

▼ NumPy (Numerical Python)

- NumPy, dizilerle çalışmak için kullanılan bir Python kütüphanesidir
- NumPy, doğrusal cebir, fourier dönüşümü ve matrisler alanında çalışmak için de gerekli işlevlere sahiptir

Neden NumPy Kullanılır:

- NumPy, geleneksel Python listelerinden 50 kat a kadar daha hızlı bir dizi nesnesi sağlamayı amaçlamaktadır.

NumPy'yi Projeye Dahil Etmek

- import anahtar sözcüğünü ekleyerek onu uygulamalarınıza dahil edebilirsiniz

NumPy np Olarak Kullanmak

- NumPy genellikle np alias (takma adı) altında içe aktarılır
- alias Python'da, aynı şeye atıfta bulunmak için alternatif bir addır
- içe aktarırken as anahtar sözcüğüyle bir alias oluşturabilirsiniz

```
import numpy as np
```

▼ NumPy ndarray Nesnesi Oluşturmak

- NumPy'deki dizi nesnesine ndarray denir
- array() işlevini kullanarak bir NumPy ndarray nesnesi oluşturabiliriz

```
a = np.array([1,2,3])
```

```
a
```

```
array([1, 2, 3])
```

```
type(a)
```

```
numpy.ndarray
```

▼ dir() fonksiyonu

- nesnelerin özellikleri hakkında bilgi edinme imkanı sağlar

```
dir(a)
```

```
['T',  
 '__abs__',  
 ...]
```

```
'__add__',
'__and__',
'__array__',
'__array_finalize__',
'__array_function__',
'__array_interface__',
'__array_prepare__',
'__array_priority__',
'__array_struct__',
'__array_ufunc__',
'__array_wrap__',
'__bool__',
'__class__',
'__complex__',
'__contains__',
'__copy__',
'__deepcopy__',
'__delattr__',
'__delitem__',
'__dir__',
'__divmod__',
'__doc__',
'__eq__',
'__float__',
'__floordiv__',
'__format__',
'__ge__',
'__getattr__',
'__getitem__',
'__gt__',
'__hash__',
'__iadd__',
'__iand__',
'__ifloordiv__',
'__ilshift__',
'__imatmul__',
'__imod__',
'__imul__',
'__index__',
'__init__',
'__init_subclass__',
'__int__',
'__invert__',
'__ior__',
'__ipow__',
'__irshift__',
'__isub__',
'__iter__',
'__itruediv__',
'__ixor__',
'__le__',
'__len__',
'__lshift__',
'__lt__',
'__matmul__',
'__mod__'
```

▼ 2-D Diziler

- Öğeleri olarak 1 boyutlu dizilere sahip bir diziye 2 boyutlu dizi denir

```
b = np.array([[1,2,3],[4,5,6]])
```

```
b
```

```
array([[1, 2, 3],  
       [4, 5, 6]])
```

▼ bir dizinin shape'ini almak

- Bir dizinin şekli (shape), dizinin her boyutundaki (dimension, örneğin 1D, 2D gibi) elemanların sayısıdır

```
a.shape
```

```
(3,)
```

▼ ndim fonksiyonu

- bir dizinin boyutunu öğrenmemizi sağlar

```
a.ndim
```

```
1
```

```
b.ndim
```

```
2
```

```
b.shape
```

```
(2, 3)
```

```
np.array(range(10))
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
np.array([i**2 for i in range (10)])
```

```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

```
c = np.array([1,0.1, 0.2,5])
```

▼ dtype fonksiyonu

- NumPy array nesnesinin hangi veri tipinde olduğunu döndürür

```
c.dtype
```

```
dtype('float64')
```

```
a.dtype
```

```
dtype('int32')
```

```
d=np.array([1,2,3.0,"deneme"])
```

```
d.dtype
```

```
dtype('<U32')
```

```
d
```

```
array(['1', '2', '3.0', 'deneme'], dtype='<U32')
```

▼ bir diziyi matris olarak yazdırma örneği

```
for i in b:  
    for j in i:  
        print(j, end=" ")  
    print()
```

```
1 2 3  
4 5 6
```

▼ np.zeros() metodu

- np.zeros() bir tuple (demet) değeri alır
- Bu tuple değeri, oluşturmak istediğimiz dizinin boyutlarının değerleridir
- np.zeros() ise bu boyutlarda ve sıfırlardan oluşan bir dizin üretir

```
np.zeros(5)
```

```
array([0., 0., 0., 0., 0.])
```

▼ np.ones() metodu

- np.ones() metodu da np.zeros() mantığında çalışır ve girilen boyutlarda oluşturduğu dizini 1'lerle doldurur

```
np.ones(5)
```

```
array([1., 1., 1., 1., 1.])
```

```
np.ones((2,3))
```

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

```
np.ones((2,3), dtype = int)
```

```
array([[1, 1, 1],
       [1, 1, 1]])
```

```
np.arange(5,10,2)
```

```
array([5, 7, 9])
```

```
np.array(range(5))
```

```
array([0, 1, 2, 3, 4])
```

▼ np.linspace() metodu

- başlıca 3 parametre alır
- Başlangıç, dizinin hangi sayıdan başlayacağıdır
- Bitiş, dizinin hangi sayıya geldiğinde son bulacağıdır
- Bir de num, dizinin kaç elemana sahip olacağıdır
- metod başlangıçtan sona num tane sayıyı birbiri arası uzaklık eşit olacak şekilde böler

```
np.linspace(0.0,1.0, num =10)
```

```
array([0.          , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
       0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.          ])
```

```
b
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

▼ reshape metodu

- Reshape (yeniden şekillendirme), bir dizinin şeklini (shape) değiştirmek anlamına gelir
- diziye yeni boyut ekler veya mevcut boyutları kaldırır ya da her boyuttaki eleman sayısını değiştirir

```
b.reshape(3,2)
```

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
b.resize(3,2)
```

▼ numPy dizisinde aritmetiksel işlemler

```
b*3
```

```
array([[ 3,  6],  
       [ 9, 12],  
       [15, 18]])
```

```
a+6
```

```
array([7, 8, 9])
```

```
d.dtype
```

```
dtype('<U32')
```

▼ full() metodu

- istediğimiz boyutta hızlıca dizi oluşturmak için kullanılır

```
np.full(10,5)
```

```
array([5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
```

▼ eye() metodu

- hızlıca matris oluşturmak için kullanılan bir metod

```
np.eye(5)
```

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

```
np.eye(5,4)
```

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

▼ random sınıfından kullanım örnekleri :

```
np.random.random((2,5))
```

```
array([[0.74290587, 0.99777759, 0.23576608, 0.15274286, 0.44434514],  
       [0.36004765, 0.82106138, 0.61236667, 0.51314624, 0.24408434]])
```

```
x = np.random.randint(10,size = (3,4))
```

```
x
```

```
array([[5, 9, 4, 0],  
       [7, 5, 2, 9],  
       [7, 6, 3, 5]])
```

▼ transpose() metodu

- Bir matrisin satır ve stünlarını yer değıştirmek için kullanılır

```
y = x.transpose()
```

```
y
```

```
array([[5, 7, 7],  
       [9, 5, 6],  
       [4, 2, 3],  
       [0, 9, 5]])
```

▼ transpose() metodu bir shallow copy örneğidir

- aşağıdaki satır bu durumu kanıtlar

```
x[0][0] = 100
```

```
y
```

```
array([[100, 7, 7],  
       [ 9, 5, 6],  
       [ 4, 2, 3],  
       [ 0, 9, 5]])
```

▼ copy() metodu deep copy yapmamıza olanak sağlar

```
y= x.copy().transpose()
```

```
x[0][0] = 10
```

```
y
```

```
array([[100, 7, 7],
       [ 9, 5, 6],
       [ 4, 2, 3],
       [ 0, 9, 5]])
```

x

```
array([[10, 9, 4, 0],
       [ 7, 5, 2, 9],
       [ 7, 6, 3, 5]])
```

```
z = np.random.randint(10,size = (3,4))
```

z

```
array([[9, 4, 5, 3],
       [9, 9, 8, 7],
       [2, 8, 2, 6]])
```

x

```
array([[6, 3, 7, 2],
       [6, 6, 7, 8],
       [1, 1, 7, 5]])
```

▼ matrisin bir matrisle aritmetik işleme tabii tutulması

x+z

```
array([[15, 7, 12, 5],
       [15, 15, 15, 15],
       [ 3, 9, 9, 11]])
```

x*z

```
array([[54, 12, 35, 6],
       [54, 54, 56, 56],
       [ 2, 8, 14, 30]])
```

▼ multiply() metodu

- bir dizideki değerleri başka bir dizideki değerlerle çarpar ve sonuçları yeni bir dizide döndürür.

```
np.multiply(x,z)
```

```
array([[54, 12, 35, 6],
       [54, 54, 56, 56],
       [ 2, 8, 14, 30]])
```


▼ dot() metodu

- dot() metodu sonuç olarak aldığı iki Numpy dizisinin nokta çarpımı ya da bir diğer adıyla skaler çarpımını döndürür

```
np.dot(x,y)
```

```
array([[ 98, 119,  68],
       [119, 185, 101],
       [ 68, 101,  76]])
```

```
a = np.array([1,2,3])
b = np.array([4,5,6])
```

```
np.dot(a,b)
```

```
32
```

```
a = np.array([[1,2,3], [4,5,6]])
b = np.array([4,5,6])
```

```
np.dot(a,b)
```

```
array([32, 77])
```

```
np.multiply(a,b)
```

```
array([[ 4, 10, 18],
       [16, 25, 36]])
```

```
a = np.array([[1,2,3], [4,5,6]])
b = np.array([[7,8,9], [4,5,6]])
```

```
np.dot(a,b.transpose())
```

```
array([[ 50,  32],
       [122,  77]])
```

```
a.shape
```

```
(2, 3)
```

```
b.transpose().shape
```

```
(3, 2)
```

```
np.dot(a.transpose(),b)
```

```
array([[23, 28, 33],
       [34, 41, 48],
       [45, 54, 63]])
```

▼ multiply() ve dot() kullanım farkı

```
np.dot(a,b)
```

```
-----
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_22952\1223026733.py in <module>
----> 1 np.dot(a,b)

<__array_function__ internals> in dot(*args, **kwargs)

ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
```

SEARCH STACK OVERFLOW

```
np.multiply(a,b)
```

```
array([[ 7, 16, 27],
       [16, 25, 36]])
```

```
a=np.array(["1","2","3"])
```

```
a
```

```
array(['1', '2', '3'], dtype='<U1')
```

```
a+1
```

```
-----
UFuncTypeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_10044\4093587236.py in <module>
----> 1 a+1

UFuncTypeError: ufunc 'add' did not contain a loop with signature matching types
(dtype('<U1'), dtype('int32')) -> None
```

SEARCH STACK OVERFLOW

▼ astype() metodu

- bir numPy sınıfının türünü deüüştirme olanağı sağlar

```
b=a.astype(np.int32)
```

```
a
```

```
array([6])
```

```
b+1
```

```
array([7])
```

```
b=np.array(range(10))
```

```
b
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
b[5]
```

```
5
```

▼ slicing örneği

```
b[2:6]
```

```
array([2, 3, 4, 5])
```

```
a=b[2:6] #view shallow copy
```

```
a
```

```
array([2, 3, 4, 5])
```

```
a[2]=100
```

```
a
```

```
array([ 2,  3, 100,  5])
```

```
b
```

```
array([ 0,  1,  2,  3, 100,  5,  6,  7,  8,  9])
```

```
a=b[2:6].copy() #deep copy
```

```
a[2]=-99
```

```
b
```

```
array([ 0,  1,  2,  3, 100,  5,  6,  7,  8,  9])
```

```
a
```

```
array([ 2,  3, -99,  5])
```

▼ absolute() metodu

- dizinin elemanlarının mutlak değerlerini döndürür

```
np.absolute(a)
```

```
array([ 2,  3, 99,  5])
```

```
x= np.array([True, True, False,0,42])
```

```
x.dtype
```

```
dtype('int32')
```

▼ logical_not(x) metodu

- dizinin elemanlarının mantıksal tersini döndürür

```
np.logical_not(x)
```

```
array([False, False,  True,  True, False])
```

▼ fibonacci fonksiyonu

```
def fib (x):  
    if x == 1 or x ==2:  
        return 1  
    a,b = 1, 1  
    for i in range(x-2):  
        a,b = b, a+b  
    return b
```

▼ fibonacci fonksiyonu'nun vektörize hale getirilmesi

```
vfonk =np.vectorize(fib)
```

```
vfonk
```

```
<numpy.vectorize at 0x21d5a9a01f0>
```

```
a=np.array(range(1,13))
```

```
vfonk(a)
```

```
array([ 1,  1,  2,  3,  5,  8, 13, 21, 34, 55, 89, 144])
```

```
a
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

▼ vektörize fonksiyonun lamda fonksiyonu ile kullanım örneği

```
vfonk2=np.vectorize(lambda x: len(x)>3)
```

```
deneme=np.array(["ali","ahmet","mehmet","ayşe"])
```

```
vfonk2(deneme)
```

```
array([False,  True,  True,  True])
```

```
x=np.random.randint(10,100, size=(4,6))
```

```
x
```

```
array([[34, 55, 17, 66, 34, 60],  
       [39, 82, 35, 23, 73, 31],  
       [10, 36, 58, 54, 57, 72],  
       [74, 77, 30, 50, 49, 84]])
```

```
y= (x>30)
```

```
y
```

```
array([[ True,  True, False,  True,  True,  True],  
       [ True,  True,  True, False,  True,  True],  
       [False,  True,  True,  True,  True,  True],  
       [ True,  True, False,  True,  True,  True]])
```

```
x[y]
```

```
array([34, 55, 66, 34, 60, 39, 82, 35, 73, 31, 36, 58, 54, 57, 72, 74, 77,  
       50, 49, 84])
```

```
x
```

```
array([[34, 55, 17, 66, 34, 60],  
       [39, 82, 35, 23, 73, 31],  
       [10, 36, 58, 54, 57, 72],  
       [74, 77, 30, 50, 49, 84]])
```

```
x
```

```
array([[34, 55, 17, 66, 34, 60],  
       [39, 82, 35, 23, 73, 31],  
       [10, 36, 58, 54, 57, 72],  
       [74, 77, 30, 50, 49, 84]])
```

▼ sum() metodu

- dizi içindeki değerlerin toplamını döndürür

```
x.sum()
```

```
1200
```

▼ mean() metodu

- dizinin değerlerinin ortalamasını döndürür

```
x.mean()
```

```
50.0
```

▼ std() metodu

- Standart sapma bir dizindeki verilerin ortalamaya göre dağılımının sayısal olarak göst erimidir
- dizinin standart sapmasını bulur

```
x.std()
```

```
20.657928260113597
```

▼ median() metodu

- Bir dizinin medyanı o dizini sıraladığımızda tam ortasına denk gelen elemana denir
- dizinin medyanını bulur

```
np.median(x)
```

```
52.0
```

▼ min() metodu

- dizideki en küçük elemanı döndürür

```
x.min()
```

```
10
```

▼ max() metodu

- dizideki en büyük elemanı döndürür

```
x.max()
```

```
84
```

▼ cumsum() metodu

- verilen eksen boyunca dizi öğelerinin kümülatif toplamını döndürmek için kullanılır
- Çıktı belirtilmedikçe sonucu tutan yeni bir dizi döndürür

```
x.cumsum()
```

```
array([ 34, 89, 106, 172, 206, 266, 305, 387, 422, 445, 518,
        549, 559, 595, 653, 707, 764, 836, 910, 987, 1017, 1067,
        1116, 1200], dtype=int32)
```

▼ cumprod() metodu

- Belirli bir eksen boyunca öğelerin kümülatif çarpımını döndürür
- Çıktı belirtilmedikçe sonucu tutan yeni bir dizi döndürür

```
x.cumprod()
```

```
array([ 34, 1870, 31790, 2098140, 71336760,
        -14761696, -575706144, 36736448, 1285775680, -491930432,
        -1551183168, -842037952, 169555072, 1809015296, 1843672064,
        774043648, 1170814976, -1600667648, 1809678336, 1906278400,
        1353777152, -1030619136, 1039269888, 1399324672], dtype=int32)
```

```
x
```

```
array([[34, 55, 17, 66, 34, 60],
       [39, 82, 35, 23, 73, 31],
       [10, 36, 58, 54, 57, 72],
       [74, 77, 30, 50, 49, 84]])
```

▼ Dosya kayıt etme ve okuma metot örnekleri

```
file ="deneme.npy"
np.save(file,x)
```

```
c=np.load(file)
```

```
c
array([[34, 55, 17, 66, 34, 60],
       [39, 82, 35, 23, 73, 31],
       [10, 36, 58, 54, 57, 72],
       [74, 77, 30, 50, 49, 84]])
```

```
import random
```

▼ numPy'in normal listelere göre daha verimli olduğunun kanıtı

```
%timeit -n3 -r2 l = [random.randrange(1,7) for i in range(0, 6000000)]
```

3.61 s ± 4.06 ms per loop (mean ± std. dev. of 2 runs, 3 loops each)

```
%timeit -n3 -r2 n = np.random.randint (1,7, 6000000)
```

46.6 ms ± 509 µs per loop (mean ± std. dev. of 2 runs, 3 loops each)

▼ PANDAS

Pandas Veri Yapıları

- Pandas'da verileri kolay analiz etmek için bazı veri yapıları vardır. Bunlardan en çok kullanılanları Series ve DataFrame veri yapılarıdır
 - Series veri yapısı bir boyutludur yani bir sütundan oluşur
 - DataFrame veri yapısı iki boyutludur yani satırlar ve sütunlardan oluşur
 - Pandas'ı import ettikten sonra Pandas'ı pd kısaltması ile kullanmak için:

```
import pandas as pd
```

▼ read_csv metodu

- veri setini import eder

```
pd.read_csv("Ann.csv")
```



```

-----
AttributeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_22952\196015359.py in <module>
----> 1 pd.read_csv("Ann.csv")

C:\ANACONDA\lib\site-packages\pandas\__init__.py in __getattr__(name)
    259         return _SparseArray
    260
--> 261         raise AttributeError(f"module 'pandas' has no attribute '{name}'")
    262
    263

AttributeError: module 'pandas' has no attribute 'read_csv'

```

▼ Seri tanımlama örneği

```
a=pd.Series([87,10,20])
```

```
a
```

```

0    87
1    10
2    20
dtype: int64

```

- uzunluğu 5 olan seri tanımlama

```
b=pd.Series([10, range(5)])
```

- a serisinin ilk indexindeki elemana erişme

```
a[0]
```

```
87
```

▼ count() methodu

- Serinin eleman sayısını döndürür

```
a.count()
```

```
3
```

▼ mean() methodu

- Serinin ort almasını döndürür

```
a.mean()
```

```
39.0
```

▼ describe() methodu

- metinsel olarak istatistiki sonuç döner

```
a.describe()
```

```
count      3.000000
mean       39.000000
std        41.868843
min        10.000000
25%        15.000000
50%        20.000000
75%        53.500000
max        87.000000
dtype: float64
```

▼ key value pair örnekleri

```
c=pd.Series([1,2,3], index=["A","B","C"])
```

```
c
```

```
A      1
B      2
C      3
dtype: int64
```

```
d=pd.Series({"A":1,"B":2,"C":3})
```

```
d
```

```
A      1
B      2
C      3
dtype: int64
```

```
d[0]
```

```
1
```

```
d["A"]
```

```
1
```

```
d.dtype
```

```
dtype('int64')
```

▼ values fonksiyonu

- d serisindeki değerleri döndürür

```
d.values
```

```
array([1, 2, 3], dtype=int64)
```

▼ DataFrame

- DataFrame'in her bir sütunu Series veri yapısındadır

```
x = {"A": [1, 2, 3], "B": [4, 5, 6], "C": [7, 8, 9], }
```

```
#d = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6], "C": [7, 8, 9], })  
d = pd.DataFrame(x)
```

```
d
```

	A	B	C
0	1	4	7
1	2	5	8
2	3	6	9

▼ index anahtar kelimesi ile satırlara index atamış olduk

```
d.index=["t1","t2","t3"]
```

```
d
```

	A	B	C
t1	1	4	7
t2	2	5	8
t3	3	6	9

▼ Sütundan veri çekme işlemi

```
d["A"]
```

```
t1    1
t2    2
t3    3
Name: A, dtype: int64
```

```
d.A
```

```
t1    1
t2    2
t3    3
Name: A, dtype: int64
```

▼ loc[] fonksiyonu sayesinde satırdan veri çekme işlemi

```
d.loc["t3"]
```

```
A    3
B    6
C    9
Name: t3, dtype: int64
```

▼ iloc[] fonksiyonu sayesinde satırdan integer değer ile veri çekme işlemi

```
d.iloc[2]
```

```
A    3
B    6
C    9
Name: t3, dtype: int64
```

▼ loc[] ve iloc[] fonksiyonunda slicing örnekleri

```
d.loc["t1":"t3"]
```

	A	B	C
t1	1	4	7
t2	2	5	8
t3	3	6	9

```
d.iloc[0:3]
```

	A	B	C
--	---	---	---

t1	1	4	7
----	---	---	---

```
d.iloc[[0,2]]
```

	A	B	C
--	---	---	---

t1	1	4	7
----	---	---	---

t3	3	6	9
----	---	---	---

```
d.iloc[[0,2], [0,2]]
```

	A	C
--	---	---

t1	1	7
----	---	---

t3	3	9
----	---	---

```
d.iloc[0:2,0:2]
```

	A	B
--	---	---

t1	1	4
----	---	---

t2	2	5
----	---	---

```
d.iloc[:2,:2]
```

	A	B
--	---	---

t1	1	4
----	---	---

t2	2	5
----	---	---

▼ DataFrame'de mantıksal operatörlerin kullanım örnekleri

```
d[d>5]
```

	A	B	C
--	---	---	---

t1	NaN	NaN	7
----	-----	-----	---

t2	NaN	NaN	8
----	-----	-----	---

t3	NaN	6.0	9
----	-----	-----	---

```
d[(d>5) & (d<8)]
```

	A	B	C
t1	NaN	NaN	7.0
t2	NaN	NaN	NaN

▼ iat[] fonksiyonu

- Bir DataFrame veya Series'de integer kullanarak yalnızca tek bir değer almanız veya ayarlamanız gerekiyorsa iat'ı kullanılır

```
d.iat[0,2]
```

```
7
```

▼ at[] fonksiyonu

- Bir DataFrame veya Series'de yalnızca tek bir değer almanız veya ayarlamanız gerekiyorsa iat'ı kullanılır

```
d.at["t1","C"]=100
```

```
d
```

	A	B	C
t1	1	4	100
t2	2	5	8
t3	3	6	9

▼ axis anahtar kelimesi

- ulaşmak istediğimiz değerler kümesi satırlar ise axis'i bir'e eşitleriz
- ulaşmak istediğimiz değerler kümesi stünlar ise axis'i sıfır'a eşitleriz

```
d.mean(axis=1)
```

```
t1    4.0
t2    5.0
t3    6.0
dtype: float64
```

```
d.describe()
```

	A	B	C
count	3.0	3.0	3.0
mean	2.0	5.0	8.0
std	1.0	1.0	1.0
min	1.0	4.0	7.0
25%	1.5	4.5	7.5
50%	2.0	5.0	8.0
75%	2.5	5.5	8.5

```
y = {"A":[1,2,3,4,5,6.25689,7,8,9],"B":[4,5,6,7,8,9.2525,1,2,3],"C":[7,8,9,1,2,3.25252,4,5,6],}
```

```
veri=pd.DataFrame(y)
```

▼ head metodu

- veri setinin ilk 5 satırını gösterir

```
veri.head()
```

	A	B	C
0	1.0	4.0	7.0
1	2.0	5.0	8.0
2	3.0	6.0	9.0
3	4.0	7.0	1.0
4	5.0	8.0	2.0

```
veri.describe()
```

	A	B	C
count	9.000000	9.000000	9.000000
mean	5.028543	5.028056	5.028058
std	2.751646	2.785603	2.716767
min	1.000000	1.000000	1.000000
25%	3.000000	3.000000	3.252520
50%	5.000000	5.000000	5.000000
75%	7.000000	7.000000	7.000000
max	9.000000	9.252500	9.000000

▼ set_option() metodu

- veriyi formatlamayı sağlar

```
pd.set_option("precision", 3)
```

```
-----
OptionError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_22952\1981155718.py in <module>
----> 1 pd.set_option("precision", 3)

C:\ANACONDA\lib\site-packages\pandas\_config\config.py in __call__(self, *args, **kwargs)
    254
    255     def __call__(self, *args, **kwargs):
--> 256         return self.__func__(*args, **kwargs)
    257
    258     @property

C:\ANACONDA\lib\site-packages\pandas\_config\config.py in _set_option(*args, **kwargs)
    147
    148     for k, v in zip(args[::2], args[1::2]):
--> 149         key = _get_single_key(k, silent)
    150
    151         o = _get_registered_option(key)

C:\ANACONDA\lib\site-packages\pandas\_config\config.py in _get_single_key(pat, silent)
    114         raise OptionError(f"No such keys(s): {repr(pat)}")
    115     if len(keys) > 1:
--> 116         raise OptionError("Pattern matched multiple keys")
    117     key = keys[0]
    118

OptionError: 'Pattern matched multiple keys'
```

SEARCH STACK OVERFLOW

```
veri.describe()
```

	A	B	C
count	9.000000	9.000000	9.000000
mean	5.000000	5.000000	5.000000
std	2.738613	2.738613	2.738613
min	1.000000	1.000000	1.000000
25%	3.000000	3.000000	3.000000
50%	5.000000	5.000000	5.000000
75%	7.000000	7.000000	7.000000
max	9.000000	9.000000	9.000000


```
veri.mean()
```

```
A    5.0  
B    5.0  
C    5.0  
dtype: float64
```

▼ Her bir satırın ortalama değerinin bulunması

```
veri.mean(axis=1)
```

```
0    4.0  
1    5.0  
2    6.0  
3    4.0  
4    5.0  
5    6.0  
6    4.0  
7    5.0  
8    6.0  
dtype: float64
```

▼ Verinin Transpose edilmesi örneği

```
veri.T
```

```
0  1  2  3  4  5  6  7  8
```

```
A  1  2  3  4  5  6  7  8  9
```

```
B  4  5  6  7  8  9  1  2  3
```

```
C  7  8  9  1  2  3  4  5  6
```

```
veri.T.describe()
```

```
0  1  2  3  4  5  6  7  8
```

```
count  3.0  3.0  3.0  3.0  3.0  3.0  3.0  3.0  3.0
```

```
mean   4.0  5.0  6.0  4.0  5.0  6.0  4.0  5.0  6.0
```

```
std     3.0  3.0  3.0  3.0  3.0  3.0  3.0  3.0  3.0
```

```
min     1.0  2.0  3.0  1.0  2.0  3.0  1.0  2.0  3.0
```

```
25%     2.5  3.5  4.5  2.5  3.5  4.5  2.5  3.5  4.5
```

```
50%     4.0  5.0  6.0  4.0  5.0  6.0  4.0  5.0  6.0
```

```
75%     5.5  6.5  7.5  5.5  6.5  7.5  5.5  6.5  7.5
```

```
max     7.0  8.0  9.0  7.0  8.0  9.0  7.0  8.0  9.0
```

```
veri
```

	A	B	C
0	1	4	7
1	2	5	8
2	3	6	9
3	4	7	1
4	5	8	2
5	6	9	3
6	7	1	4
7	8	2	5
8	9	3	6

▼ columns fonksiyonu

- verinin kolonlarını liste olarak döndürür

```
veri.columns
```

```
Index(['A', 'B', 'C'], dtype='object')
```

▼ to_numpy() fonksiyonu

- Veriyi numPy dizisine dönüştürür

```
veri2=veri.to_numpy()
```

```
veri2
```

```
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9],
       [4, 7, 1],
       [5, 8, 2],
       [6, 9, 3],
       [7, 1, 4],
       [8, 2, 5],
       [9, 3, 6]], dtype=int64)
```

```
veri2.shape
```

```
(9, 3)
```

```
veri2.ndim
```

▼ deep copy örneği

```
x=veri.copy
```

```
x
```

```
<bound method NDFrame.copy of      A  B  C  
0  1  4  7  
1  2  5  8  
2  3  6  9  
3  4  7  1  
4  5  8  2  
5  6  9  3  
6  7  1  4  
7  8  2  5  
8  9  3  6>
```