**PANDAS (Panel Data System)**

**Import pandas as pd**

**dir(pd) pandastaki gomulu fonksiyonlari gosterir**

Numpy eksikliklerini gidermek icin gelistirilmis bir kutuphane

Gorsellestirme yapmamiz, raporlar cikarmamiz, veriyi temizlememiz gerekebiliyor.

Veriyi modelleme icin, makine ogrenmesi ya da deepL icin hazirlamamiz istenebilir

Ya da veriyi anlamlandirip powerBI veya tableau’ya sokmamiz gerekebilir

Ikisi icin de pandas kullaniyoruz.

**Numpy ve Pandas farklari:**

-Numpy homojen tutuyordu, pandas’ta oyle degil. Homojen de olur heterojen de.

-Pandasta, sutun ve satir indekslerini istedigimiz gibi isimlendirebiliyoruz.

-Numpy’in temeli array’ler iken, Pandas’ta ise, seriler ve data frame var. Data frame daha cok

-Daha yuksek performansli ve daha esnek

-Pandas daha cok ekonomi ile ilgili kodlanmaya baslanmis once

Panel data tipi var bir de:

-Numpy’da veriler daha az yer kaplamasi icin veriler homojendi

**Pip install pandas**

**Import pandas as pd**

**Series**: 1D- tek boyutlu array’lere benziyor. Homojen data aliyor.

**Data frame:** 2D- heterojen data alabiliyor. 2 boyutlu

**Panel:** 3D arka arkaya paneller seklinde (3D array gibi)

**Series:**

Belirli index degerlerine sahip, belirli homojen data 1D

**Icine liste vererek series olusturabiliriz.**

**Pandas.Series(data, index,dtype, copy)**

**Pd.series ([85,90,70,80]), name= ‘marks’**

**85,90.. da value olarak gecer.**

S=pd.Series(np.random.randn(5), index=[“a”, “b”, “c”, “d”, “e”] diyebiliriz mesela

Default 0’dan baslar ama istedigimiz degeri verebiliriz index olarak

Series’e de isim verebiliriz.

Numpy’da array olusturdugumuzda, index sayilarini getirmez.

Pandas’ta 0,1,2,3 gibi index’lerin numaralarini da gostererek value’lari gosterir.

|  |  |
| --- | --- |
| 0 | x |
| 1 | y |
| 2 | z |
| 3 | a |
| 4 | b |

**Data Frame**

Series’lerin birlesmesinden data frame olusuyor denebilir.

Index’ler de gosterilir basinda. 2D

Bir sutununda int bir sutununda str data tipinde **heterojen** data alabilir

|  |  |  |
| --- | --- | --- |
| 0 | x | 12 |
| 1 | y | 5 |
| 2 | z | 7 |
| 3 | a | 8 |
| 4 | b | 1 |

Iki dict formatinda yazdigimiz seriyi toplayarak, data frame olusturulabilir.

**Df=pd.DataFrame(d)**

Eger birinde 3 index digerinde 4 index olan 2 seriyi birlestirmek istersem bunu pandasta yapabiliyorum.

Arraylerin boyutlari birbirine uymadiginda, concatenate yaparken sorun cikiyordu np’de

Ama pandasta bunu yapabiliyoruz. NaN dedigimiz bilinmeyen bir degerle dolduruyor.

Index’ler default olarak 0,1,2,3 ama a,b,c,d olarak da isimlendirebiliyorsun

Ya da index’e 100, 101, 102 seklinde atama yapabilirsin

\*Python’da degisken isimlerine foo bar baz isimleri verilebilir. calisma notlari olusturmak icin bir isimlendirme olustururlar

**Scalar**: agirlik gibi sicaklik gibi degerler

**3 yontem:**

1. Dict item’lardan series olusturmak

**data={‘a’:0., ‘b’:1., ‘c’:2.}**

**S=pd.Series(data)**

Burda indexler a,b,c olur, value’lar 0.0, 1.0, 2.0

2.

**data={‘a’:0., ‘b’:1., ‘c’:2.}**

**S=pd.Series(data, index=[‘b’,’c’,’d’,’a’])**

Bu da d indexine NaN atayarak yapar.

3.

**S=pd.Series(5, index=[0,1,2,3]**

bu mesela tum indekslere 5 atayan bir seri

**Indexing and slicing**

s[a] yazinca a’ya karsilik hangi deger varsa o gelir

S[[‘a’, ‘c’, ‘d’] a,c,d. indexleri getirir.

Birden fazla degeri getirirken cift parantezle kullaniyorum

Ser[[0,2,4]] seklinde cagirdigimda, o indeksleri getirir.

Ser[[‘1st’, ‘3rd’ ‘5th’]] seklinde de yazabilirim.

Ser[ser>3] yazdigimda True False seklinde dondurur.

Ser[2:]=100

Dedigimde, 2. indeksten sonrasini 100 yap demek

Ser+ser gibi bunlari da toplayabiliyorum

Ser/2 gibi bolebiliyorum. Ortaya hep float deger cikar bolmede

Array’lerde kullandigimiz methodlari bunda da kullanabiliyoruz

**Ser.mean()**

**Np.mean(ser)** diyerek de ortalama alabiliyoruz

Index label’lar ayni isimde de olabilir

Oyle atarsan oyle olur.2 tane a, 2 tane b gibi

A’dan 2 tane atadin index olarak diyelim

1. Indexi cagirirsan 2 value’yu da verir

Ilkini cagirmak icin yanina[0] dersin, ikinciyi cagirmak icin [1] dersin

Iki seriyi birbiri ile carparken, degerlerden biri Nan ise carpim da NaN olacak

**Positional index** (sr[2])

**Label index** [‘India’]

**Negative index:** [-1] dedim mesela

**Slicing:** [1:3] 1’den basla 3’e kadar git demek,

**Row index**

**Column index**

Bir kosul ile de indeksleme islemi yapabilirz

Sutun>6 diyince sarti saglayanlari getiriyor.

**Data frame append:**

Append methodu ile data frame’leri birbirine ekleyebilirim.

Sutun indexleri farkli olsaydi, NaN olarak doldurararak ekler

**reindex**

**Fill\_value=black** dedim mesela NaN degeleri black ile doldur

**Method=’ffill’** bir oncekiyle dolduruyor

**Describe -omrun boyunca unutma!**

Veri hakkinda genel istatistiksel veri

**Creating Pandas Series**

**1.Using list**

**2.Using NumPy Arrays**

**3.Dicts**

**4.Scalar value**

**1.USING LISTS**

|  |
| --- |
| **Pandalar serisi tek boyutlu bir veri yapısıdır.**  **Nesneler, objects, floats, strings ve integers dahil olmak üzere birçok türde veri tutabilir.**  **pandas.Series() öğesini çağırarak bir seri oluşturabilirsiniz.**  **Bir liste, numpy array ve dictionary bir pandas serisine dönüştürülebilir.**  **Axis labels (eksen etiketleri) toplu olarak index olarak adlandırılır.**  **Unutmayalım ki pandas serilerinin numpy array lerinden farkı, değerleri indexleriyle beraber tutar.** |

-seriler label degerleri ile birlikte gelir

-pd.Series seklinde yazilir.

-data tipi float doner

-bos serilerde data tipi object olacak

-arr=np.array([1.5, 2,"a"]) #arrayin tipini Unicode32 olarak verdi

Arr

type(seri) #bu bize serinin tipini verir.

#homojen yapiya donusturmek icin hepsini ayni data tipine dondu

array(['1.5', '2', 'a'], dtype='<U32')

#bundan farkli olarak serilerde, ortak tip olarak object tip alir

#int ve str birlikte icinde yer aldiginda numpy’da hepsini str’ye cevirirdi.

#serilerde numpy'dan farkli olarak icindeki elemanlarin orjinal tipleri korunuyor

#ama en kapsayici tip object oldugu icin, serinin tipini homojen olabilmesi icin object yapti.

seri=pd.Series([1.5, 2, 'a'])

Seriyi cagirdigimda data tipi object olur.

#float ve int birlikte oldugu zaman, float kapsayici oldugu icn float verir

#sonradan tipini degsitirme istersem ise astype ile degistiririm.

**Index tanimlama**

seri=pd.Series(d, index=["b", "c", "a", "d", "e"]) #indekslere label denir.

seri

#once label'i yani indexi getirecek, value'larini da yaninda getirecek

# ne kadar value’m varsa, ayni sayida index girmem lazim. (bu dict’lerle series olusturmada boyle degil, o zaman NaN atayacak)

#ndexler de unique olmak zorunda degil, gruplandirma avantaji sagliyor bu bize.

1. **Using NumPy Arrays**

**arr=np.arange(5) #1'den 5'e kadar bir array olusturduk tek boyutlu**

**pd.Series(arr) #bu arrayden de series olusturduk**

**pd.Series(arr) # seriler tek boyutlu, Series’in icine tek boyutlu array vermem lazim**

1. **Using dicts**

#sozluklerden de seriler olusturabiliyorum. dict'teki key ve value degerlerini vererek

**data={"ali":20, "veli":30, "mehmet":40}**

**seri=pd.Series(data)**

Seri

#ornegin seriyi cagirirken, Mehmet yerine index’i ahmet dedim

# eger index'ten farkli bir index atayarak cagirirsam, value'su olmayan

#index'e NaN atayarak getirdi.

#NaN eksik veri dedigimiz bir kavram. Missing value.

#cesitli tekniklerle data manipulasyonu yapmamiza olanak sagliyor

#**seri=pd.Series(data, index=["ali", "ahmet", "veli", "ayse"])**

seri

#listelerle series olustururken index daha az/fazla verince hata vermisti,

#ama dict ile series olustururken, NaN atayarak series olusturuyor hata vermeden

**4.Using Scalar Value**

Skalerler, skalerden oluşan liste veya demet gibi veri yapılarının aksine, bir tamsayı veya bool gibi bir veri birimini temsil eden tek değerlerdir.

Veri skaler bir değerse, bir dizin sağlanmalıdır.seri=pd.Series(10, index=["a", "b", "c"])

seri=pd.Series(10, index=["a", "b", "c"])

Seri

Bu mesela bize, label’lari a,b,c olan herbir degeri 10 olan bir series olusturur:

a 10

b 10

c 10

dtype: int64

pd.concat([seri,seri])# bu sekilde yazdirdigimda, serileri birlestiriyor

**## Indexing and Slicing with Pandas Series**

seri= pd.Series([1,2,3,4], index=["Audi", "BMW", "Volvo", "Mercedes"])

Seriyi cagirdigimda asagidaki gibi getirir.

Audi 1

BMW 2

Volvo 3

Mercedes 4

dtype: int64

|  |
| --- |
| seri[3] #bu sekilde 3.indeksini getiriyor. Label'dan farkli olarak  seri.index[3] #indeksin icine 3 girerek cagirdigimda 3. indekse ait olan labeli getirir.  seri["Audi"] #bu da Audi'nin degerini getir demek  seri[2:] #2.indeksten baslayarak tum hepsini getirir. Label+value olarak  seri[::-1]#tersten getirir  seri[[0,2]] #2 tane deger getirmek istiyorsam. 2 koseli parantez kullanirim  seri[["Audi"]] # bunu 2 koseliye alirsam, hem label degerini hem value'sunu verir  seri[["Audi", "Mercedes"]] #bu sekilde yazdigimda da degerleri labellari ile getirir  seri["Audi": "Mercedes"] #slicing yaparken de tek koseli parantez ve : kullandik  # farkli olarak burda, son index olan Mercedesi de getirdi |

seri.head #ilk yazacagimiz verisetinde bu koddur.

**#seri.head()**  bu şekilde 5 satır getirir eğer biz paranteze başka bir sayı yazmassak default değeri 5

**#seri.tail()** son 5 satırı getirir yine parantez içine herhangi bi sayı yazmassak

#seri.tail(2) sondan 2 tane getirir

**DATA FRAMES**

**Creating a DataFrame**

1. **Lists**
2. **Arrays**
3. **Dicts**

DataFrame, iki boyutlu bir veri koleksiyonudur.

Verilerin tablo şeklinde (tabular) saklandığı bir veri yapısıdır.

Veri kümeleri satırlar ve sütunlar halinde düzenlenir; veri çerçevesinde birden çok veri kümesi depolayabiliriz.

DataFrame'i, aynı dizini paylaşmak için bir araya getirilmiş bir dizi Series nesnesi olarak düşünebiliriz.

Veri çerçevesine sütun/satır seçimi ve sütun/satır ekleme gibi çeşitli aritmetik işlemleri gerçekleştirebiliriz.

DataFrame'leri harici depolamadan içe aktarabiliriz; SQL Veritabanı, CSV dosyası ve bir Excel dosyası.

**1.Creating a DataFrame Using the Lists of Data & Columns**

**data= [1,2,3], [4,5,6]**

**columns=["A", "B", "C"]**

**df=pd.DataFrame(data=data, columns=columns)**

**df**

**#icine data ve columns vererek olusturuyorum**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **C** |
| **1** | **2** | **3** |
| **4** | **5** | **6** |

**data= [["a",2,3.5], ["b",5,6.5]]**

**columns=["A", "B", "C"]**

**df=pd.DataFrame(data=data, columns=columns)**

**df**

**#bu sekilde A,B,C sutunlarina yukaridaki verileri atayarak dataframe olusturdum**

df.apply(type) #apply func ile de type'ini cagirirsin

**2.Creating a DataFrame Using a Numpy Arrays**

**arr=np.arange(1,27,3).reshape(3,3)**

**arr**

**df= pd.DataFrame(data=arr)**

**df**

#bu sekilde 3’e 3, yani toplam 9 elemandan olusan 2 boyutlu bir df olusturdum

df=pd.DataFrame(arr, columns=["A1","A2","A3"])

Df

#Dedigim zaman, columns’un adini, "A1","A2","A3" olarak degistirmis olurum.

df= pd.DataFrame(arr, columns=["A1","A2","A3"], index=["B1", "B2", "B3"])

Df

##Dedigim zaman, indexi de "B1", "B2", "B3" yapmis olurum.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** |
| **B1** |  |  |  |
| **B2** |  |  |  |
| **B3** |  |  |  |

**3.Creating a DataFrame Using a Dictionary**

**data={"Name" : ["Ali", "Veli", "Ahmet"], "Age" :[20,30,40]}**

df=pd.DataFrame(data)

df

**## Basic Attributes & Methods of DataFrames**

**df.head(2)** #default'u 5ti. 2 yazinca 2 tanesini getirdi

**df.tail(1)** #sondan bir tane getirir

**df.sample(2)** #bu da bize 2 deger getirir.

**df.columns** #serilerden farkli olarak columnsu cagirabiliyorum

**for i in df.columns:** #columns icinde dolas bana getir demis oldum

print(i)

Makine ogrenmesi numerik verilerle calisir, mesela diyecegiz ki, column’lar icinde dolas

Bana numeric olanlari getir diyecegim.

**df.Age**  #df'in age'ini getirdi

**df.Age.mean()** #age ortalamalarini getirdi, bolme islemi varsa degerler float olarak gelir

**df.Age.min()**

**df.["Age"].min()**

#bu sutunun icinde iki kelime olsa, arada bosluk olsa da

#[""] bu sekilde cagirirsam hata vermeden calisir.

**df.columns=["isim", "yas"]** # bu columnsun adini degistirdi

**df.index=["a", "b", "c"]** # bu da indekslerin adini degistirdi

**df.rename(columns={"isim": "x", "yas": "y"})**

#bu sekilde yaparsam column'larin adini x,y olarak degistirdi

#df'yi yeniden cagirdigimda eski haliyle gelir. df func'u degistirmez rename normalde. Cunku inplace default degeri:False

**df.rename(columns={"isim": "x", "yas": "y"}, inplace=True)** dersem kalici olarak degistirir.

#defaultu False idi, yani kalici degisiklik yapmiyordu. True dersem kalici degistirmesini saglarim.

df.shape mesela (3,2) olsun

**df.shape[0]** #0 deyince 3'u,

**df.shape[1]** #1 deyince 2'yi verir.

#yani 0 satir sayisini, 1 ise sutun sayisini verir

**df.ndim** #2 boyutlu zaten DataFrames

**df.size** #icindeki eleman sayisi

**"Name" in df**

##datasetinde olup olmadigini ogrenebiliyorum bu sorgulamayla