**NUMPY**

Hucreye gelip iki kere d ye basarsan siler. Z ise geri getirir

Satira gelip a dersen uste, b dersen alta bos kod satiri ekler

Pandas, numpy uzerine kurulmus. Numpy daha matematiksel

Pandas ta yaptigimiz seyi matplotlib ile gorsellestiriyoruz

Tableau ile powerBI ile de gorsellestiriyoruz ve dogrudan sunum icin kullaniyoruz

**Pip install numpy**

**Import numpy as np**

Pip install diyip anacondaya numpy ekledim

Bunu anaconda sayfasindan da yapabilirdim

environment

**Numpy Arrays**

**1D Array**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |

Shape (9,)

**2D Array**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

2 rows, 6 columns. This is 2D array.

Shape(2,5)

Shape(satir, sutun)

**3D Array**

Shape(3,5,4)

bu da 3 row 5 column olan bir array’den 4 tane var demek.

My\_np.array([4,5,9]) seklinde gosteriliyorsa mesela:

|  |  |  |
| --- | --- | --- |
| 4 | 5 | 9 |

**Array Creation Methods**

-Manuel olusturabilecegin gibi, arange, zero, empty, full.. gibi methodlarla da array olusturulabilir. Bir de random ile rastgele olusturabilirsin.

1. Once bir liste tanimlayip, Np.array(mylist) diyebilirsin array olusturmak icin
2. Np.array([1,2,3]) 3 tane koseli parantez yaparsam 3 boyutlu array olur
3. random

**-shape methodu:**

Bu kac satir kac sutun oldugunu verir sana

My\_array2.shape dersem mesela

**-reshape methodu:**

**My\_array2.reshape(x,y) x satir, y sutundan olusan yeni bir array olusturuyor**

Boyut degistirmeye yariyor

Data.reshape(2,3) dediginde 2 row, 3 column 2D bir arraye donusturur, ama data sayisi yetmeli bunu yapabilmek icin. Mesela 3 column 2 row olan 6 datali bir arrayi, 3 row 2 column olan 6 datali baska bir array’e ceviriyor.

Reshape(-1) yaparsan tek boyutlu hale donusturur.

# reshape uzerine reshape yapabiliyoruz

# 2 boyutluyu 3 boyutlu hale de getirebiliyoruz.

**myarray.reshape(12).reshape(2,2,3)**

#2 satir 3 sutundan 2 tane seklinde 3 boyutlu hazirliyor boyle yazinca. Bizim dizimizi o hale getiriyor, donusturuyor.

#-1 bilinmeyen bir boyut, verdigimiz satir ve sutuna gore boyutu ayarliyor mesela

**-np.empty**

Np.empty([2,2]) diyelim mesela

Rastgele degerlerle dolu bir array olusturuyor 2 satir 2 sutunlu

Avantaji, bellek tahsis etmenin maliyetini azaltiyor

**-dtype**

**My\_int.dtype**

# elemanlarin veri tipi demek bu. kapladigi boyut sayisiyla alakali

#array’ler listelerden farkli olarak homojen veri tipi icerir.

#hem integer hem float olabilir, formatini tek bir tipe cevirip homojen olarak saklanmasini sagliyor

# yani hem integer hem float koyarsak, tipini float yapar.

#numpy'lar homojen, tek bir veri tipine cevirerek saklama yapiyor.

X=np.array([1,2], dtype=np.float64) dersen de mesela, bu da tipini float’a cevir demek

1. 2.] hale getirir yani

**-astype**

Bu da arraylerin dtype’ini degistiriyor

Astype(‘int’) mesela int’e ceviriyor.

Astype(bool) mesela bool’a ceviriyor tipini.

**Size kullanimi**

Method olarak kullanildiginda **a. size** eleman sayisini verir.

Parantez icinde parametre olarak kullanilan **size=(3,5)** ise shape’ini belirler

**Resize**

Icindeki elemanlarin sayisina bagli olmadan yeniden boyutlandirabiliyor.

Reshape eleman sayisina uygun olursa yeniden boyutlandirabiliyordu.

**Np.resize(array, (boyut))**

**CONCATENATE**

Iki farkli arrayi birlestiriyor.

0 x ekseni, 1 y ekseni diye gecer.

Axis 0 yaptigimda alt alta, axis 1 yaotigimda yanyana ekleme yapiyor arrayleri

**Sort**

Np.sort() dedigimde kucukten buyuge olarak siraliyor

2 boyutlu ve daha fazlasi icin de siralama yaptirir.

Axis 0’a gore siralama yaptirdigimiz zaman satir indekslerini aliyor kendi icinde (column icinde) siraliyor

Axis 1’e gore sirala deyince sutunlari aliyor kendi icinde (satira gore) siraliyor

np.sort(myarr, **axis=None**) #axis olmasin yani 2 boyutlu olmasin dersek (normalde 2 boyutlu olan bir array icin)

#yani bir acidan vektore cevirirsek, tum sayilari kendi icinde siralatmis oluruz.

**Fromiter**

Fromiter da iterable nesneden array olusturuyor.

My\_iter= (i for i in range(5))

**np.fromiter(my\_iter, int)** mesela 0,1,2,3,4 elemanlarindan olusan bir array olusturuyor

**Arange**

Python’daki range() fonksiyonuna benzer.

Belirtilen başlangıç değerinden başlayıp, her seferinde adım sayısı kadar arttırarak ,bitiş değerine kadar olan sayıları bulunduran bir numpy dizisi dödürür.

Not: Bitiş değerinin diziye dahil edilmediğine dikkat edelim.

Start vermezsen 0’dan kabul eder, step de gerek yok. Ama stop degeri vermen sart

**Np.arange(10)** 0’dan 9’a kadar array olusturur. (np.arange(0, 10) da )

**-np.linspace**

Start, stop degerler ile kac veri istedigimizi soyleyebiliyoruz. Linear bir array olusturuyor

Esit aralikli bir dizi donduruyor.

**Np.linspace(start, stop)**

Default olarak 50 tane **float** deger olusturuyor. Ama kac tane deger istedigimizi da yazabiliyoruz.

**np.linspace(1, 20, num=10)** dersem mesela 10 tane deger olusturur.

**np.linspace(1, 20, num=10)**

np.linspace(0, [10, 20],5, **axis=1**) # 0'dan basla 10’a kadar git

0’dan basla 10'a kadar git, **sutunlara yerlestir** demek bu

np.linspace(0, [10, 20],5, **axis=0**) dedigim zaman ise,

Aynisini da ikinci satirda, 0’dan basla 20'ye kadar git, **satirlara yerlestir** demek bu

**-np.logspace**

Start, stop degerler ile kac veri istedigimizi soyleyebiliyoruz. Logoritmik araliklarla bir array olusturuyor

np.logspace (1,3) #linspace'den farkli olarak base var, tabaniizi gosteriyor

#defaultu 10. 10\*\*1 10\*\*3 u hesaplayip arasina 50 tane logorotmik deger dondurur

Base=2 dersem de 10’luk degil, 2’lik tabanda hesaplatiyor.

**zeros**

Belirtilen satır ve sütuna sahip 0'lardan oluşan bir matris döndürür.

0’in bool type’i False oldugu icin, dtype bool dersem icini False ile doldurur.

**-ones**

Belirtilen satır ve sütuna sahip 1'lerdan oluşan bir matris döndürür.

np.ones([3,5])\*55 diyerek mesela Numpy’da carpma yapabiliyoruz. Vektorizasyon dedigimiz olay bu

**-np.eye(3)**

-bu mesela 0. indeksten basliyor, diagonal sekilde 1’leri doldurup geri kalani 0’larla doldurdugu bir array hazirliyor. K=2 dersem de icine, 2.indeksten baslayarak bunu yapar.

Np.eye(3,3) mesela asagidaki arrayi verir:

array([[1., 0., 0.],

[0., 1., 0.],

[0., 0., 1.]])

**full**

Sabit sayıdan oluşan matris oluşturur

np.full((2,2),10)

**#zeros 0 ile, ones 1 ile, full de kac yazarsam onla dolduruyor**

**Rand**

0 ile 1 arasinda **uniform dagilim** ile degerler olusturarak degerler donduruyor.

#UNIFORM DAGILIMLI RASTGELE SAyilardan olusan array uretiyor

#degerlerin esit bir olasilik dagilimda gelmesi demek bu

**np.random.rand(3,3)**

**Randn**

random.randn(), ortalama değeri 0 ve standart sapması 1 olan normal dağılıma sahip rastgele sayılar üretmek için kullanılır.

Fonksiyon, normal dağılımdan rastgele sayılar üretir ve bu sayıları NumPy dizisi olarak döndürür.

**plt.hist(np.random.rand(50000),bins=20)** #bins20 demek 20 tane aralik olustur. sayilari araliklara koy

#bunu da grafikte goster demis oldum.

**randint**

np.random.randint(1,100, 10) #size da girersem bu sefer 10 tane random sayi uretir

Start stop degerlerini yazdigimda, o degerler arasinda getiriyor

Size=(2,4) dersem de icine, 2 satir 4 sutun olacak sekilde array yapiyor

**max & argmax & min & argmin**

**min()**, bir dizinin içindeki en küçük değeri döndürür

**max()**, bir dizinin içindeki en büyük değeri döndürür

**argmin()**, bir dizinin içindeki en küçük değerin indeksini döndürür

**argmax()**, bir dizinin içindeki en büyük değerin indeksini döndürür

**-ndmin**

-boyut sayisini ayarliyoruz

myarray=np.array(mylist, ndmin=4) dedigimizde 4 boyutlu hale getiriyor.

**-Ndim**

#ndmin istedigin boyuta ceviriyor, ndim ise kac boyutlu oldugunu veriyor

Ndim=number of dimensioni verir

Ndmin=kac yazarsan sana o kadar boyutlu array veriyor

**itemsize**

itemsize(), NumPy dizisinin her bir elemanının bellekte kaç byte kapladığını gösterir.

Bu özellik, NumPy dizisinin veri tipine (dtype) bağlıdır ve bu veri tipinin boyutuna göre değişir.

itemsize(), bir NumPy dizisinin bellekte kapladığı toplam alanı hesaplamak için de kullanılabilir. Bu hesaplama, dizinin eleman sayısının itemsize() ile çarpımı ile yapılabilir.

float32 tipi 32 bit (4 byte) bellek kullanır ve yaklaşık 7 basamak hassasiyetle ondalık sayıları temsil edebilir.

32 bitlik bir sayı, 2^32 farklı sayıyı ifade edebilir.

float64 tipi ise 64 bit (8 byte) bellek kullanır ve yaklaşık 15-16 basamak hassasiyetle ondalık sayıları temsil edebilir

64 bitlik bir sayı, 2^64 farklı sayıyı ifade edebilir.

-Bellekte ne kadar yer kapladigini veriyor itemsize. En cok kullanilan 64’tur.

-Hassasiyet onemliyse 64 kullan

|  |
| --- |
| **arr.dtype** #data tipini verir  **arr.size** (kac eleman var onu verir)  **arr.itemsize** #herbir eleman kac byte yer kapliyor onu verir. (8 diyelim)  **arr.itemsize\* arr.size** #6 eleman var, her biri 8 byte toplamda 48 byte yer kapliyor |

**copy**

copy(), Numpy dizilerinde bir kopya oluşturmak için kullanılır.

Bu fonksiyon, orijinal dizinin aynısını oluşturur ancak bu kopya bağımsızdır. Yani, kopya dizisi üzerinde yapılan herhangi bir değişiklik, orijinal diziyi etkilemez ve orijinal dizi üzerinde yapılan değişiklikler de kopya dizisini etkilemez.

Verisetinin aynisindan kopyalamaya yariyor. Array kopyaladiktan sonra, array’de degisiklik yaparsan, kopyasini cagirdiginda degisiklik yapmadan onceki halini cagirabilirsin.

**Transpose**

transpose() fonksiyonu, NumPy dizilerinin boyutlarını değiştiren bir işlevdir. Özellikle matrislerin transpozunu almak için kullanılır.

Bir matrisin transpozunu almak, matrisin satırlarını sütunlarına, sütunlarını da satırlarına dönüştürür.

**arr.transpose()** #satirlari sutunlara sutunlari satirlara koymus oldu.(3,2) olan bir seyi (2,3)'e cevirdi.

**Split**

1. Split var **split(2**) yazarsam mesela 6 elemanli bir arrayi 3 elemanli 2 array haline getiriyor. 6 elemanli bir array’i ancak katlarina bolebilirim
2. **Array.split** var bir de. Array.split(3) dersem mesela, 5 elemanli bir arrayi de bolebiliyoruz. Kendince 3 parcaya ayiriyor.

np.split(a,4) dedigimizde 4 esit parcaya bol

X bir array olsun mesela 1’den 9’a kadar giden

**Np.split(x,[2,6]**) bu da 0’dan 2.indekse kadar bol, yani 0. ve 1. indeksten bir arrayu

Daha sonra kalan kisimdan, yani 2. indeksten 6. indeksine git ordan da bol (2,3,4,5. indekslerden bir array)

Geri kalani da (6,7,8. indekskerden de) baska bir array olarak bize ver demek,

Yani 2 farkli yerden boldugumde, 3 farkli array olusturmus oluyor.

**np.split(x, [1,3])** #x arrayini al 1. indeksine kadar git demek, ordan bir array yapiyor

#daha sonra 1'den 3. indeksine kadar git onu bir array yap diyor.

#geri kalani da bir array yapiyor. 2 bolme yeri toplam 3 array yine.

**np.split(x, [1,3], axis=0)** #dersem de satirlar bazinda ayiriyor

**np.vsplit(x, [1,3])** #vertical split demek bu axis=0 gibi bazinda ayiriyor

**np.hsplit(x, [1,3]**) #horizontal split demek bu axis=1 islevi goruyor

**Numpy Indexing & Selection**

NumPy dizilerinde belirli bir elemana erişmek için, dizi adını takiben köşeli parantez içinde ilgili elemanın indeks numarasını belirtiriz.Örn; myarr[indexno]

İndeks numaraları 0'dan başlar, yani ilk elemanın indeksi 0'dır ve diğer elemanların indeks numaraları sırasıyla 1, 2, 3, gibi artar.

Ayrıca negatif indeksleme de kullanabiliriz. Negatif indeksleme, NumPy dizilerinde sondan başlayarak elemanlara erişmek için kullanılır. Son elemanın indeksi -1'dir ve önceki elemanların indeksleri sırayla -2, -3, gibi devam eder.

2 boyutlu NumPy dizilerinde, belirli bir elemana erişmek için satır ve sütun numaralarını kullanırız. Bu şekilde erişim için myarr[satırno, sütunno] formatı kullanılır.

3 boyutlu NumPy dizilerinde ise, elemana erişmek için derinlik, satır ve sütun numaralarını kullanırız. Yani erişim için myarr[derinlikno, satırno, sütunno] formatı kullanılır.

**Bracket Indexing & Selection**

Bir dizinin bir veya birkaç öğesini seçmenin en basit yolu, python listelerindeki gibi yapılır.

**Indeksleme**

Array’de **tek bir elemana** ulasmak icin indeksleri kullanirim.

arr[5] #5.indeksi getirir

arr[-1] #-1 son indeksi getirir

Start stop ve step degerleri verebiliyoruz.

2 boyutlu bir araryde [0,1] dedigimizde indeksinde, 0. satirin 1. sutununda yer alan indeks demektir.

[2,0] dedigimizde de 2. satirin 0. sutununda yer alan tek bir elemani getirdigimiz method

**Slicing**

**arr[0:5] #0'dan basla 5. indekse kadar olan elemanlari getir**

**arr[:5] #bu sekilde de gosterilir**

**Data[1:3]** dedigimizde 1.satirdan basla 3.satira kadar git demek bu, hepsini alir

**Data[0:2, 0]** dedigimizde ise, 0’dan basla 2’ye kadar git ama 0. columndaki degeri al demek

**Slicing 2-D array**

**#virgulun sol tarafina satirin indeksini, sag tarafa sutunun indeksini vererek yapiyoruz.**

**arr[0,1] #boyle iki deger yazarsak ikisinin kesisimini getirir**

**#sadece 0 yazsak ilk satirin hepsini getirecekti**

**Data[0:2, :0]** 0’dan 2’ye kadar git satirlarda, column olarak da 0. columnu al

**Data[2, :1]** 2. satira git, onun da 1. columundan basla sonuna kadar git

**Data[:2, 2:3]** satirlarda 0’dan basla 2’ye kadar git, 2.columundan basla, 3.columnu da al dur.

**Data[:,::2]** tum satirlari al, columnlarda da bastan sona 2’ser 2’ser git

**Data[::2, ::3]** 2’ser atlayarak tum satirlari, 3’er atlayarak tum sutunlari al demek

**Array icindeki elemani degistirme:**

Ornegin arr\_2d[1,1] #3'e 3 olan bir array'in ortasindaki eleman.

**arr\_2d[1,1]=100** #dersek debu elemani 100e donusturmus oluyoruz kolalikla

**Fancy Indexing**

Fancy Indexing, NumPy dizilerinde, bir veya daha fazla koşula veya belirtilen indeksler listesine göre dizideki belirli elemanlara erişmek için kullanılan bir indeksleme yöntemidir.

Bu yöntemde, köşeli parantez içinde bir liste veya dizi kullanarak, ilgili dizinin belirli elemanlarına erişebilirsiniz. Bu indeksleme yöntemi, normal indeksleme yöntemlerine göre daha esnek ve güçlüdür.

-Bana 6’dan kucukleri getir, 5 haric olanlari getir gibi spesifik olanlari cagirmamiz saglar

a[1],a[3],a[5] #bu ayri ayri olan indekleri yanyana koyunca bize tuple formatinda bunlari bize getirdi

#bunu array formtatinda olsun istersem de hepsini koseli paranteze alip cagiririm

[a[1],a[3],a[5]] seklinde

\*

a diye bir arrayim var diyelim.

**index\_list=[1,3,5]** #1.3.ve 5.indeksleri bir liste olarak tanimladim diyelim.

**a[index\_list]** # bunu da array'e cevirdim. yani a arrayinin,

#list halinde istedigim indekslerinden bir array olusturdu

\*

Mesela arr’in 2,4,6,8. indekslerini getirsin istiyoruz diyelim

arr[2,4,6,8] #boyle hata veriyor. 2 boyutlu oldugu icin, bunu [[]] 2 koseli parantezle cagirmam gerekir.

**arr[[2,4,6,8]]** #iki kere koseli paranteze alarak, yani boyutunu dikkate alarak,

#indeksini cagirmam gerek

\*

**arr[[1,2],[0,3]]** #1.indeksteki satirla 0. indeksteki sutunla carpistiriyor once

#sonra da 2. indeksteki satirla, 3. indeksteki sutunun kesisimini veriyor.

\*

**arr[1,[1,3]]** #bu da 1. satiri, 1. sutunla kesistir indeksi getir, sonra yine 1. satiri 3. sutunla kesistir indeksi getir demek

\*

**arr[[0,9],[1]]** #bu da 0.satiri 1. sutunla kesistir, sonra 9. satiri yine 1. sutunla kesistir demek

**arr[[0,-1],[1]]** #bu da ayni seyi verir usttekiyle

\*

**arr[1:3, [1,2]]** #satirlarda 1'den 3'e kadar git dedik, [1,2] dedik sutun icin,

#yani 1. ve 2. sutunu getir dedik

**arr[1:3, 1:3**] #yukaridaki ile ayni array i veriyor bu da slicing yontemi

\*

**arr[(arr !=3) & (arr!=4)]** #bu da 3'e ve 4'e esit olmayanlari getir demek