numpy crash course for python for those of you who don't know what numpy is numpy is a library for python that allows you to work efficiently with vectors matrices arrays it's mainly written in c but it is a python module so you can just import it work with it and it is the basis for most data science and machine learning

libraries like pandas scikit-learn math.lib tensorflow and so on and if you are interested in data science in machine learning numpy is a minimum requirement you need to know what you need to to be familiar with it you need to work with it and another thing i want to show you here is if you go to the stack overflow developer survey and you go to the section other frameworks and libraries you can see that numpy is actually on

rank 2 when you look at all the responses so three point eight four percent and if you only look at professional developers it is still at rank number three after the dot net framework in the dot net core framework so numpy is definitely not a niche thing and also pandas and tensorflow that you see here are actually based on numpy so learning numpy as a data scientist is a very important thing to do a minimum requirement in today's video i'm going

to give you a full crash course on this library all right so my goal for this crash course is to keep things as simple and straightforward as possible and because of that we're going to get right into it with the installation the first thing you want to do is you want to open up the command line of your choice so cmd on windows or terminal on linux and mac and then you want to type pip install

numpy to install numpy so run that it's going to install numpy and once you have that you're going to be able to import it and you're going to say import numpy snp now this as np is an alias it's optional you can also do it without it so you can just import numpy and then whenever you want to call something you can just say numpy dot and whatever you want to use but this alias np is basically

the conventional best practice way to do it in all the documentation in all the code samples you're always going to find import numpy snp so i recommend you also use the alias np it's also shorter and easier to work with so the basics of numpy arrays are actually not that different from the basics of ordinary python lists remember when we have ordinary python lists we create them like that a equals one two three four five and then we can do all

sorts of things like print a we can also print a specific index so a index one would be position two by the way some basic python knowledge is a prerequisite for this course because you should know basic python like lists functions and so on before you get into numpy or data science in general but yeah you can access individual positions here you can also slice you can say front position on up until position four if we have that many positions you

can do print a and then negative index and all that so those are ordinary python lists the same can be done with numpy arrays but they work differently behind the scenes they're written in c they're actually optimized for linear algebra and how you create a numpy array is the following way np dot array and you pass in the constructor a python list so for example one two three four five like that and then we can print that we can also

print the type of a and when we run this you can see that we have the list down here it's a numpy.nd array so that's the type and what you see here when you when you print it is that ordinary python lists are separated by commas those here like numpy arrays are separated just by spaces and besides that we can use them like ordinary python lists so i can copy this now here and i can say okay give me

index one or give me from one until the end or give me up until negative two so this also works with numpy arrays it's not unique to to python list and what we get here as a result is also numpy array so we don't get python lists here so that's quite simple we can also use the same thing for assignment i can say a and then position two equals ten and then i can print a and you're going

to see that this changes so that's very simple very basic what we don't have with python lists though are some of the attributes that we have with numpy arrays first of all let me show you that we can also create multi-dimensional numpy arrays so i'm going to call this a mul or maybe a underscore mall and then np dot array and we're going to have a simple one two three oh i need an additional square bracket

here four five six it's like a multi-dimensional list seven eight nine and then close it again and we can do the same thing here right so we can say a mall and then we can say just zero to get one list we can also say zero one to get the second element of the first list which is two so we can run that as you can see too and it also works like that so that's also similar to ordinary python lists

but we also have attributes that we cannot access that easily with python lists for example i can print a underscore mall dot and now we have different things so first of all we can print the shape the shape is obviously the shape of the dimension so 3 times 3 is in this case because we have one list here and this list has three lists and each of those lists has three elements so three by three if i now add

i don't know i'm going to add a 1 in each list here if i now run this you're going to see that we have 3x4 so essentially you can think about this 3x4 like a sentence you have three times four elements three lists of four elements of course you can also scale this so you can say now i don't only have that but this itself is a collection and then i have another such list

like that for example okay now i messed up the formatting but now i can say one one one one i don't know then one one one and one one one one there you go then maybe do it like that and now this should be what do we have we have two two lists consisting of three lists with four elements so two times three times four should be the shape there you go two three four and you can get the shape like that

and you can also get something else you can also get the dimensions so a underscore mal dot endem will give us the number of dimensions so how deep the whole thing goes in this case three because we have three levels off of deepness you could say or of depth so essentially you have the elements which are in a list these lists here are in another list and these two lists are in another list so we have a d a depth

of three we can also go with the size so we can say a underscore dot size this will give us the amount of elements so basically this is the same as saying two times three times four just multiplying the individual yeah the shape the the shape values and then you get the total number of elements and last but not least we can also access the data type so we can say amol d type now you might say d type

since when do we care about data types in python with numpy is different because numpy is written in c and a reason for why it's so fast and so optimized is because it is statically typed to some degree and because it is it is a bit more strict and statically typed essentially and you can see that in this case the data type is int 32 now in order to see the static typing of numpy in action let's go ahead and

create an array with multiple data types so np.array and in there we're going to have one two three let's make it multi-dimensional and here i'm going to say four and now i'm going to pass a string hello and then six and remember this is possible with ordinary python lists so ordinary python lists can have elements of different data types but in numpy this does not work

so if we go ahead now and say print a dot d type you're going to see that it's no longer in 32 it is this data type here which is like less than u11 this basically stands for a string with less than 11 characters or less or equal to 11 characters so essentially the data type of the array now is string this is the essence of this of this information here so we can see that this is actually the case by

accessing an element that is not a string or was not a string in the first place so if i go a0 0 which is a 1 we can see that it still prints a 1 however if i go ahead now and i say type of this thing you're going to say that this is a numpy string even though it was an integer and if i change this here from hello to 5 we're going to see that it stays in

integer in 32 so a 32 bit integer this is not the case if we have a string in here then all the individual elements have to be adjusted everything becomes a data type like that so basically a string and i think we should also be able i'm not sure about this but i think we should also be able to call the d type attribute here or to access it there you go so you can see it's

less than u1 basically character of length one so so you can see that specifying a certain or mixing data types changes the data types into the same data type so we can type cast these numbers into strings we cannot type cast hello into into an integer now we can try that we can specify here d type equals and then np dot int 32 we can do it like that but we're going to get a problem

because we cannot typecast hello into an integer so what we can do though is if we don't have hello but we have 5 as a string this would work and then we would see that the whole thing has the data type in 32 this works if we have a 5 now if i don't specify that i want to have this data type it's still going to be a string because yeah this is a string so if i

don't say i want to typecast it it's going to keep it as a string i need to provide here that i want to use a certain data type i think this should also be possible for example with a float so float 32 should be able to type cast all of this into into floating point numbers and then we can also go to one one which is the 5 and we can see that the data type is float 32 i can also print it

a 1 1. does it work like that as well there you go 5.0 even though it started as a string so this is important to keep in mind whenever you have something in a numpy array that does not or whenever you mix data types you're going to have one data type in the end now if you do something more extreme like let's do something crazy let's import from some library or maybe i don't know classifier no that's too much i think a

dictionary is enough let's go with a dictionary d equals and then i don't know a or one goes to a and if i now pass this dictionary here i'm not sure if we have a separate data type for dictionary now let's change this here you're going to see that the data type is object so whenever you have something that is not a primitive data type that we can type cast into everything becomes

an object because at the end of the day this means dynamic typing so if i want to have these numbers in the same data type as the dictionary i have to do it with python style of objects i cannot do it with with c data types i have to do it with objects so now we're back to dynamic typing i can add all sorts of things and it's going to to remain objects so i can add strings

and everything is still going to work and we're also going to be able to print a 1 0 for example which is the 4 and it's still going to work but i think when we go to d type this is also going to be an object actually it's an integer so i think that when we're working with the object data type they go okay now now it switches back to the default python data type so remember when everything worked out

when we only had integers or we were able to type typecast we had int 32 from numpy now that we have the object type we have the ordinary python in so this is different this is the the the unoptimized python integer whereas the np in 32 is different so when you mix up data types that get too complex you're going to have less efficiency you could say so that's the basic thing you can also specify the data type by

the way if you have for example only numbers but you want to have them as strings so if you have it like that but you want to type cast into a string you can go ahead and provide a string here and say less than u 2 for example and i think this should work out there you go you can see that we have string and you have less than u2 i'm not sure if we can specify whatever we want here

or if it's going to go to the yeah actually we can do whatever we want so this provides a data type and or basically you can you can force type casting if possible by providing this data type keyword and the key essence here is that in numpy we have to focus or we should focus on one data type in an array and when we mix things up too much it goes back to ordinary python data types

and python objects so if you want to have a full list of all the data types that numpy provides you can go to documentation i have the link here even though i think you should be able to google it on your own but this is the link to the numpy documentation you just go to the basic types and now we're going to move on to another topic which is oftentimes you want to create arrays with default values so you don't want to

say a equals np array and then you want to provide a list but you want to have some default values in there or you want to fill with the same value and for that numpy provides you with a couple of functions so for example let's say you want to start with a with an array full of the number seven for example what you do here is you basically say a equals and then you can use the np dot

full function and what you pass to the np.full function is first of all the shape of the array and second of all the value that you want to fill with so for example 2 times 3 times 4 is what we had before and the number i don't know 7 for example let's go with 9 for neural nine is going to fill this array with that shape so you can see here that we have now two times three times four this is the

structured we had before we have two two lists here consisting of three lists with four elements each and of course you can do whatever you want here so you can go ahead and say 10 10 10 and even 10 you know four dimensional with 10 and it's going to be huge as you can see here but this function allows you to create basically a numpy array filled with one value given a certain shape this is a very very nice thing to have

there are also some some functions for default values so if you don't wanna if you don't wanna go with a specific value but a commonly used value you can also say np.0s so here you only provide the shape for example 10 times 5 times 2 and if you print a you're going to get an array full of zeros with that shape the same can be done for once so np ones let's go with the same

shape there you go and one more thing that we have here this is also quite interesting is the npmt function so you can say a equals np dot empty and then you provide a shape for example five times five times five and you might say okay what's the difference between empty and zero the difference is that empty doesn't initialize the value so empty allocates the space this is a more technical thing that you usually don't do in python you don't allocate manually

you just do stuff and it works but in c you allocate memory so what the empty function does is it essentially reserves the space five times five times five and whatever values are there because in the in the memory you have some default values that are there maybe from for whatever reasons for whatever reason there are there are some values there and mt essentially just allocates the space without initializing the values whereas zeros ones and all the other functions

essentially take the space and fill it up with values and this just takes the values that are there in the first place this of course is a bit faster but yeah you have the default values so whatever is there is kind of random and then we also have two functions that are quite useful to generate sequences so what you can do here is you can say let's say you you want to look or you want to generate some x values because

you have certain y values that you want to plot and you want to now generate a range what you can do is you can say x underscore values is equal to np and now a range is the function a range and what you do here is you essentially provide three values the first one is the beginning the second one is the end and the third one is the step size so you can say okay i want to have the values from zero to a

thousand with a step size of five and if i now print the x values you're going to see that this is what we get we get 0 5 10 15 and so on up until 995. we can change this to a thousand and five if we also want to have a thousand this is the a range function we also have a similar function which is the linspace function so we can say x values equals np lin space

and the lin space function essentially provides also a beginning and an end so let's change this again to a thousand to compare provides also a beginning and an end but instead of providing the step size we specify how many values we want to generate so if i only want to generate two values it's going to evenly spread two values i should also print that it's going to to evenly distribute these two values so zero and a thousand if i go with four

it's going to give me these values here and if i go with a thousand values okay a thousand is not what i wanted 1001 there you go then you get the values that you may want to have here so this here provides a step size beginning and step size and this provides beginning and and how many values we want to have what's also interesting to know is that numpy provides us with two special values called nan and inf nan stands for not a

number and in stands for infinity and they can be quite useful if you want to use them you basically say for example print np dot nan like that and np.inf and they don't really have so much functionality but nan can be used for example in data sets if you if you load a huge data set in some columns or some values for some columns are missing you fill it up with nan and then you can drop the nan or you can

change the nand impute the nan and so on in data science and infinity can be useful for example when you get results like a division by zero instead of just throwing a throwing an exception you can just return an infinity value and you can also check for these values by saying print and then for example np is nan so is this a not a number value and then you can provide something for example npnand this of course is going to return

true now and we can do the same thing for is in and i also want to give you some examples for how to create these values i'm going to give you two examples here first of all let's go ahead and say np dot sqrt which is basically taking the square root but with a numpy method for that and we passed negative one here for example i'm not sure if we have to pass negative one or a list of

negative one i think this should be fine this is going to result in a not a number because it's essentially a complex number and because of that it returns a nan and if we try to divide by zero so essentially np dot array we're going to have value 10 here and we're going to divide this array by zero these two things are going to return first of all we're going to get a runtime warning this is not an exception

this is just a warning but we can get true and true here and if i omit this here you're going to see that this actually in fact returns nan and inf so those are the two special values that numpy provides all right so next up let us get into something more interesting which is performing mathematical operations with numpy arrays and for that we're going to make the comparison with ordinary python lists so i'm going to say l1 is going to be one two three

four five for example and l2 is going to be six seven eight nine and zero and then we're going to have a1 being np dot array of the same thing actually i can pass l1 and l2 here and now we're going to do some operations here some mathematical operations to see what the differences are between python lists and numpy arrays and the most basic basic thing that we can do here is we can apply

some operation with a scalar so we can go ahead for example and say l1 times five what happens if you take a python list and multiplied by five i think you know that you basically repeat the list five times this is what happens here if you do the same thing with a numpy array so if you change that to a1 you're going to see that actually what happens is you take the 5 and multiply each element of

the array with that number so it's more like working with vectors and working with matrices then then it is the case with python lists so python lists are basically taken repeated five times and here we do an actual calculation with all the values same goes for all the other operators so if i go ahead and say plus this is also going to work for the numpy array if i go ahead now and i say plus 5

for the python list we're going to get the problem type error because we cannot concatenate a list with an integer so this doesn't work this is the concatenation you can see that this is the case because i can go and say l1 plus l2 with ordinary python list and then you can see that essentially takes one list and the other list and it combines them into a new list whereas if you do that with numpy so if i say a1 plus a2

we get a vector addition you could say so we get one plus six two plus seven three plus eight four plus nine five 5 0 and so on same goes of course for all the other operations so if i go with multiplication here what happens is i multiply all the elements if i go with division we're going to get an infinity value here because we're dividing by zero if i go with what did we miss subtraction right

it also works with python lists most of these things don't work so i don't think that we can multiply two lists here this is going to give us a type error again i also don't think that we can divide lists i'm not sure if we can subtract lists maybe we get the difference no so it's basically all unsupported for ordinary python list with numpy arrays we can perform calculations like that now what's also interesting in numpy is

that we can do that with lists of different dimensions so i'm going to delete these python lists here what we can do is let's say we have the numpy array one two three and then we have the numpy array one and two like that so we have here we have basically just an array of of the shape three or one times three and here we have basically two times one so we have a two dimensional array here

and what i can do now is i can say print a one plus a two and this is going to result in a two by three matrix here because essentially we're doing one plus one two plus one three plus one for the first row and then one plus two two plus two three plus two for the second row so this works as well of course they have to be somewhat compatible so this works because we can create a reasonable

result out of that but if i go ahead now and i add one more dimension here now the two shapes are not really compatible so it's not going to work you cannot use this operand with these two shapes here they're just not compatible so there is some limitation but essentially whenever you have two numpy arrays which you can think of as matrices or vectors you can do the respective calculations you can do multiplications you can do divisions and

so on and so forth numpy also offers us some mathematical functions one of them we used already which is the square root function so np.sqrt and when you apply these functions onto arrays you apply the function onto each element so if i say np array and then one two three and maybe make it two dimensional four five six there you go we can now go ahead and say print np

dot sqrt a and as a result of that we're going to get an array with the same shape with the results of the individual calculations so square root of 1 square root of 2 and so on and the same can be done with the sine function the same can be done with a cosine function and with a bunch of different functions so we have cosine we have 10 we have arc 10 we have x for the exponential calculations we

have logarithm we have logarithm base 2 logarithm base 10 and so on for a list of all the functions again i would refer to the documentation because there's no value in just listing all of them here you can just go to this link here and you're going to find all the important math functions but essentially this is important to know when you use an np dot whatever mathematical function you use onto an array you apply this

function onto all the individual elements and this can also be quite useful at times so next i would go ahead and look at array functions array functions essentially meaning that how do we append to a to an array how do we insert into an array how do we delete from an array and so on and for that we're going to start with a basic appending so maybe let me get back this array here if we have this array what we can do

here is we can or actually for the sake of simplicity let's start with a simple one-dimensional array here like that let's say we have this array and now we want to append some additional values what we can do here is we can go ahead and say np dot append and then we provide a as the base array and then whatever we want to append as a second parameter so for example 789 can be appended here and the interesting

thing with numpy and also with pandas and all the other libraries is that they usually return the result they don't change it so if i print this here we're going to get the result but if i print a afterwards we're not going to get this stored in the actual array so what we have to do is we need to actually say a equals that otherwise it's not going to make the actual change and then we can print

a as a result here as you can see so we can also insert into an array in order to do that we just use the np.insert function and here we specify again a and then the position so we want to insert four five six at the fourth position so actually at or in this case at the third position so index three which is position four and we want to insert 4 5 6.

i think this should work of course we need to assign this a equals np insert there you go one two three four five six seven eight nine now if you want to delete elements from the array you have to use the np.delete function this function is a little bit confusing because it doesn't work as intuitively as you might think so what you have to do is you have to type np.delete and then you pass the array and then you pass the index

followed by the axis so if you just pass an index here for example one it's going to delete the element at that index so it's not going to delete this whole list it's going to go through the elements so zero one two three four five and it's going to delete index one so in this case two if i specify three it's going to delete the four and so on we can actually print the results here so the first one should delete two this

one should delete four and this one should delete five you can see that this is the case however the second parameter is very important because the second parameter specifies the axis if we don't provide an axis this is what's hap what's happening if we provide an axis depending on the number we pass it will either change it will either delete the row or the column so if i say a1 0 it is going to get rid of the

second row so in this case of 4 5 6 because we have row 0 row 1. index 0 means that we're looking at this row wise and if i say delete the row with the index 1 it's going to delete that if i change this to 0 it's going to delete the first row and of course if we have more rows it's going to delete whatever we provide here however if i now specify axis one this is going to delete the column with

the index one so column like one and four are column zero two and five are column one and three and six are column two and you can see here that two and five is missing because we removed column the column with the index one and of course if we have a highly high dimensional array with like 20 axis we can also do this in more complicated ways but of course then it becomes quite difficult for us

to understand all right now let's move on to some more structural methods that numpy provides so methods that are able to change the shape the structure of the array and the probably most important or most interesting one is the reshape function so we have this array here of shape four five so we have four lists with five elements each we can see that this is the case by printing a dot shape you can see four five here

and if we now want to reshape this we can go ahead and say print and a dot reshape notice we're now not saying np reshape we're saying a dot reshape and then we're specifying a shape for example one that is compatible with four five is obviously five four so that would mean that we have five lists with four elements each and we can print the result of that here this would look like that so the order is actually

the same we have one two three four five six seven eight nine and so on up until 20 so the order is not messed up but we have a different shape now and of course we can also do different shapes like for example a very simple one would be 20 with a comma to specify that we have essentially just 20 which is not the same by the way is specifying 20 comma 1 because 20 comma nothing

means that we have just one dimension with 20 elements whereas if we provide 20 comma 1 this would mean that we have 20 lists with one element each so this is what 21 looks like we can also do 210 and a couple of other things so 210 would look like that we can also go with two two five two five two so making it three dimensional this is also possibility two two five

two five two or five two two for example and the difference remember is look at it like a sentence this means that we have maybe let's delete these here so that we have an overview and let's comment these two out so two times two times five means that we have two collections with two lists each with five elements each whereas this means

that we have two collections with five lists each with two elements each and the last one means we have five collections with two lists each with two elements each and so on so you can you can play around with that if you want to you can also do something more crazy like last example for this reshaping here 1 times 1 times 1 times 1 times 2 times 10. you can add as many once as you want

and then you're going to get a lot of lists that are unnecessary but you can also of course do it like that then it looks a bit cooler maybe i don't know so this is how you do the reshaping now what you of course need to do is you need to assign it so if you say reshape you need to also say a dot a equals a dot reshape so 5 4 for example and if you don't want to do that you can

also use a different function so if you don't want to assign it if you don't want to return it but to actually apply it immediately you can use the resize function so if you just call a equals a reshape and then you do 10 2 for example you're going to see that a doesn't change a stays the same because you need to assign in order to change this however if you say a dot resize and you pass 10 2 and you print a

you can see by the way we should pass actually tuples of the shape just as a general guideline here but what we get here is essentially this this actually applies the the transformation onto the array without returning it so we don't have to return it and store it this does this immediately that is that so besides that we can also do stuff like flattening an array so this array here we can flatten it by

just saying print a dot flatten and this is going to give us the one dimensional view on that with a one dimensional version of that and there's a second function that is quite similar to that which is called ravel so ravel is essentially like flatten and you might say okay but what's the difference then the difference is that flatten returns a flattened copy of the array whereas revel only returns a view a flattened view onto the array the

difference is that if i say var1 so variable 1 equals a.flatten and then i say variable 1 at position 10 or at position 2 equals 100 for example then i can print variable 1 and i can also print a and you will be able to see that this here is the copy so variable one

and it changed the value but the original array is not changed in any way whereas if i change this to ravel this is going to change the actual array why because we're not returning a copy you can see here that 100 changed we're not returning a copy that is flattened so we're not returning a new array which is flattened we return the same array with a flattened view on it so we use it as a flattened version but

when we make changes we're actually changing the original array so we're just looking at it from a different perspective that is that and last but not least for the flattening we can also do we can also use the flat attribute so we can say for example variable equals and then we can just say v for v in a dot flat and we can print that

there you go you can see this works as well so the last structural function that we can look at is the transposing or the swapping of the axis transposing basically means swapping or turning it around so making columns rows

and rows columns so if we have this right here one two three four five six and so on we can go ahead and say a dot transpose and this is going to basically swap the axis so we have one two three four five

six seven eight nine ten column wise instead of row wise this can also be done not with a function but with a t so the transposed version can be accessed with a dot t it's the same thing and then there's also a similar function called swap axis or swap axes and this can be done like that swap axes zero and one and in this case it's going to have the same effect as the transposing the difference is that if

you have a high dimensional array so like 50 dimensions for example you maybe want to just swap two axes and then you use the swap axis function because you can specify the two axes in this case we only have two that you want to swap and the transposing basically swaps everything so it transposes the whole array where swap axis picks two axes to swap all right so next we're going to talk about joining and splitting array so

let's say we have two arrays we want to merge them into one array or we have one array and we want to split it into two arrays how can we do that in python or in numpy actually we're going to change this now to a1 being that and we're going to then actually close this here and we're going to say that a2 equals np dot array like this so we now have two separate arrays here

and we want to merge them for example into one array as we had it before what we can do for that is we can use different functions one of them is the concatenate function so we can say a equals np dot concatenate and basically we pass here the two arrays so a1 a2 and then we also pass an axis this is important because depending on which axis you pass it's going to add them in a different

way so if i say axis equals 0 which stands for rows of course i need to print this as well if we do it like that you're going to see that we basically just add the rows we concatenate the rows we have two rows here two rows here and we concatenate these rows if i now change this to one so if i say the axis is one we're going to do it column wise so we're going to

have one two three four five six seven eight nine ten and then we're going to append this as column so we're going to take this and append it at the end of this and we're going to take this and append it to the end of this so depending on the axis you're going to get different results now there's also another function which is called stack and stack does a different thing stack adds a new dimension so here we're we're

concatenating on the same dimension so with concatenate we're concatenating on existing dimensions whereas stack adds a new dimension so what we do here is we basically just say stack a1 a2 and this results in in a new dimension so we now have not just two rows into four rows or two rows combined two and two combined into four rows but we have two additional lists here so adding a new dimension to the whole array here for the combination

and we also have two special versions here which are npv stack and nph stack so vertical and horizontal v-stack

essentially in this case at least does the same as the concatenation on axis zero and h stack does the same as the concatenation on axis one so nothing nothing fancy here this is how you merge to a race what you can also do is you can split so let's go back to the beginning let me just reverse this here

there you go so we have one array now and this array can now be split into multiple arrays into multiple smaller arrays and for that we use the split function the split function can be used the following way a dot split and what we pass here or actually sorry i messed this up we say np dot split and we pass the array and then we decide into how many into how many things into how many arrays we want to split this so for

example we can say that we want to split into two arrays on x is zero so we're taking the rows and we're splitting into two arrays this would be what we did before basically one array from one to ten and one array from 11 to 20. yeah that that's essentially what we do here we can also go ahead and split into four so four rows essentially so we have each row into a separate array we can also go ahead and

actually let me just copy this array here because this is a more interesting example because there we have six elements so we can do more splitting here if we do it column wise we can also go ahead now and split into into twos so two axis one and you can see then that we have an array with one two three seven eight nine 13 to 15 and 18 to 20 here and the

second array is with the rest so we split the columns in half here let me just move this there you go we split the columns in half here we can also specify that we want to have three arrays then we're going to split the columns respectively so we're going to have 1 2 7 8 13 14 18 19 in one array then the middle in one array and the right two columns in one array and we can also

split into six here to have each column in a separate array and that that's basically you specify how many splits you want to have how many arrays you want to end up with and you specify the axis of course if you have more dimensions you can also specify access 2 3 4 and so on which is hard to imagine but this is how you split numpy arrays another interesting thing that numpy arrays provide is aggregate functions so

let's say we have this array here and we want to know certain things like what's the smallest value what's the largest value what's the sum of all the elements we can just go ahead and say a min and we can do the same thing with max we can do the same thing with mean we can do the same thing with std for standard deviation we can do the same thing with sum and by doing that we can get

interesting statistical values we can also get the median but for this we need to use numpy so np.median and we need to provide a as a parameter here but yeah you can see that we can do that easily in numpy this is also not as or actually it's not that difficult with with python list as well because then we just call them in function the max function and so on but this is also something you can do in numpy

and besides that i would like to move on to np random so to the randomness of numpy because numpy allows us

to generate random values and random lists and this can be done by just saying for example if you want a single number you can say number equals np.random dot randint and then you specify a range for example up until 100 and then you get your random number so this is not too special you can also do that

with a basic python randomness but numpy also allows you to do that with a certain dimension so if i change this to numbers so plural i can also specify here the size of the whole thing and i can see okay i want to have shape two three four and now we have a numpy array of that shape with random values and i can also say one here then we're going to get

okay we should probably say two here then we'll go we're going to get zeros and once here i'm not sure if we can specify a range so zero to a hundred for example or actually this is probably not the best test let's go 90 up until 100 does this work as well yeah there you go so you can also specify minimum maximum and the shape to generate random values you can also do that according to a

distribution so if you don't want to just do randomness entirely you can also do for example the result of a binomial distribution so you can say numbers equals np dot random dot binomial and you say for example okay we do 10 tries and we have the probability of 0.5 and the size of the array should be 5 10 for example so like a coin flip the result of coin flips here

so how many times you get head for example out of 10 and this is the result in a numpy array you can also do the same thing with a normal distribution but here for example you can say you want to have the size of students and you can say okay the the mean is 170 centimeters and the scale is 15 so the standard deviation is 15 centimeters and then you get sizes here in that shape also one thing that you can do here and

also what we can do is we can also do a random choice so let's say we have for example np dot random dot choice what we can do here is we can say okay choose from these numbers a random number like that and then we can of course also do that into in a numpy array so 5 10 for example and then we would get choices random choices a couple of times

so this is what you can do with numpy randomness and now last but not least we're going to talk about exporting and importing numpy arrays and we can do that in two ways we can do that with a numpy format or we can do that with csv files so if you want to do that with a numpy file you just say np dot safe and you pass first of all the path so my array dot npy

and then the array it's as simple as that if we do that now we're going to have this new file here you can see it's binary so you cannot really read it but i can now comment this out and i can also comment this out and i can say a equals np load my array.npy then i can print a and you can see that we now have the memory back in the script and the same

can be done with a csv file but the csv file of course can be used for other purposes as well so np dot safe txt is

the function and we save this into myarray.csv the array is being saved and the delimiter is going to be just a comma for a comma separated values file and i'm going to not load it for now i'm just going to save this and here you can see that we have the

csv file with the values and if i now go ahead again and uncomment all of this and i say a equals np.load txt myarray.csv the limiter is comma print a then you can see that we have the same values in this case we have floating point values but the values are the same and of course this can also be

used for data sets so you don't have to store and you don't have to export your arrays and then load your arrays again you can also load from a csv file which is a data set and then work with numpy so you don't have to use your own csv files for that but this is how you can do that in numpy all right so that's it for this numpy crash course i hope you enjoyed i hope

you learned something if so let me know by hitting a like button and leaving a comment in the comment section down below let me know if you're interested in seeing more crash courses like this on pandas matplotlib and so on and of course don't forget to subscribe to this channel and hit the notification bell to not miss a single future video for free other than that thank you for watching see you next video and bye

you