VERİ BİLİMİ İÇİN PYTHON VE TENSORFLOW

BEGUM UNER

Istanbul University



1. Python Temelleri
2. Obje Odaklı Programlama
3. Numpy
4. Pandas
5. Matlotlib
6. Tensorflow

**VERİ BİLİMİ İÇİN PYTHON VE TENSORFLOW**

1. **PYTHON TEMELLERİ**

* Pycharm, python için oluşturulmuş IDE programıdır. Pycharm kullanacağım.
  1. **Değişkenler (Variable)**

Bir veriyi içerisinde depolayan birime değişken denir. Değişkeni kutuya benzetebilirsiniz. Siz ona bir değer verirsiniz. O da verdiğiniz değeri sizin için saklar. Değişken isimleri sayı ile başlayamaz ve program içinde bulunan herhangi bir komut değişken ismi olarak atanamaz.

Değişken tanımlarken adını ve değerini yazmak yeterlidir.

tamSayi = 5  
tamSayi2 = 3  
  
degisken = tamSayi + tamSayi2  
print(degisken)

Python, varsayılan olarak yani yerleşik olarak aşağıdaki veri tiplerini barındırır:

* **String (metin) veri tipleri →**str
* **Numerik (sayısal) veri tipleri →**int, float, complex
* **Sequence (sıralama) veri tipleri →**list, tuple, range
* **Mapping (haritalama) veri tipleri →**dict
* **Set veri tipleri →**set, frozenset
* **Boolean veri tipleri →**bool
* **Binary veri tipleri →**bytes, bytearray, memoryview
  1. **Numerik (sayısal) veri tipleri**
* **Int** veya tamsayı veri tipi, sınırsız uzunlukta pozitif veya negatif tam sayıları ifade etmektedir.
* **Float** veya ondalıklı sayı veri tipi, noktadan sonra bir veya daha fazla ondalık sayı içeren pozitif veya negatif sayıları ifade etmektedir. Float, ayrıca içinde e veya E barındıran tüm bilimsel sayıları tanımlarken kullanılır.
* **Complex sayılar** daha çok ileri düzey matematiksel işlemlerde kullanılmaktadır.

x = 57 #int  
y = 57.5 #float  
z = 57j # complex

* 1. **String veri tipleri**
* Python’daki string değerler tek tırnak işareti veya çift tırnak işareti içinde ifade edilmektedir. Diğer bir ifade ile ‘Merhaba’ ve “Merhaba” Python için aynıdır.
* Python’daki string değerler de unicode karakterleri temsil eden bayt dizileridir. Bununla birlikte, Python’un karakter (char) veri tipi yoktur, tek bir karakter sadece 1 uzunluğunda bir stringtir.
* Bir fazla satırda ifade edilecek bir string değer varsa değer üç çift tırnak (“””) veya üç tek tırnak (‘’’) içine alınır.

a = "Merhaba!" #str  
  
s = """naber nasilsin nasil gidiyor?"""

* 1. **Sequence (sıralama) veri tipleri**
* **List**, herhangi bir sayıda diğer objeleri içinde bulunduran bir sandık vazifesi görüyor.

**[ ]** parantezler kullanılarak yazılır.

* **Tuple** sıralı ve **değiştirilemez** bir koleksiyon çeşididir. **()** parantezler kullanılarak yazılır.
* *Range* isimli bir yapı oluşturur ve bu yapı listelere oldukça benzer. Bu yapı başlangıç, bitiş ve opsiyonel olarak artırma değeri alarak listelere benzeyen bir sayı dizisi oluşturur.

List ile Tuple farkı

* Tuple yapısına öğe eklenemez; list yapısındaki ***append()***metodu yoktur.
* Tuple yapısında öğe atılamaz; list yapısındaki ***remove()*** ve ***pop()***metotları yoktur.
* Tuple içinde bir öğe aranamaz; list yapısındaki ***index()***metodu yoktur.
* Döngüler tuple’da **list yapısına göre daha hızlıdır.**
* Değişmeyecek veriler için tuple yapısı list yapısına göre daha uygun seçimdir.
* Değişmezliği nedeniyle tuple, dictionary yapısı için anahtar kelimeler olarak kullanılabilir.
* Tuple string dönüşümü için kullanabilir.
* Tuple yapısından list yapısına ve list yapısından tuple yapısına rahatlıkla ***dönüşüm***yapılabilir.

e = ["begum","ali"] #list  
  
f = ("begum","ali") #tuple  
  
g = range(54) #range 0 DAN 54 E KADAR YAZAR

* 1. **Mapping (haritalama) veri tipleri**
* Sözlükler de Tuple ve List veri türleri gibi farklı veri türleri bulunduran bulundan mutable(değiştirilebilir) veri türü olduğunu ve süslü parantezlerle gösterildiğini söylemiştik.
* Fakat sözlükler biraz farklıdır çünkü Sözlükler iki kısımdan oluşur; keys(anahtar) ve value(değer), value kısmı bütün veri türünü içerebilir fakat keys kısmı sadece string ve int tipinde olabilir.

h = {"ad" : "begum", "yas" : 15} #dict

* 1. **Set veri tipleri**
* Liste, sözlük ve demet veri türü gibi birden çok veri türünü birlikte barındıran veri tipidir. Kümeler ile ilgili yaptığınız her türlü işlevi (birleşme, kesişim vs.) bu veri tipi ile yapabilirsiniz.
* Set (Küme) aynı Sözlükler gibi **sıralı veri tipi değildir**. Bu yüzden indexleme desteklemez.
* Frozenset veri tipi de set veri turunun kısıtlanmış halidir. Peki neden kısıtlanış onu size açıklayayım biz Python da veri tiplerini mutable(değiştirilebilir) ve immutable(değiştirilmez) olarak ayırmıştık işte bu veri tipide immutable(değiştirilemez) veri tipidir. O yüzden kısıtlanmış küme diyoruz biz bu veri türüne ekleme, silme, değiştirme vb. işlemler yapamıyoruz.

i = {"erkan", "begum"} #set  
  
j = frozenset({"erkan", "begum"}) #frozenset

* 1. **Boolean veri tipleri**
* Boolean tanımlanan değişkenler iki değerden birini döndürür: True (Doğru) veya False (Yanlış).

k = True #bool

* 1. **Binary veri tipleri**

l = b"MERHABA" #bytes  
  
m = bytearray(2) #bytearray  
  
n = memoryview(bytes(2)) #memoryview

* 1. **Kullanıcıdan veri almak**

* **İnput** ile kullanıcıdan veri alınır **print** ile ekrana yazdırılır.
* **\n** yeni satir anlamına gelir

yas = input("yaşını gir")  
print(yas)

* 1. **İF – Else**
* If else i kullanarak kodumuzda belirli can alıcı kontroller yazabilir ve bu kontrollere göre belirli işlemler yaptırabiliriz

if x < y:  
 print "x, y'den küçüktür"  
elif x > y:  
 print "x, y'den büyüktür"  
else:  
 print "x ve y birbirine eşittir"

if x == y:  
 print  
 "x ve y birbirine eşittir"  
else:  
 if x < y:  
 print  
 "x, y'den küçüktür"  
 else:  
 print  
 "x, y'den büyüktür"

* 1. **For Döngüsü**
* Python ’da iki tane döngü bulunur: while ve for
* Liste, dize, çokuz gibi sıralı bir nesnedeki elemanları sırayla tek tek alır ve döngü blokunda işler.

for s in ["fındık", "fıstık", "ceviz"]:  
 print(s)

for x in range(11):  
 print(x, x\*\*2)

* 1. **While Döngüsü**
* While döngüleri, bir mantıksal şart mantıksal “doğru” değerine sahip olduğu sürece tekrarlanır. Döngünün sona ermesi için şartın eninde sonunda yanlış hale gelmesi gerekir. Bunun için döngünün içinde değişkenler uygun şekilde güncellenir. Bu güncellemeyi unutursak sonsuz döngü içine düşeriz ve programımız biz zorla kapatmadıkça durmaz.

a=1  
b=10  
while a<b:  
 print(a, end=" ")  
 a += 1 # a = a + 1 ile aynı

* 1. **Break: Döngüyü Bitirmek**
* Bir döngü bloku içinde verilen bir break komutu, döngünün hemen o anda bitirilmesine yol açar. Program akışı döngü blokunun dışına atlar ve ardından gelen komutlarla devam eder. Örnek olarak, kullanıcının verdiği sayıları toplayan bir program yazalım. Kullanıcı -99 girene kadar sayıları toplama eklemeye devam etsin.

while True: # Şart hep doğru - sonsuz döngü  
 x = int(input("Bir sayı girin (bitirmek için -99): "))  
 if x == -99:  
 break  
 toplam += x  
print("Toplam:", toplam)

* 1. **Continue: Döngünün kalanını atlamak**
* continue komutu da, aynı break gibi, sadece bir döngü içinde ve bir if şartı altında mânâ ifade eder. continue döngü blokunun işlemesini yarıda keser ve başa döner. break‘den farkı, programın döngünün dışına çıkmaması, ama döngünün başına dönmesi ve tekrar başlatmasıdır. Bu arada döngü şartının doğru olup olmadığı da kontrol edilir.

toplam = 0  
while True:  
 x = int(input("Bir sayı girin (bitirmek için -99): "))  
 if x == -99:  
 break  
 if x < 0 or x > 100:  
 print("0-100 arası olmalı.")  
 continue  
 toplam += x  
print("Toplam:", toplam)

* 1. **Random kütüphanesini kullanmak**

from random import randint  
  
print(randint(0,100))

* 1. **Fonksiyonlar**
* Programcılar böyle tekrarlanan işler için aynı kodu tekrar tekrar yazmaktansa, bunları bir *fonksiyon* (prosedür veya yordam olarak da bilinir) olarak paketleyip kullanmayı tercih ederler.
* Bir fonksiyon bir kara kutu gibi düşünülebilir: Aldığı *parametre*ler onun girdisi, verdiği (“döndürdüğü”) değer ise çıktısıdır. Fonksiyonlara istediğiniz sayıda parametre verebilirsiniz. Parametre almayan ve/veya geriye bir değer vermeyen fonksiyonlar da olabilir.

def f1(x):  
 print(x)  
  
f1(3)

def f(x, y):  
 return x+y, x-y  
  
toplam, fark = f(3,5)  
print("Toplam = {}, Fark = {}".format(toplam,fark))

* 1. **Hata Yakalama (Try - Except)**
* **Hata yakalama** (exception handling) beklenmedik durumlarda programınızın bir hata mesajı vermesi ve çalışmayı durdurması yerine, hataya kendi istediğimiz şekilde cevap vermesini sağlamanın bir yoludur.

try:  
 a=5  
 b=0  
 print(a/b)  
except:  
 print("Sayı Sıfıra Bölünemez")

**Basit hesap makinesi**

x = int(input("1.sayi ="))  
y = int(input("2.sayi = "))  
  
print("1 numaralarali islem + \n 2 numarali islem - \n 3 numarali islem \* \n 4 numarali islem / \n")  
islem = int(input("islem secin \n"))  
  
if islem == 1:  
 print(x+y)  
  
elif islem == 2:  
 toplam2 = x - y  
 print(toplam2)  
  
elif islem == 3:  
 toplam3 = x \* y  
 print(toplam3)  
  
elif islem == 4:  
 toplam4 = x / y  
 print(toplam4)

1. **Obje Odaklı Programlama**
   1. **Class yapısı**

* Sınıflar, nesne üretmemizi sağlayan veri tipleridir. İşte nesne tabanlı programlama, adından da anlaşılacağı gibi, nesneler (ve dolayısıyla sınıflar) temel alınarak gerçekleştirilen bir programlama faaliyetidir.
* Sınıfların ilk harfi genelde büyük yazılır.

class Muhendis():  
 adi="yazılımhanem"  
 meslek="Bilgisayar Mühendisi"  
  
  
  
print(Muhendis.adi)  
print(Muhendis.meslek)

* Nesne üretmek

muhendis=Muhendis()

* 1. **Methodlar**
* Fonksiyon (*function*) diyoruz, *method* yani yöntem ise **class**içinde tanımlanınca bu ismi alıyor. Tabii aşağıdaki *method* çağırırken, o **class’a** ait bir nesne ile çağırmanız gerekli

class Muhendis():  
 adi="yazılımhanem"  
 meslek="Bilgisayar Mühendisi"  
  
 def EkranaYaz(self):  
 print(self.adi)  
 print(self.meslek)

* 1. **İnit ve Self Kavramları**
* **\_\_init\_\_** metodu, OOP ile programlamada bir class’ın yapıcı (constructor) metodur. Eğer bir class’tan nesne türetecek isek **\_\_init\_\_**, class’ın ilk metodu olmak zorundadır. Class içinden türetilen nesnelere ait özellikler bu metot ile nesnelere atanır.
* **Self**anahtar sözcüğü (keyword) \_**\_init\_\_** metodu ile gelen ve **class** içinden türetmiş olduğumuz nesnelere ulaşmamızı sağlayan bir kavramdır

class SuperKahraman:  
 def \_\_init\_\_ (self):  
 print("init cagrildi")  
  
super = SuperKahraman()

class Calisan():  
 class Arabalar():  
 def \_\_init\_\_(self, marka, model, renk, vites):  
 self.marka = marka  
 self.model = model  
 self.renk = renk  
 self.vites = vites  
  
 araba\_1 = Arabalar("Ford", "Mustang", "Siyah", "Manuel")  
  
 araba\_2 = Arabalar("BMW", "X6", "Beyaz", "Otomatik")  
  
 araba\_3 = Arabalar()  
  
 print(araba\_3)

* 1. **Kalıtım (Inheritance)**
* Inheritance veya kalıtım bir sınıfın başka bir sınıftan özelliklerini (**attribute**) ve metodlarını miras almasıdır.

class Hayvan():  
 def \_\_init\_\_(self):  
 print("hayvan init calisti")  
  
 def method1(self):  
 print("hayvan method1 calisti")  
  
 def mothod2(self):  
 print("hayvan method2 calisti")  
  
  
class Kedi(Hayvan):  
 def \_\_init\_\_(self):  
 Hayvan.\_\_init\_\_(self)  
 print("kedi init calisi")  
  
k = Kedi()

* 1. **Overridding (Iptal Etmek)**
* Eğer biz miras aldığımız metodları **aynı isimle** kendi sınıfımızda tekrar tanımlarsak, artık metodu çağırdığımız zaman **miras aldığımız değil kendi metodumuz çalışacaktır.** Buna Nesne Tabanlı Programlamada bir metodu override etmek denmektedir.

class Hayvan():  
 def \_\_init\_\_(self):  
 print("hayvan init calisti")  
  
 def method1(self):  
 print("hayvan method1 calisti")  
  
 def mothod2(self):  
 print("hayvan method2 calisti")  
  
  
class Kedi(Hayvan):  
 def \_\_init\_\_(self):  
 Hayvan.\_\_init\_\_(self)  
 print("kedi init calisi")  
  
 def miyav(self):  
 print("miyav")  
  
 def method1(self):  
 print("kedi method1")

* 1. **Polimorfizm**
* Her iki nesnede de aynı isimdeki metodu kullanıp farklı çıktılar elde etme işlemine **Polymorphism** yani**Çok Biçimlilik** denilmektedir. Bizler, aynı isme sahip birden fazla metot oluşturabilir fakat her metodun işlevini birbirinden farklı şekilde düzenleyebiliriz.

class Calisanlar():  
  
 def \_\_init\_\_(self, isim, soyisim, maas, departman):  
 self.isim = isim  
 self.soyisim = soyisim  
 self.maas = maas  
 self.departman = departman  
  
 def bilgileri\_goster(self):  
 print("İsim:", self.isim,"\n","Soyisim:", self.soyisim,"\n","Maaş:", self.maas,"\n", "Departman:", self.departman)  
  
 def mesai\_hesapla(self, mesai\_saati):  
 saat\_ucreti = self.maas/30/8  
 mesai\_ucreti = mesai\_saati \* (saat\_ucreti \* 1.2)  
 print("{} çalışanına ait mesai ücreti {} Türk Lirasıdır.".format(self.isim, mesai\_ucreti))  
  
calisan\_1 = Calisanlar("Mert", "Kaya", 6500, "Bilgi İşlem")  
  
class Yonetici(Calisanlar):  
  
 def \_\_init\_\_(self, isim, soyisim, maas, departman, sicil\_puani):  
 super().\_\_init\_\_(isim, soyisim, maas, departman,)  
 self.\_\_sicil\_puani = sicil\_puani  
  
 def bilgileri\_goster(self):  
 print("İsim:", self.isim,"\n","Soyisim:", self.soyisim,"\n","Maaş:", self.maas,"\n", "Departman:", self.departman,"\n","Sicil Puanı:",self.\_\_sicil\_puani)  
  
 def mesai\_hesapla(self, mesai\_saati):  
 saat\_ucreti = self.maas/30/8  
 mesai\_ucreti = mesai\_saati \* (saat\_ucreti \* 1.4)  
 print("{} yöneticisine ait mesai ücreti {} Türk Lirasıdır.".format(self.isim, mesai\_ucreti))  
  
yonetici\_1 = Yonetici("Hakan", "Elmas", 9500, "Bilgi İşlem",100)  
  
calisan\_1.mesai\_hesapla(5)  
  
yonetici\_1.mesai\_hesapla(5)

1. **Numpy**
   1. **Kurulum**

* Numpy’ı indirmek için terminale “conda install numpy” yazip enter’ a basıyoruz. (anaconda kullanıyorsak)

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

* Eğer hata alırsanız bu linkten bakabilirsiniz:

<https://towardsdatascience.com/how-to-successfully-install-anaconda-on-a-mac-and-actually-get-it-to-work-53ce18025f97>

* Pycharm için

1. İlk olarak proje oluştur ve içine python dosyası oluştur
2. File kısmından other settings ve preferences for new projects diyoruz

metin, ekran görüntüsü, ekran içeren bir resim

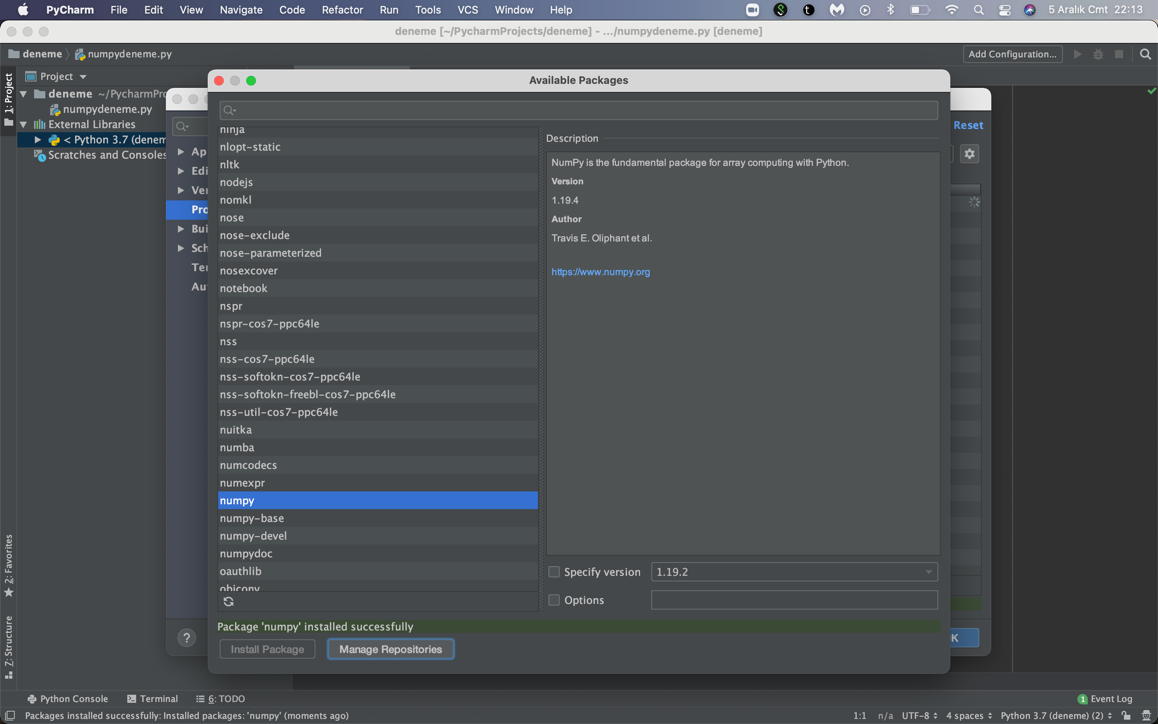
Açıklama otomatik olarak oluşturuldu

1. Project interpretter’dan a girip project interpreter’dan kendi projemizi seçiyoruz. Ve aşağıdaki “+” işaretine basıyoruz.

metin, ekran, ekran görüntüsü, siyah içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. Açılan pencerede Numpy’ı seçip “install package” diyoruz. Successfully yazısını görünce bu pencereyi kapatıyoruz.



1. OK deyip çıkıyoruz

Her kütüphane için böyle yapıyoruz.

* 1. **Numpy nedir?**
* **NumPy**(Numerical Python) bilimsel hesaplamaları hızlı bir şekilde yapmamızı sağlayan bir matematik kütüphanesidir. Numpy’ın temelini numpy dizileri oluşturur. Numpy dizileri python listelerine benzer fakat hız ve işlevsellik açısından python listelerinden daha kullanışlıdır. Ayrıca python listelerinden farklı olarak Numpy dizileri homojen yapıda olmalıdır yani dizi içindeki tüm elemanlar aynı veri tipinden olmalıdır.
  1. **Basit Örnek**

import numpy as np  
  
import matplotlib.pyplot as matplot  
  
maasListesi = np.random.normal(4000,500,1000)  
np.mean(maasListesi)  
  
matplot.hist(maasListesi,50)  
matplot.show()

metin, ekran görüntüsü, ekran, dizüstü içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. **Numpy Giriş**

Tek boyutlu ya da iki boyutlu(matris) diziler oluşturabiliyoruz.

import numpy as np  
  
#list  
benimListem = [20,30,40]  
ikiBoyutluListe = [[10,20,30],[30,40,50],[60,70,80]]  
  
#numpy dizisi  
  
np.array(benimListem)  
np.array(ikiBoyutluListe)  
  
print(np.array(ikiBoyutluListe))

* **Kolay bir şekilde dizi oluşturmak**
  + 1. **Arange**

**np.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 döndürür.

Not: Bitiş değerinin diziye dahil edilmediğine dikkat edelim.

**Genel kullanım:**

np.arange(başlangıç,bitiş,adım sayısı)

np.arange(bitiş) → bu kullanımda default başlangıç değeri 0 , adım sayısı ise 1 olarak kabul edilir. Yani aslında yukarıda ki formatta ifade edecek olursak np.arange(0,bitiş,1) şeklinde yazabiliriz.

print(np.arange(0,10))  
  
print(np.arange(0,10,2))

* + 1. **Zeros**
* Belirlediğimiz sütun ve satır sayısında**0** değerlerinde oluşan diziler oluşturur.

print(np.zeros(5))  
  
print(np.zeros((2,2)))

* + 1. **Ones**
* Belirlediğimiz sütun ve satır sayısında**1** değerlerinde oluşan diziler oluşturur.

print(np.ones(5))  
  
print(np.ones((2,2)))

* + 1. **Linspace**
* 3 parametre alır. Birinci parametre ile ikinci parametre**arasında**, üçüncü parametre değeri kadar**float** verisinden oluşan bir dizi döndürür.

print(np.linspace(0,20,6))

* + 1. **Eye**
* Eye fonksiyonu aldığı integer parametrenin değeri kadar **sütun** ve**satır** oluşturup**0** ile doldurur. Sol üst köşeden sağ alt köşeye, birlerden oluşan çapraz bir çizgi geçer.

print(np.eye(2))

* 1. **Random**

#tek bir deger degil bize numpy dizisi gönderir  
np.random.randn(8)

#tek bir deger dondurmek icin  
np.random.randint(1,10)

# tek deger dizisi dondurmek  
np.random.randint(1,10,5)

* 1. **Array**
* Numpy’da dizi, aynı türde bir değerler tablosudur ve negatif olmayan tamsayılar dizisiyle dizinlenmiştir. Boyut sayısı dizi sıralamasıdır; bir dizinin şekli, her eksen boyunca dizinin boyutunu veren tamsayı dizisidir.

print(benimDizim.reshape(4,5)) #diziyi yeniden sekillendirir  
  
print(benimDizim.max()) #max deger  
  
print(benimDizim.min()) #min deger  
  
print(benimDizim.argmax()) #max kacinci index  
  
print(benimDizim.argmin()) #min kacinci index  
  
print(benimDizim.shape) #dizinin seklini soyler

* 1. **İndexleme**

benimDizim = np.arange(0,15)  
  
print(benimDizim)  
  
print(benimDizim[5])  
  
print(benimDizim[5:10]) #5 8 arasini getir demek  
  
benimDizim[5:8] = -5 #5 8 arasini -5 yap demek  
  
print(benimDizim[5:8])  
  
baskaDizi = np.arange(0,24)  
print(baskaDizi)  
  
slicingDizi = baskaDizi[4:9]  
print(slicingDizi)  
  
slicingDizi[:] = 700  
print(slicingDizi)

**metin, cihaz, metre içeren bir resim

Açıklama otomatik olarak oluşturuldu**

* 1. **Dizi işlemleri**

yeniDizi = np.random.randint(1,100,20)  
  
print(yeniDizi)  
  
yeni = yeniDizi > 24  
  
print(yeni)  
  
print(yeniDizi[yeni])  
  
print(yeniDizi + yeniDizi)  
print(yeniDizi - yeniDizi)  
print(yeniDizi / yeniDizi)  
print(yeniDizi \* yeniDizi)  
print(np.sqrt(yeniDizi))

1. **PANDAS**

Yapay zekâ, makine öğrenmesi veya derin öğrenme konularıyla ilgilenen hemen herkesin yolu Python ile kesişiyor. Bunun en büyük sebebi Python ’da bu alanlara yönelik çok geniş ve büyük kolaylık sağlayan kütüphanelerin var olmasıdır.

**Pandas** da kullanımı kolay veri yapıları sunan ve veri üzerinde analiz yapmayı sağlayan açık kaynak kodlu Python kütüphanelerinden birisidir. Sahip olduğu birçok özellik sebebiyle veri okuma ve önişleme aşamalarında makine öğrenmesi alanlarında çalışanlara büyük kolaylıklar sağlamaktadır.

Pandas paketiyle yapabileceğimiz temel işlemleri aşağıda bulabilirsiniz:

* Veriyi çeşitli formatlardaki dosyalardan okutmak ve dosyalara yazdırmak mümkün. Pandas, metin ve Excel dosyalarının yanında, Stata ve SAS gibi yazılımların formatlarını da destekliyor.
* Eksik değerleri doldurmak/çıkarmak için yöntemler içeriyor.
* Çeşitli veri çerçevelerini birleştirmek (SQL dilindeki JOIN ve MERGE işlemleri) mümkün.
* Farklı türlerde grafikler çizdirebiliriz.
* Groupby fonksiyonu sayesinde veriyi gruplara bölerek, yapacağımız işlemlerle (toplama, sayma, ortalama alma gibi) özetleyebiliriz
  1. **Seriler (Series)**
* Seri, etiketli verilerden oluşan tek boyutlu bir veri yapısıdır. Etiket değerlerine indeks denir. Verinin kendisi sayılar, dizeler ya da başka Python objelerinden oluşabilir. Serileri oluşturmak için listeler, sıralı diziler ya da sözlükler kullanılabilir.

import pandas as pd  
  
benimSozlugum = {"begum" : 50, "zeynep" : 10 }  
  
pd.Series(benimSozlugum)

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

benimYaslarim = [50,10]  
benimIsimlerim = ["begum", "zeynep"]  
  
pd.Series(benimIsimlerim, benimYaslarim)  
  
print(pd.Series(benimIsimlerim, benimYaslarim))  
  
print(pd.Series(data = benimYaslarim, index= benimIsimlerim))

benimIsimlerim = ["begum", "zeynep"]  
  
numpyDizisi = np.array([50,40])  
  
print(numpyDizisi)  
  
print(pd.Series(numpyDizisi))  
  
print(pd.Series(numpyDizisi, benimIsimlerim))

* 1. **Seri Özellikleri**

yarisma1 = pd.Series([1,2,3], ["begum", "zeynep","hatice"])  
  
yarisma2 = pd.Series([7,42,73], ["begum", "zeynep","hatice"])  
  
print(yarisma1 + yarisma2)

* 1. **Data Frame**
* Pandas Dataframe’ler pandasın çok boyutlu veri yapılarıdır. Bunu en kolay benzetebileceğimiz yapı sql veya excel tabloları gibi görebiliriz. Bunlar gibi satır ve sütunlardan oluşurlar. Dataframe yapıları veri analizi işlemlerinde oldukça faydalı olmaktadırlar.

**Data Frame oluşturma**

data = np.random.randn(4,3)  
  
dataFrame = pd.DataFrame(data)  
  
print(dataFrame)

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Data Frame’in 0. İndexsini görüntüleme**

data = np.random.randn(4,3)  
  
dataFrame = pd.DataFrame(data)  
  
print(dataFrame[0])

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Data Frame oluşturma satır sütun ismi verme**

data = np.random.randn(4,3)  
  
yeni = pd.DataFrame(data, index = ["atil", "begum","arda","melis"], columns= ["Maas", "yas", "calisma alani"])  
  
print(yeni)

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Sütun çağırma**

print(yeni["yas"])

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Satır çağırma**

print(yeni.loc["atil"])

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Yeni alan ekleme**

yeni["emeklilik yasi"] = yeni["yas"] + yeni["yas"]  
  
print(yeni)

**Alan silmek**

yeni.drop("emeklilik yasi", axis = 1, inplace = True)  
print(yeni)

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Belirli bir değer çağırmak**

print(yeni.loc["begum"]["Maas"])  
print(yeni.loc["begum","Maas"])

**Yaşı 0’dan büyük olanları getir**

print(yeni[yeni["yas"] > 0])

**İndex resetlemek**

print(yeni.reset\_index())

**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**İndex değiştirmek**

newIndex= ["a","z","a","m"]  
  
yeni["yeni index"] = newIndex  
yeni.set\_index("yeni index", inplace= True)  
  
print(yeni)

**Birleşmiş indexler**

ilkIndexler = ["simpson", "simpson","simpson", "south park","south park","south park"]  
  
icIndexler = ["homer", "bart" , "marge" , "cartman", "kyle" , "kenny"]  
  
birlesmisIndex = list(zip(ilkIndexler,icIndexler))  
  
birlesmisIndex = pd.MultiIndex.from\_tuples(birlesmisIndex)  
  
listem = [[40, "a"], [10, "b"], [30,"c"], [40,"e"],[20,"y"],[60,"o"]]  
  
numpyDizisi = np.array(listem)  
  
dataFrame = pd.DataFrame(numpyDizisi, index=birlesmisIndex, columns= ["yas", "meslek"])  
  
print(dataFrame)

**metin, cihaz, metre içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Eksik veri olduğunda ne yapacağız**

**Silinebilir**

sozlukVerisi = {"istanbul" : [30,29,np.nan], "Ankara" : [20,np.nan,30], "izmir" : [40,34,56]}  
havaDurumDataFrame = pd.DataFrame(sozlukVerisi)  
  
print(havaDurumDataFrame)  
  
#olmayan verileri direk silebiliriz  
  
silinenSutun = havaDurumDataFrame.dropna(axis=0)  
  
silinenSatir = havaDurumDataFrame.dropna(axis=1)  
  
print(silinenSatir)  
print(silinenSutun)

#yeni veri ekledik

yeniVeri = {"istanbul" : [30,29,np.nan], "Ankara" : [20,np.nan,30], "izmir" : [40,34,56], "antalya" : [45,np.nan,np.nan]}  
  
yeniDataFrame = pd.DataFrame(yeniVeri)  
  
print(yeniDataFrame)

#iki ve uzeri nan olanlari sil  
  
ikiveUzeriNanSil = yeniDataFrame.dropna(axis=1,thresh =2)  
  
print(ikiveUzeriNanSil)

**Boş olanlar doldurulabilir**

doldur = yeniDataFrame.fillna(10)  
print(doldur)

**Groupby metodu**

maasVerisi = {"departman" : ["yazilim","satis","yazilim","satis"],  
 "calisan ismi" : ["begum" , "zeynep" , "ridat" , "ali"],  
 "maas" : [100,150,200,300]}  
  
maasDataFrame = pd.DataFrame(maasVerisi)  
  
print(maasDataFrame)  
  
grupObjesi = maasDataFrame.groupby("departman")  
  
print(grupObjesi.count())  
print(grupObjesi.mean())  
print(grupObjesi.min())  
print(grupObjesi.max())  
print(grupObjesi.describe())

**Kolon birleştirmek (**concatenation**)**

sozluk1 = {"isim" : ["ahmet" , "mehmet", "ali"],  
 "spor" : ["kosu", "yuzme" , "basketbol"],  
 "kalori" : [100,200,300]  
 }  
  
dataFrame1 = pd.DataFrame(sozluk1, index= [1,2,3])  
  
sozluk2 = {"isim" : ["hasan" , "veli", "zeki"],  
 "spor" : ["kosu", "yuzme" , "basketbol"],  
 "kalori" : [600,800,300]  
 }  
  
dataFrame2 = pd.DataFrame(sozluk2, index= [4,5,6])  
  
sozluk3 = {"isim" : ["begum" , "duygu", "zeynep"],  
 "spor" : ["tenis", "yuzme" , "basketbol"],  
 "kalori" : [600,800,300]  
 }  
  
dataFrame3 = pd.DataFrame(sozluk3, index=[7,8,9])  
  
print(dataFrame1)  
print(dataFrame2)  
print(dataFrame3)  
  
#concatenation islemi  
  
print(pd.concat([dataFrame1,dataFrame2,dataFrame3]))

**Merge**

sozluk1 = {"isim" : ["ahmet" , "mehmet", "ali"],  
 "spor" : ["kosu", "yuzme" , "basketbol"]  
 }  
  
dataFrame1 = pd.DataFrame(sozluk1, index= [1,2,3])  
  
sozluk2 = {"isim" : ["ahmet" , "mehmet", "ali"],  
 "kalori" : [600,800,300]  
 }  
  
dataFrame2 = pd.DataFrame(sozluk2, index= [4,5,6])  
  
print(dataFrame1)  
print(dataFrame2)  
  
print(pd.merge(dataFrame1,dataFrame2,on="isim"))

**İleri Operasyonlar**

maasVerisi = {"departman" : ["yazilim","satis","yazilim","satis"],  
 "calisan ismi" : ["begum" , "zeynep" , "ridat" , "ali"],  
 "maas" : [100,150,200,300]}  
  
maasDataFrame = pd.DataFrame(maasVerisi)  
  
print(maasDataFrame)  
  
print(maasDataFrame["departman"].unique())  
  
print(maasDataFrame["departman"].nunique())  
  
print(maasDataFrame["departman"].value\_counts())  
  
def bruttenNete (maas):  
 return maas \* 0.66  
  
print(maasDataFrame["maas"].apply(bruttenNete))  
  
print(maasDataFrame.isnull())

**Excel dosyası açmak ve içine yazmak**

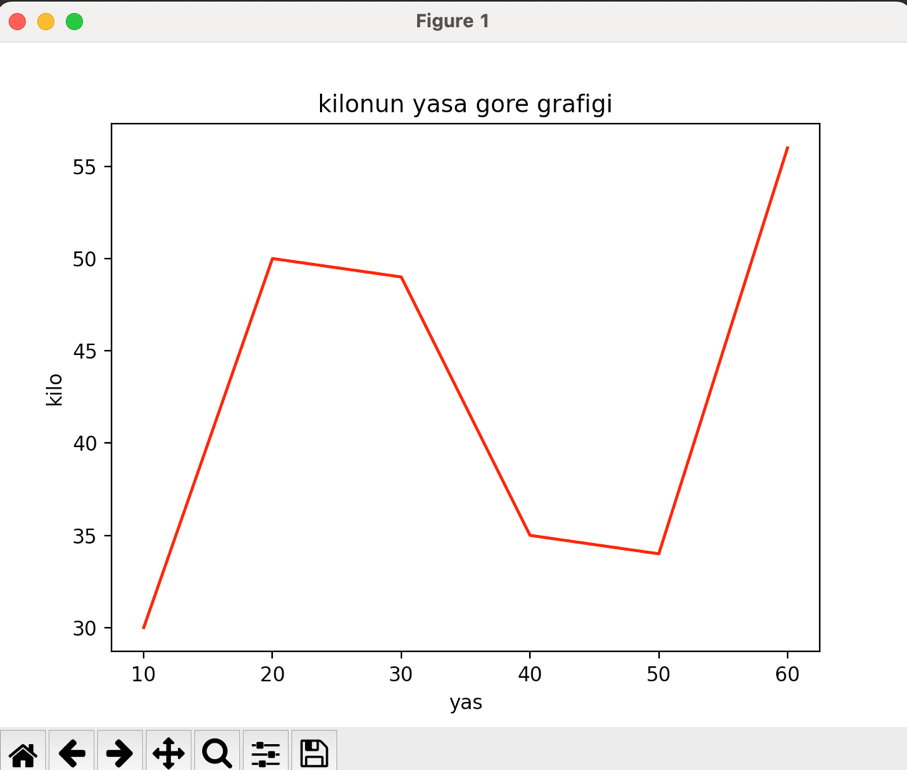
import openpyxl  
import pandas as pd  
  
df = pd.read\_excel("Kitap1.xlsx")  
  
print(df.head())  
  
doluD = df.dropna()  
  
print(doluD)  
  
writer = pd.ExcelWriter('new\_book.xlsx')  
doluD.to\_excel(writer, 'new\_sheet')  
writer.save()

1. **Matplotlib**

Matplotlib; veri görselleştirmesinde kullandığımız temel python kütüphanesidir. 2 ve 3 boyutlu çizimler yapmamızı sağlar. Matplotlib genelde 2 boyutlu çizimlerde kullanılırken, 3 boyutlu çizimlerde başka kütüphanelerden yararlanılır.

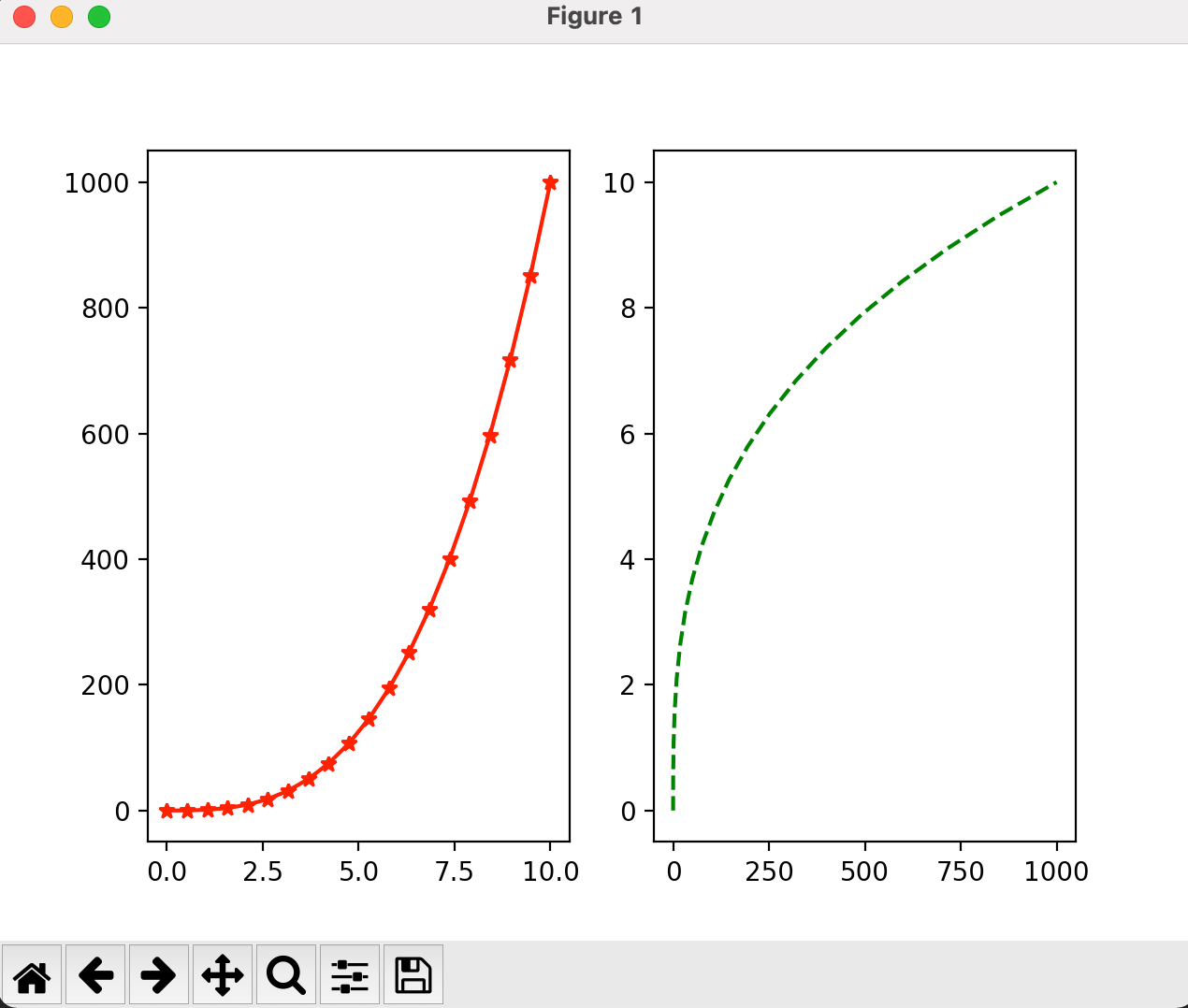
* Matplotlib en popüler görselleştirme kütüphanesidir.
* Bir figür üzerinde tamamen kontrol sağlayabilirsiniz.
* Matlab’ın grafik çizimine benzer bir his vermesi ve kıyasla daha kolay ve ücretsiz olması.

import numpy as np  
import matplotlib.pyplot as plt  
  
  
yasListesi = [10,20,30,40,50,60]  
  
kiloListesi = [30,50,49,35,34,56]  
  
numpyYasListesi = np.array(yasListesi)  
  
numpyKiloListesi = np.array(kiloListesi)  
  
plt.plot(numpyYasListesi,numpyKiloListesi,"r")  
plt.xlabel("yas")  
plt.ylabel("kilo")  
plt.title("kilonun yasa gore grafigi")  
  
plt.show()

****

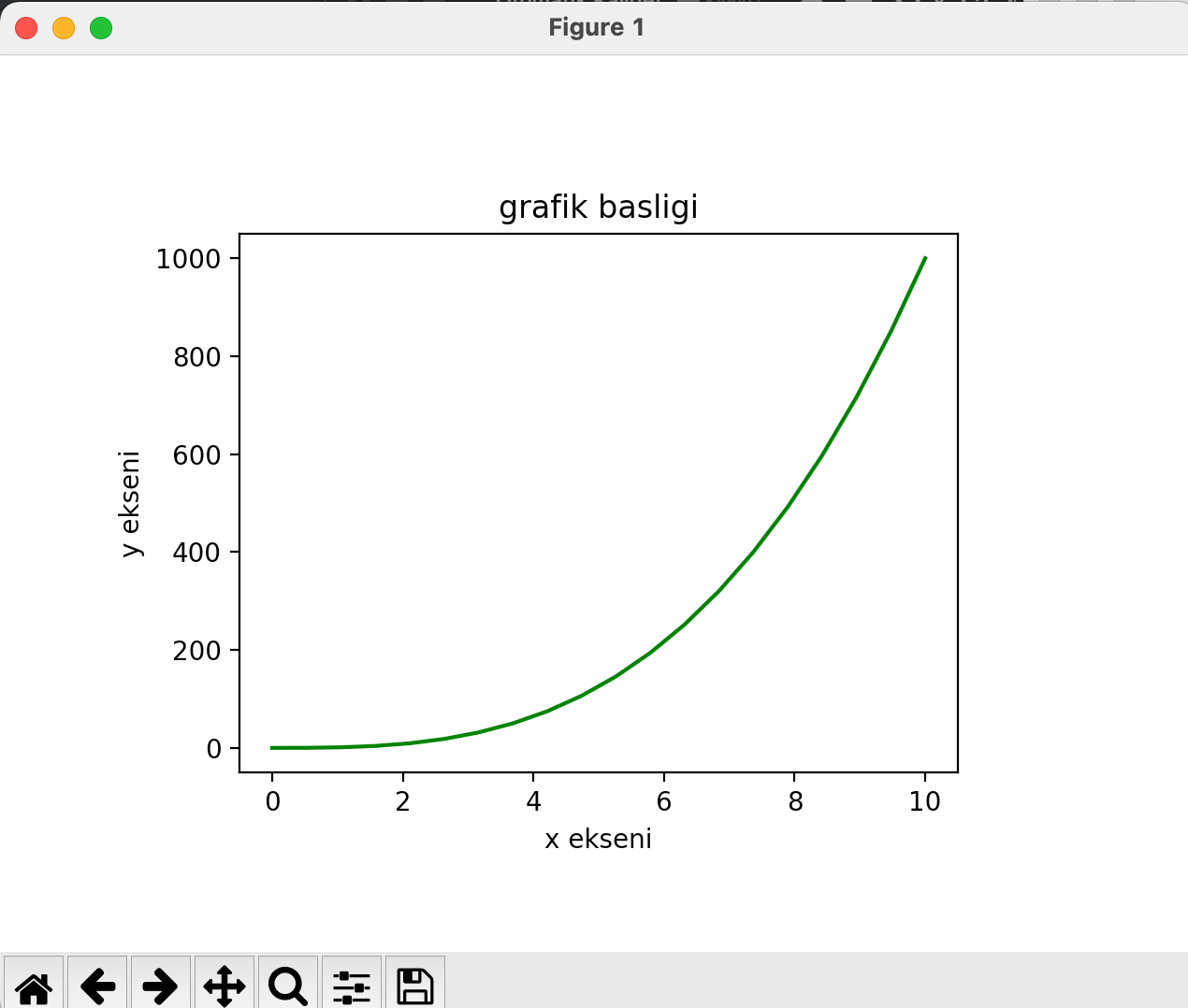
**2 grafik çizmek**

numpyDizi1 = np.linspace(0,10,20)  
  
numpyDizi2 = numpyDizi1 \*\*3  
  
#2 grafik cizmek  
#1.yi ciziyorum demek  
plt.subplot(1,2,1)  
plt.plot(numpyDizi1,numpyDizi2,'r\*-')  
#2.yi ciziyorum demek  
plt.subplot(1,2,2)  
plt.plot(numpyDizi2,numpyDizi1,"g--")  
  
plt.show()

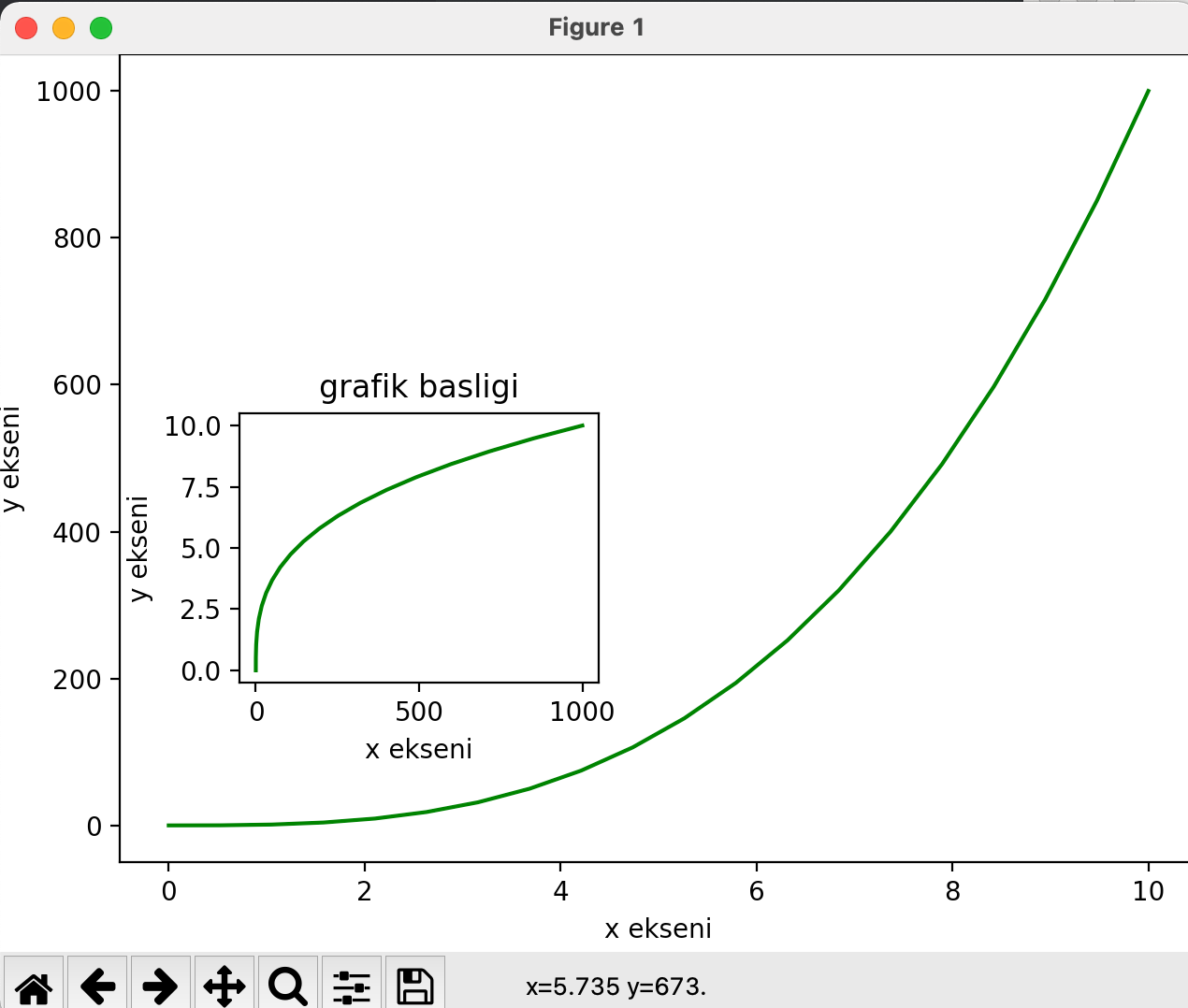
****

**Kendi figürümüzü oluşturmak**

#kendi figurumuz  
benimFigure = plt.figure()  
#argumanlari liste icinde vermek gerek  
figureAksanlari = benimFigure.add\_axes([0.2,0.2,0.6,0.6])  
figureAksanlari.plot(numpyDizi1,numpyDizi2,"g")  
figureAksanlari.set\_xlabel("x ekseni")  
figureAksanlari.set\_ylabel("y ekseni")  
figureAksanlari.set\_title("grafik basligi")  
  
plt.show()

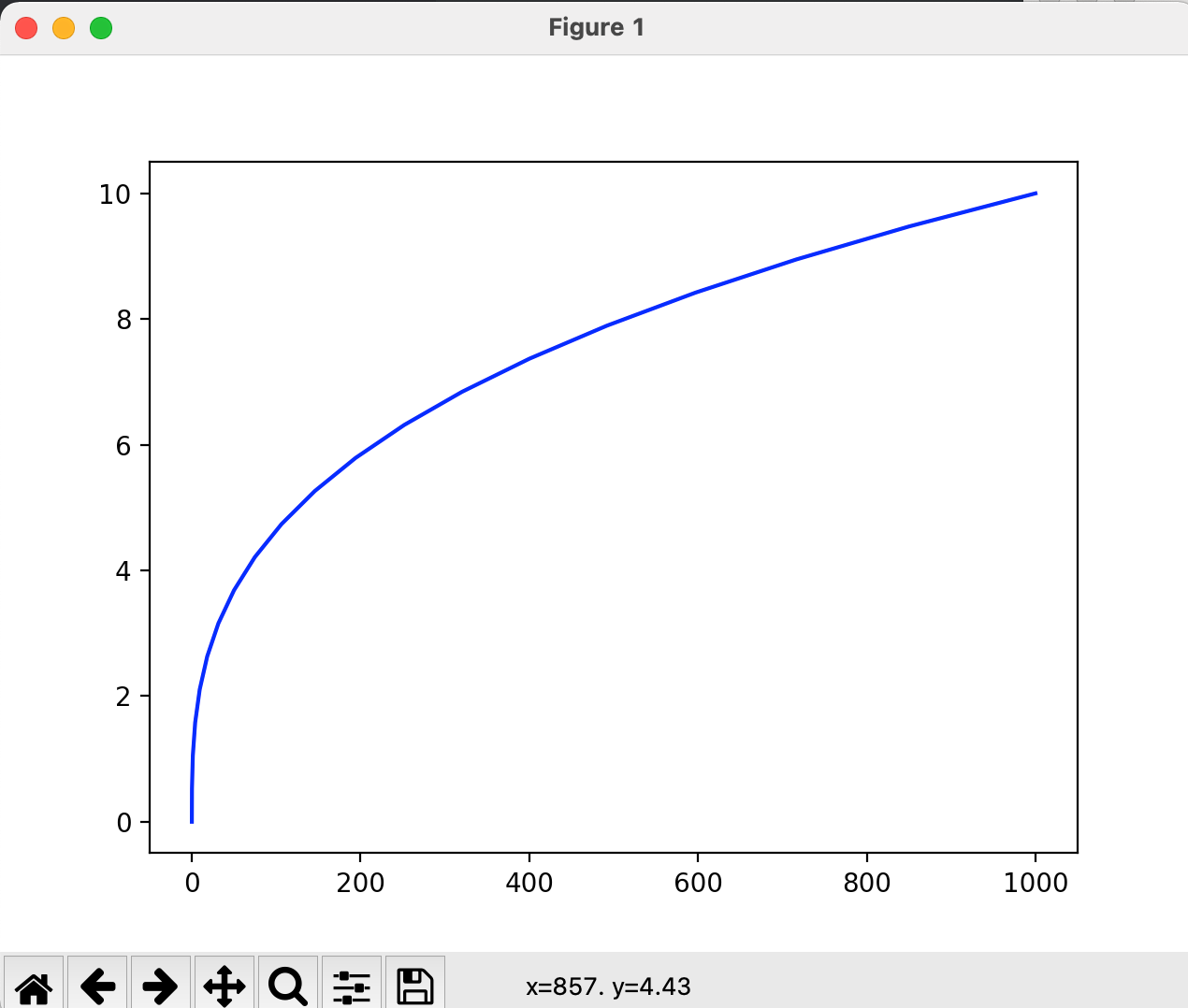
****

numpyDizi1 = np.linspace(0,10,20)  
  
numpyDizi2 = numpyDizi1 \*\*3  
  
figure2 = plt.figure()  
eksen1 = figure2.add\_axes([0.1,0.1,0.9,0.9])  
eksen2 = figure2.add\_axes([0.2,0.3,0.3,0.3])  
  
eksen1.plot(numpyDizi1,numpyDizi2,"g")  
eksen1.set\_xlabel("x ekseni")  
eksen1.set\_ylabel("y ekseni")  
eksen1.set\_title("grafik basligi")  
  
  
eksen2.plot(numpyDizi2,numpyDizi1,"g")  
eksen2.set\_xlabel("x ekseni")  
eksen2.set\_ylabel("y ekseni")  
eksen2.set\_title("grafik basligi")  
  
plt.show()

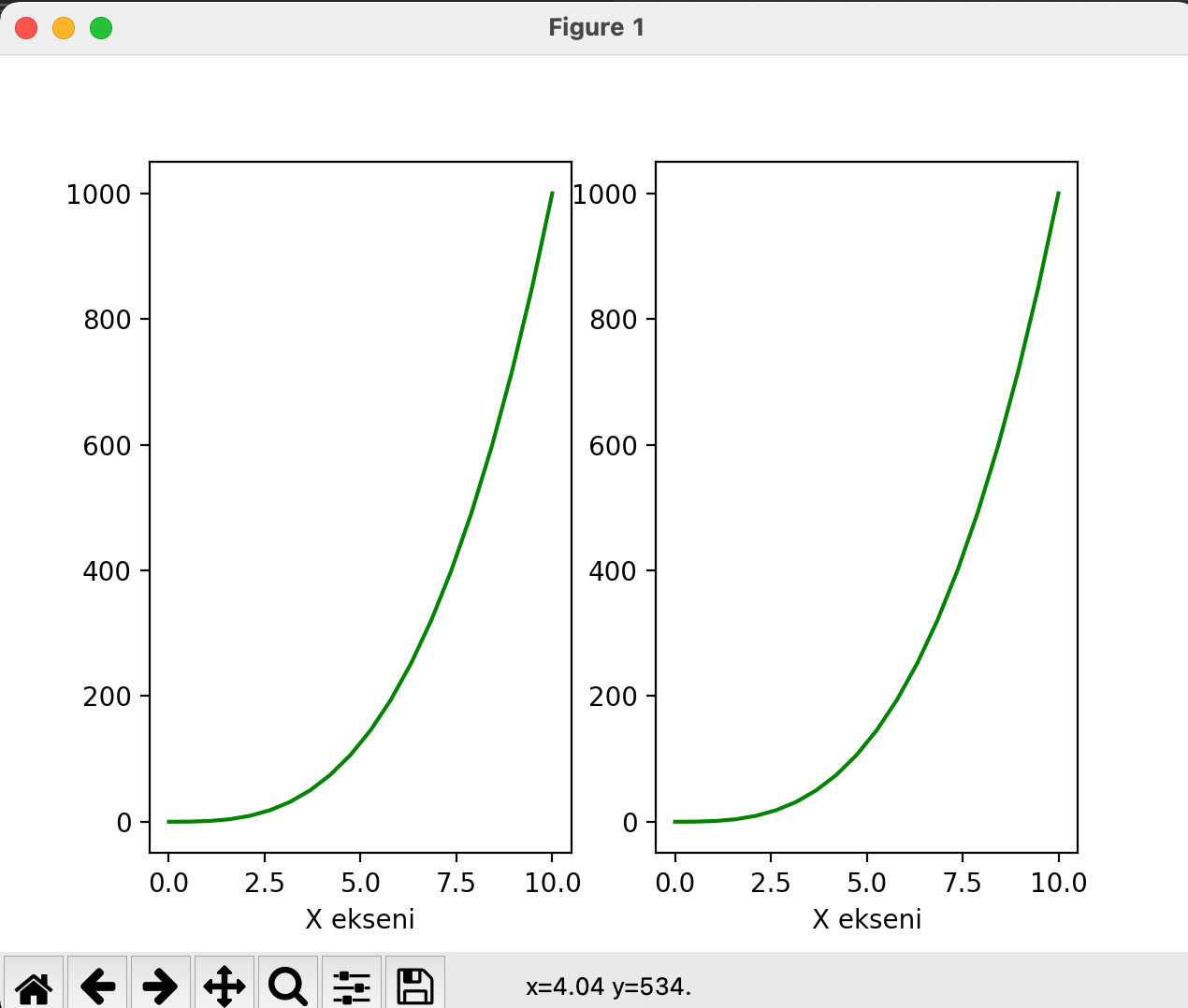
****

**Subplots kullanımı**

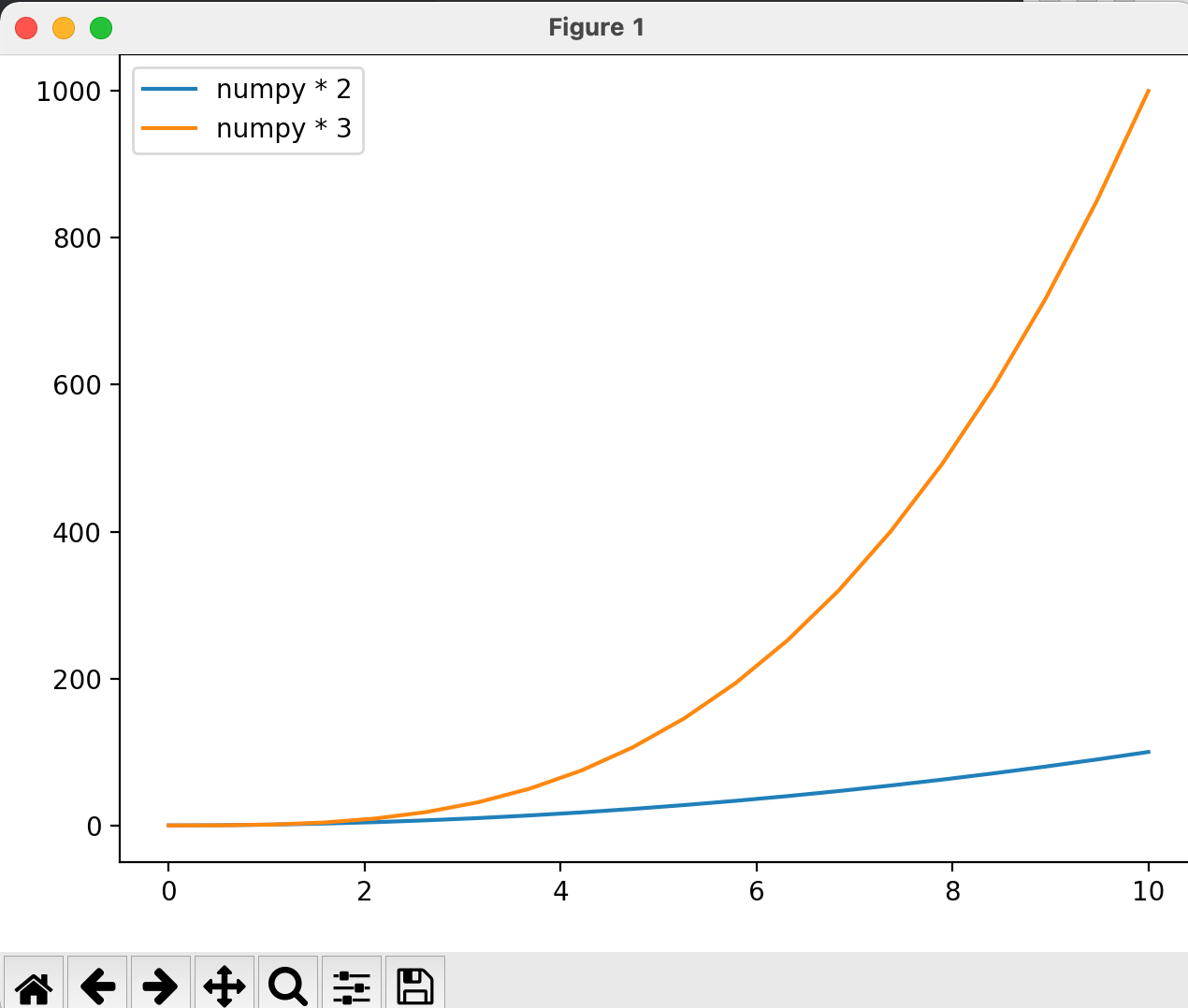
(benimFigure, benimEksen) = plt.subplots()  
  
benimEksen.plot(numpyDizi2,numpyDizi1,"b")  
  
plt.show()

****

(benimFigure, benimEksen) = plt.subplots(nrows=1, ncols=2)  
  
for eksen in benimEksen:  
 eksen.plot(numpyDizi1,numpyDizi2,'g')  
 eksen.set\_xlabel("X ekseni")  
  
plt.show()

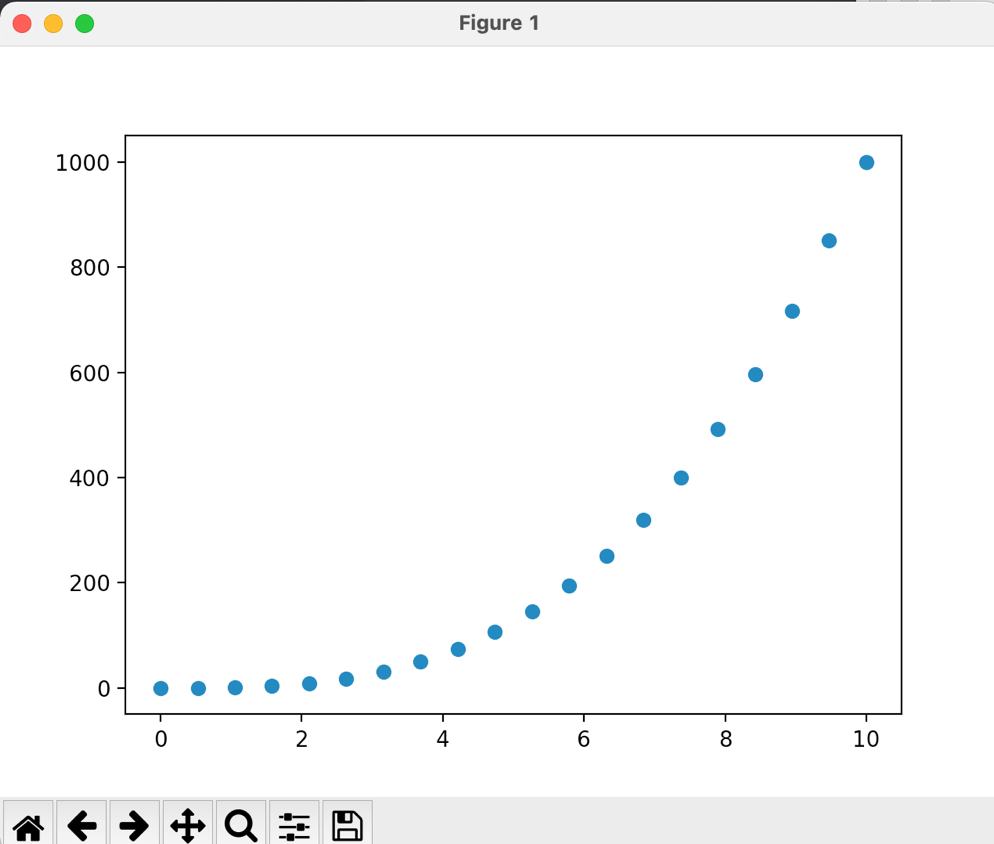
****

import numpy as np  
import matplotlib.pyplot as plt  
  
numpyDizi1 = np.linspace(0,10,20)  
  
numpyDizi2 = numpyDizi1 \*\*3  
  
  
yeniFigur = plt.figure()  
  
yeniEksen = yeniFigur.add\_axes([0.1,0.1,0.9,0.9])  
  
yeniEksen.plot(numpyDizi1,numpyDizi1 \*\* 2,label = "numpy \* 2")  
yeniEksen.plot(numpyDizi1,numpyDizi1 \*\* 3,label = "numpy \* 3")  
yeniEksen.legend(loc = 2)  
  
plt.show()  
  
#figur kaydetme  
yeniFigur.savefig("benim.png", dpi =200)

****

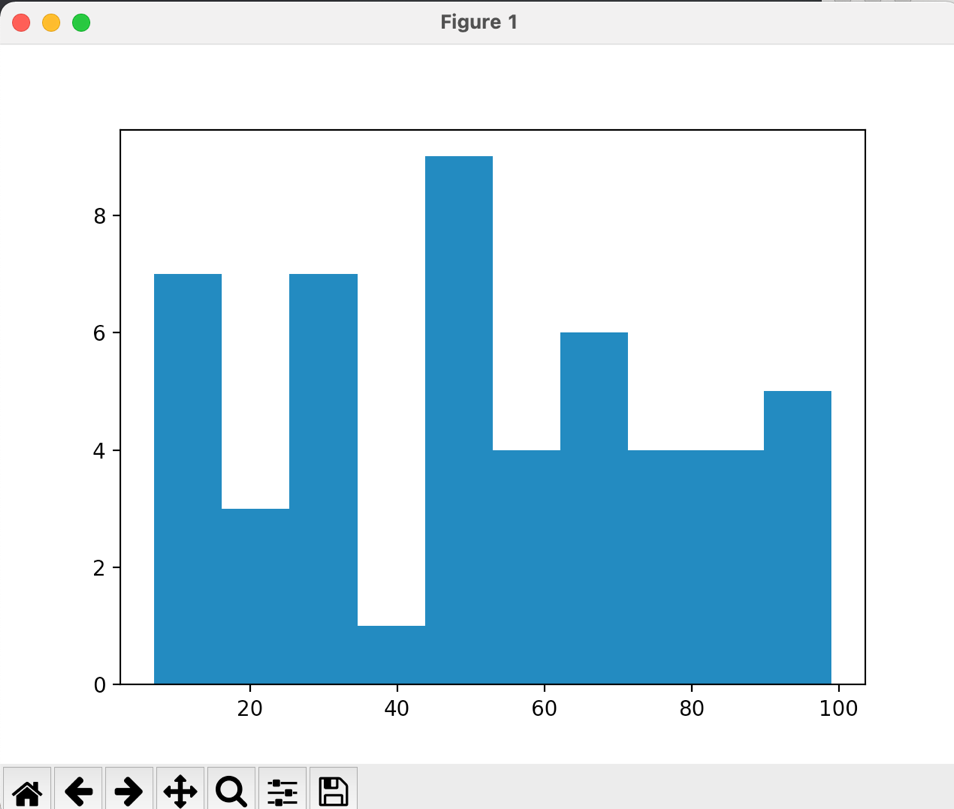
**Scatter**

plt.scatter(numpyDizi1,numpyDizi2)

****

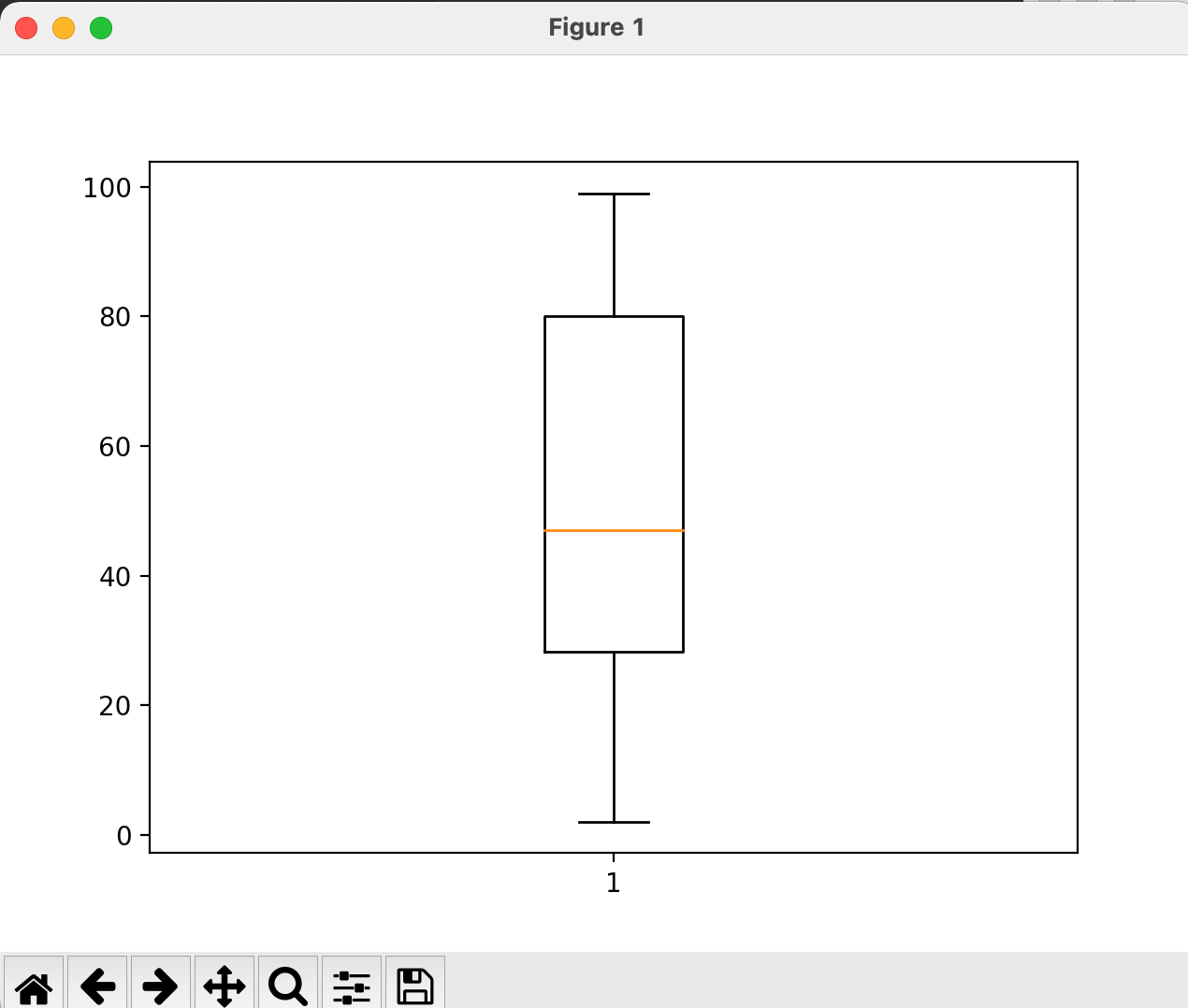
**Histogram**

numpyDizi1 = np.random.randint(0,100,50)  
plt.hist(numpyDizi1)  
plt.show()

****

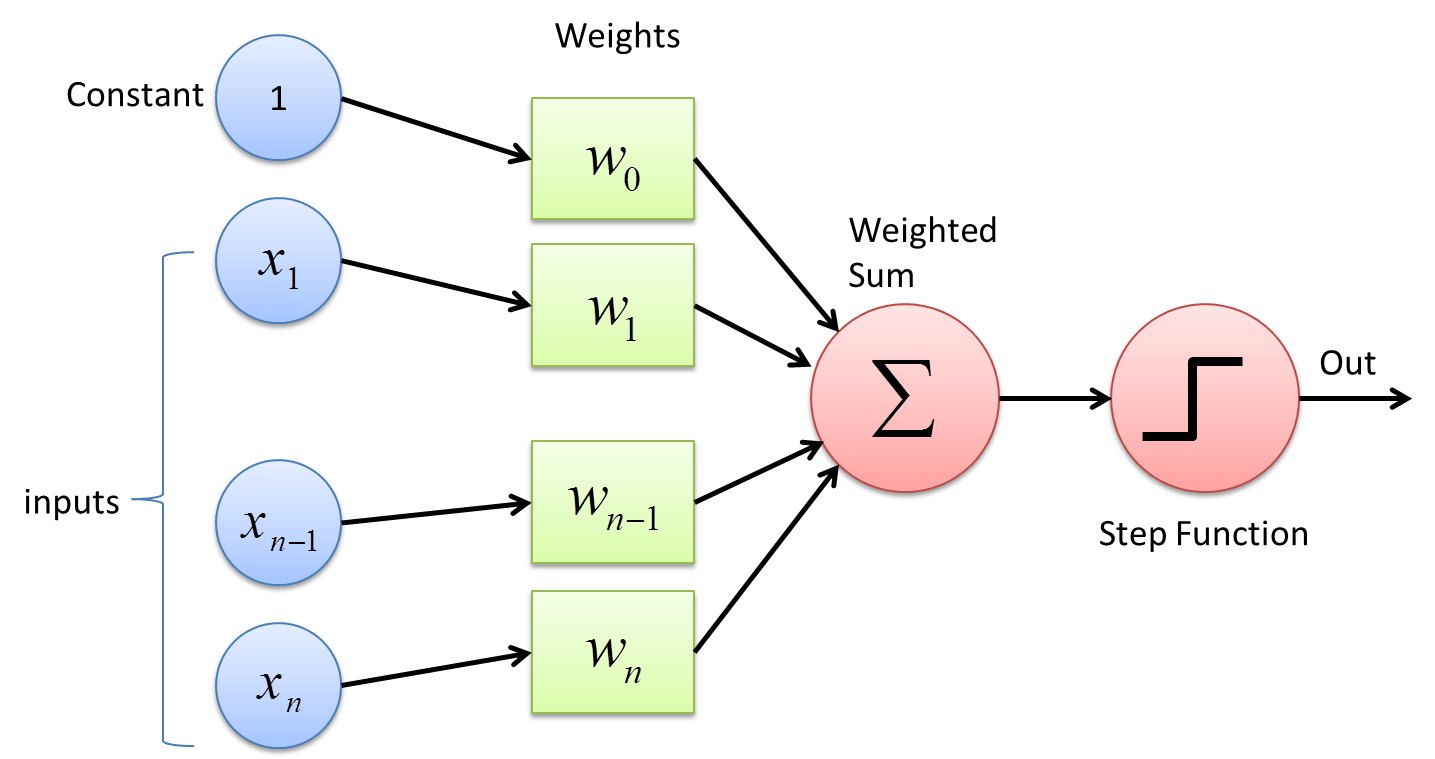
**Boxplot**

plt.boxplot(numpyDizi1)

****

1. **TENSORFLOW**

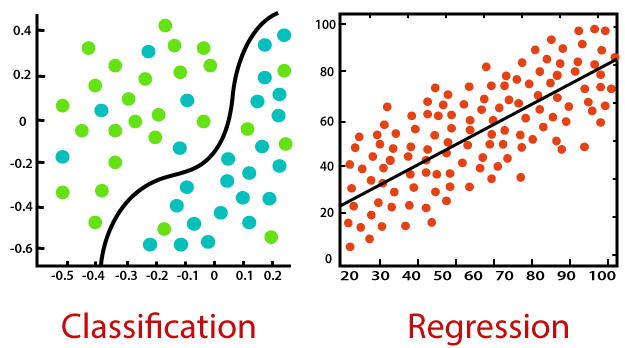
* **Google** tarafından geliştirilen açık kaynaklı TensorFlow ile derin öğrenme destekli yapay zekâ uygulamaları geliştirebiliyoruz
* Açık kaynak kodlu bir deep learning (derin öğrenme) kütüphanesidir. Esnek yapısı sayesinde, tek bir API ile platform farketmeksizin hesaplamaları, bir veya birden fazla CPU, GPU kullanarak deploy etmenize olanak sağlar.
* Temelinde Python kullanılarak geliştirilen bu framework, günümüzde Python‘ın yanısıra C++, Java, C#, Javascript ve R gibi birçok dili desteklemektedir.
  1. **Perceptron**
* Artificial Neural Network (ANN)
* Perceptron-> Yapay nöron

****

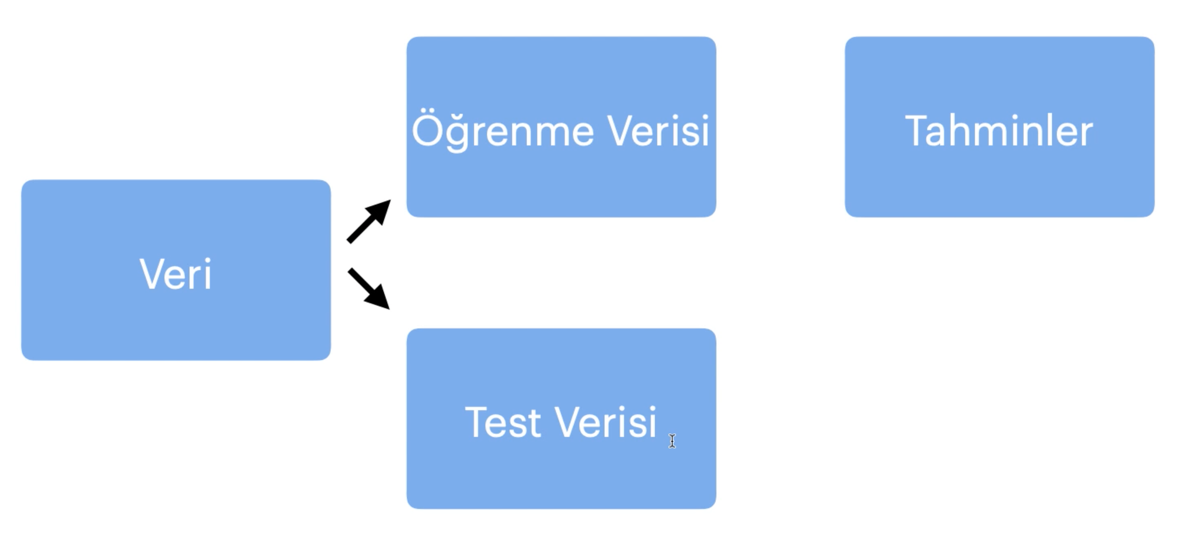
* Perceptron; en basit tek katmanlı sinir ağı modelidir. Temel olarak eğitilebilecek tek bir yapay sinir hücresinden oluşmaktadırlar. Biyolojik nöronları taklit ederler. Temel elemanları:
* Weights
* İnputs
* Adder
* Activation fonksiyonu (step function)
* Output
* İnputlar(i) nöron’a aktarılırken direnç oluşur. Bu ağırlıkları(w) almamızın sebebi budur. Ağırlıklar birbirinden farklıdır.
* Derin ağda birçok nöron olur.
* Bias: Inputlara 0 verirsek sorun çıkabilir bu tarz işlemleri engellesin diye verilen bir sabittir. Yukarıdaki şekilde görülüyor constant’tır.
  1. **Aktivasyon Fonksiyonları**
* **Sigmoid fonksiyonu**
* 0 ile 1 arasında değer alır.
* Genelde sınıflandırma problemlerinde işe yarar.
* **Tanh(Hiperbolik Tanjant) fonksiyonu**
* -1 ile 1 arasında değer alır.
* Negatif değerlerle daha geniş bir kapsam sağlar ve genelde sınıflandırma operasyonlarında kullanılır.
* **ReLU (Rectified Linear Unit) fonksiyonu**
* 0 ile sonsuz arasında değer alır.
* Derin öğrenme alanında sıklıkla karşımıza çıkar.
* **Linear fonksiyonlar**
* F(x) = x
* Sonsuz değer alabilir fakat non-linear olmaması sebebiyle modellerde sorunlara yol açabilir.
  1. **Problem Tipleri**

1. Regression
2. Classification
   * 1. **Regression**

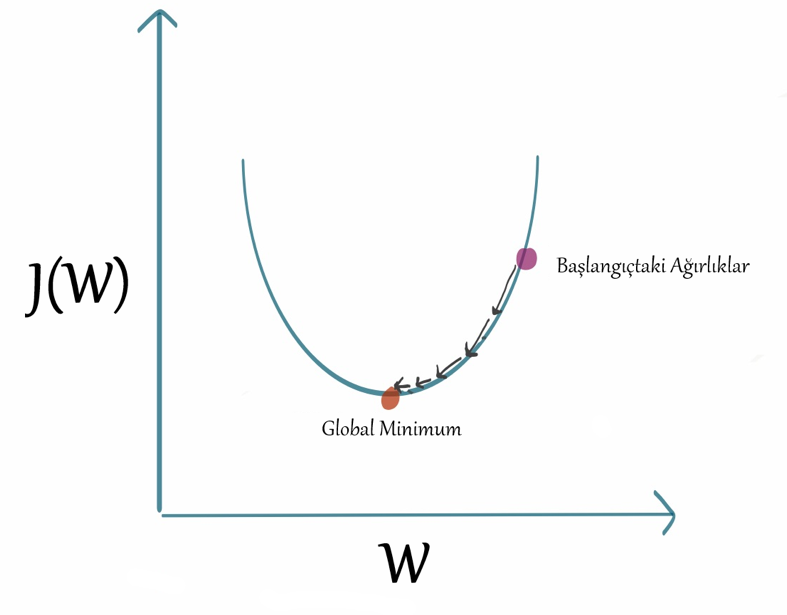
* Fiyat, yaş gibi sürekli(continuous) değerleri tahmin etmek için kullanılır.
* Output değerleri sürekli(continuous) olmalı ama İnputlar hem ayrık hem de sürekli değerler olabilir.
* Toplam veri setindeki ortalamaya yakın olma eğilimi gösterir
* Y = a \* x + b (bu denkleme ulaşmaya çalışıyoruz)
* Örnek
* Y = araba fiyatı
* x = arabanın özelliği
* a = ağırlık (bias’ı doğru bulursak fiyatı da doğru buluyoruz)
* b = bias
  + 1. **Classification (Sınıflandırma)**
* Kadın/erkek, true/false gibi ayrık(discrete) değerleri tahmin etmek için kullanılır.
* Output değerleri ayrık(discrete) olmalı ama İnputlar hem ayrık hem de sürekli değerler olabilir.



* 1. **Veri Setleri**

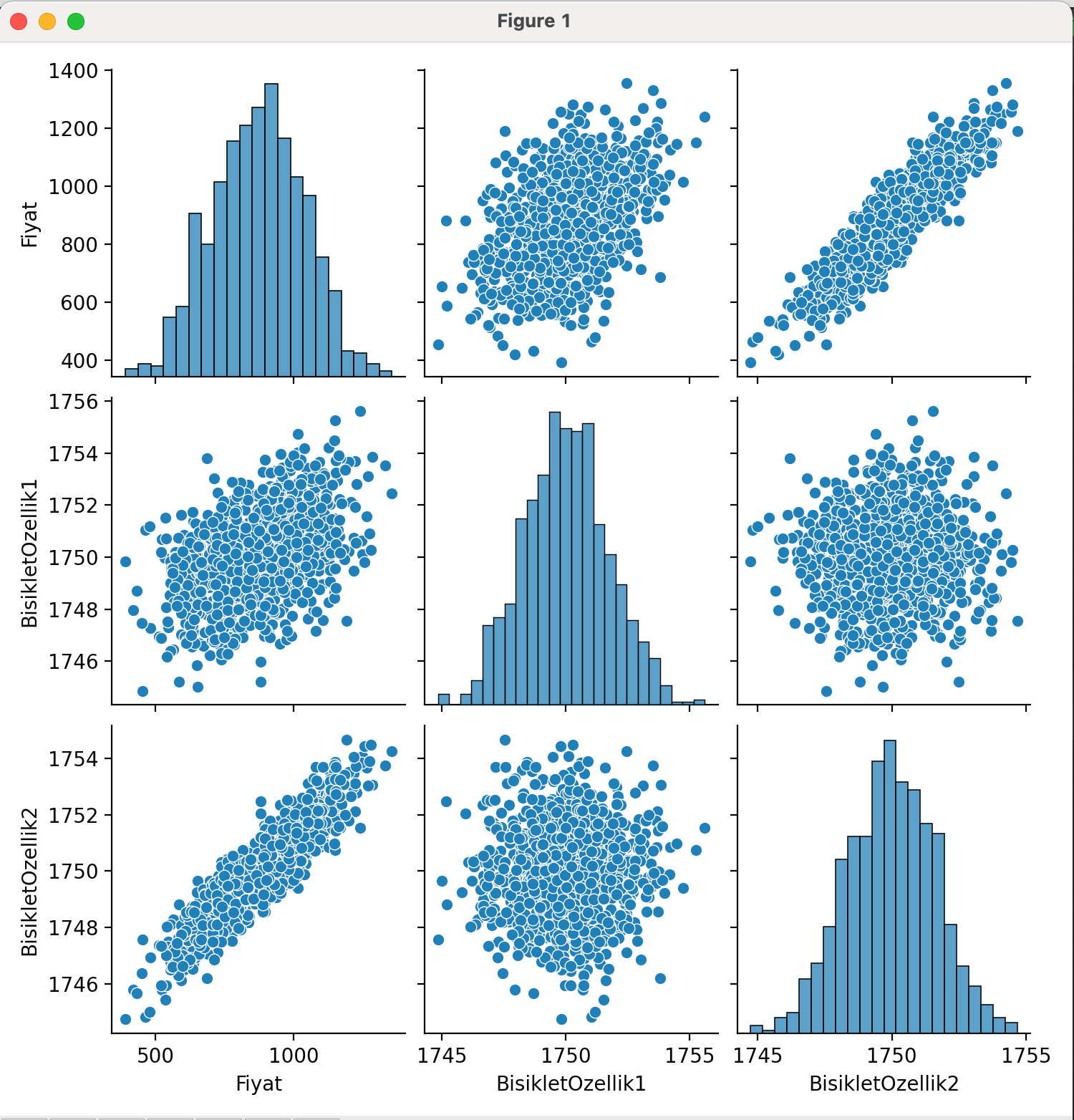


* Veri setleri 2 ye ayrılır. Öğrenme verisi (train set) ve test verisi (test set) olarak bunun nedeni öğrettiğimiz veriyi test etmek içinde kullanacağımız verinin olması gerekir. Genelde %80 e %20 bölünür.
  + 1. **Maliyet Fonksiyonu (Cost Function)**
* Öğrenme süresince bize yardımcı olur, gerçek veriden ne kadar uzaktayız ölçer.
* Ne kadar düşük bir değer çıkarsa o kadar yakınız demektir.
* Z = a \* g + b
* F(z) = tahminiDeger (nöron ‘un tahmini)
* Quadratic Maliyet = sum (gercekDeger- tahminiDeger) \*\* 2 / n
* Cross Entropy Maliyet = (-1/n) \* sum (gercekDeger \* In(tahminiDeger) + (1- gercekDeger) \* In (1- tahminiDeger)
  + 1. **Gradient Descent (Gradyan Azalması)**
* Bir fonksiyonun minimumunu bulmak için kullandığımız optimizasyon fonksiyonudur.
* Maliyet fonksiyonunu minimize etmek için kullanabiliriz.



* 1. **Tensorflow Giriş**

import openpyxl  
import pandas as pd  
import seaborn as sbn  
import matplotlib.pyplot as plt  
  
df = pd.read\_excel("bisiklet\_fiyatlari.xlsx")  
  
print(df.head())  
  
sbn.pairplot(df)  
  
plt.show()



 -> veri burada.

* 1. **Veriyi test/train olarak ikiye ayırmak**

import openpyxl  
import pandas as pd  
import seaborn as sbn  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split

# excel’ oku  
df = pd.read\_excel("bisiklet\_fiyatlari.xlsx")  
  
# y = wx + b için y ve x i bul  
  
y = df["Fiyat"].values  
  
x = df[["BisikletOzellik1","BisikletOzellik2"]].values

#modeli 2 ye bol  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size= 0.33 , random\_state= 15)  
  
print(x\_train.shape)  
print(x\_test.shape)  
print(y\_train.shape)  
print(y\_test.shape)

* 1. **Modeli oluşturmak**

import openpyxl  
import pandas as pd  
import seaborn as sbn  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import MinMaxScaler  
import tensorflow as tf  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
  
  
df = pd.read\_excel("bisiklet\_fiyatlari.xlsx")  
  
# y = wx + b  
  
y = df["Fiyat"].values  
  
x = df[["BisikletOzellik1","BisikletOzellik2"]].values

#modeli 2 ye bol  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size= 0.33 , random\_state= 15)  
  
print(x\_train.shape)  
print(x\_test.shape)  
print(y\_train.shape)  
print(y\_test.shape)  
  
#scalling(boyutunu degistirmek)  
#noronlara verilen veri setini kucultmek  
  
scaler = MinMaxScaler()  
scaler.fit(x\_train)  
x\_train = scaler.transform(x\_train)  
x\_test = scaler.transform(x\_test)  
  
#model olustur  
  
model = Sequential()  
#hidden layer kac tane is o kadar model.add(dense) yap  
model.add(Dense(5,activation = "relu")) # 5 noron sayisi, relu aktivasyon fonksiyonu tipi  
model.add(Dense(5,activation = "relu")) # 5 noron sayisi, relu aktivasyon fonksiyonu tipi  
model.add(Dense(5,activation = "relu")) # 5 noron sayisi, relu aktivasyon fonksiyonu tipi  
  
model.add(Dense(1)) #cikti nöronu

model.compile(optimizer = "rmsprop", loss = "mse") #calistirmak icin

* 1. **Traning**

import openpyxl  
import pandas as pd  
import seaborn as sbn  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import MinMaxScaler  
import tensorflow as tf  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
  
  
df = pd.read\_excel("bisiklet\_fiyatlari.xlsx")  
  
# y = wx + b  
  
y = df["Fiyat"].values  
  
x = df[["BisikletOzellik1","BisikletOzellik2"]].values  
  
#veriyi ikiye bol  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size= 0.33 , random\_state= 15)  
  
print(x\_train.shape)  
print(x\_test.shape)  
print(y\_train.shape)  
print(y\_test.shape)  
  
#scalling(boyutunu degistirmek)  
#noronlara verilen veri setini kucultmek  
  
scaler = MinMaxScaler()  
scaler.fit(x\_train)  
x\_train = scaler.transform(x\_train)  
x\_test = scaler.transform(x\_test)  
  
#modeli olustur  
  
model = Sequential()  
#hidden layer kac tane is o kadar model.add(dense) yap  
model.add(Dense(5,activation = "relu")) # 5 noron sayisi, relu aktivasyon fonksiyonu tipi  
model.add(Dense(5,activation = "relu")) # 5 noron sayisi, relu aktivasyon fonksiyonu tipi  
model.add(Dense(5,activation = "relu")) # 5 noron sayisi, relu aktivasyon fonksiyonu tipi  
  
model.add(Dense(1)) #cikti noronu  
  
model.compile(optimizer = "rmsprop", loss = "mse") #calistirmak icin  
  
#train  
# x\_train' i y\_train ile egitecegiz  
  
model.fit(x\_train, y\_train, epochs = 250)  
  
#bize sozluk olarak vermis icine loss verdik dizi yapmak icin ,yukaridaki train edilen veriyi veriyor  
loss = model.history.history["loss"]  
  
#x ekseninde kac loss verisi var onu goster y de loss u goster  
sbn.lineplot(x = range(len(loss)), y =loss)  
  
plt.show()  
  
#modelin kendi degerlendirme modeli evaluate verbose islem detayi demek  
trainLoss = model.evaluate(x\_train,y\_train, verbose = 0)  
  
testLoss = model.evaluate(x\_test,y\_test, verbose =0)  
  
print(trainLoss)  
  
print(testLoss)

* 1. **Model Değerlendirilmesi**

import openpyxl  
import pandas as pd  
import seaborn as sbn  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import MinMaxScaler  
import tensorflow as tf  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  
  
  
df = pd.read\_excel("bisiklet\_fiyatlari.xlsx")  
  
# y = wx + b  
  
y = df["Fiyat"].values  
  
x = df[["BisikletOzellik1","BisikletOzellik2"]].values  
  
#veriyi ikiye bol  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size= 0.33 , random\_state= 15)  
  
print(x\_train.shape)  
print(x\_test.shape)  
print(y\_train.shape)  
print(y\_test.shape)  
  
#scalling(boyutunu degistirmek)  
#noronlara verilen veri setini kucultmek  
  
scaler = MinMaxScaler()  
scaler.fit(x\_train)  
x\_train = scaler.transform(x\_train)  
x\_test = scaler.transform(x\_test)  
  
#modeli olustur  
  
model = Sequential()  
#hidden layer kac tane is o kadar model.add(dense) yap  
model.add(Dense(4,activation = "relu")) # 5 noron sayisi, relu aktivasyon fonksiyonu tipi  
model.add(Dense(4,activation = "relu")) # 5 noron sayisi, relu aktivasyon fonksiyonu tipi  
model.add(Dense(4,activation = "relu")) # 5 noron sayisi, relu aktivasyon fonksiyonu tipi  
  
model.add(Dense(1)) #cikti noronu  
  
model.compile(optimizer = "rmsprop", loss = "mse") #calistirmak icin  
  
#train  
# x\_train' i y\_train ile egitecegiz  
  
model.fit(x\_train, y\_train, epochs = 250)  
  
#bize sozluk olarak vermis icine loss verdik dizi yapmak icin ,yukaridaki train edilen veriyi veriyor  
loss = model.history.history["loss"]  
  
#x ekseninde kac loss verisi var onu goster y de loss u goster  
sbn.lineplot(x = range(len(loss)), y =loss)  
  
#plt.show()  
  
#modelin kendi degerlendirme modeli evaluate verbose islem detayi demek  
trainLoss = model.evaluate(x\_train,y\_train, verbose = 0)  
  
testLoss = model.evaluate(x\_test,y\_test, verbose =0)  
  
print(trainLoss)  
  
print(testLoss)  
  
#degerlendirme  
#x\_testi verecek bir tahmin yaptirdik x\_testi verip y\_testi tahmin etmesini istedik  
  
testTahminleri = model.predict(x\_test)  
  
print(x\_test)  
  
tahminDf = pd.DataFrame(y\_test,columns=["gercek y"])  
  
print(tahminDf)  
#kac data var bakmak icin  
print(testTahminleri.shape)  
#test tahminlerini series yaptik  
testTahminleri = pd.Series(testTahminleri.reshape(330))  
  
tahminDf = pd.concat([tahminDf,testTahminleri], axis=1)  
  
tahminDf.columns = ["gercek y", "tahmin y"]  
  
#print(tahminDf)  
  
sbn.scatterplot(x = 'gercek y', y= "tahmin y", data= tahminDf)  
  
plt.show()

#degerlendirme hesaplamasi

Mean\_absolute\_error(tahminDF[“gerçek y”], tatminDf[“tahmin y”])

Mean\_squared\_error(tahminDF[“gerçek y”], tatminDf[“tahmin y”])

Print(df.describe())

#kendi urunumuzu verip fiyatini tahmin ettiriyoruz

yeniBisikletOzellikleri = [[1750,1780]]

yeniBisikletOzellikleri = scaler.transform(yeniBisikletOzellikleri)

model.predict(yeniBisikletOzellikleri)

from tensorflow.keras.models import load\_model

model.save(“bisiklet\_model.h5”)

sonradanCagrilanModel = load\_model(“bisiklet\_model.h5”)

print(aonradanCagrilanModel.predict(yeniBisikletOzellikleri))

[**https://www.kaggle.com/datasets**](https://www.kaggle.com/datasets) **(hazır data seti bulmak için)**