**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

**PANDAS**

**APPEND, CONCAT, MERGE, JOIN**

|  |
| --- |
| **\*\*append() ve concat()\*\* ile dikey birleştirme yapabiliriz.**  **\*\*concat(), join() ve merge()\*\* ile yatay birleştirme yapabiliriz.**  **\*\*append() ve concat()\*\* ile aynı anda iki veya ikiden fazla dataframe i birleştirebiliriz.**  **\*\*append() ve concat()\*\* da ortak sütun olması gerekmez.**  **\*\*concat()\*\* ile hem satır hem de sütun bazlı birleştirme yapabiliriz. Yani "axis" sadece concat() de var.**  **\*\*join()\*\* de aynı anda iki veya daha fazla dataframe i birleştirebiliriz ancak aynı sütun isimleri varsa en fazla iki df birleştirebiliriz.**  **\*\*merge()\*\* ile aynı anda iki dataframe i birleştirebiliriz.**  **\*\*merge()\*\* ile birleştirme işlemimizi ortak sütun üzerinden yapıyoruz.** |

#concat, merge, append ve join methodlarini gorecegiz.

#merge'u kullanmadan yapamayacaklarimiz var

#hem append hem concat'la yapilabilenler var.

#satirlari alt ata getirerek dikey birlestirme yapmis oluyuoruz

#append dikey, merge, join, concat ise yatay birlesme

#birden fazla df birlestirebiliyoruz.

#ayni sutun ismi yoksa merge ile yapamiyoruz.

#merge'de ancak sutunlar ayni ise birlestirebiliyor.

#concat, append ortak sutun olmasa da yapiyor.

#karsilik gelen deger yoksa nan ile dolduruyor bazilari

**APPEND (Vertical)**

append() fonksiyonu, iki veri kümesini birleştirmek için kullanılan en temel yöntemdir.

#append alir vertical birlestirir. yani satirlari altalta birlestirir vertical olarak

**one.append(two)**

#boyle yazarsam vertical bir birlestirme yapiyor

#en temel birlestirme bu

#2 kere ayni index'le deger veriyor.

#ama bu karisikliga sebep oluyor

#bunu duzeltmem gerek

#ignore\_index=True diyince 0’dan baslayip yeniden sirali index veriyordu, bu sekilde duzeltebilirim.

**new\_df=one.append(two, ignore\_index=True)**

#append ile ayni anda birden fazla df ekleyebilirsin

#kalici degisiklik yapmasi icin atama yapman gerek

**CONCAT (AXIS=0,1)**

concat(), iki veya daha fazla DataFrame'i satırlar veya sütunlar bazında birleştirmek için kullanılır.

Farklı veri kümelerini aynı veri kümesinde bir araya getirmek veya veri kümesine yeni sütunlar veya satırlar eklemek için kullanılabilir.

Veri kümeleri arasında herhangi bir ortak sütun gerekmez.

**pd.concat([one, two])**

**#pd.concat ([x,y])**

#concat alt alta hepsini koyuyor, olmayan degerlere de nan koyuyor. Default axis=0

#concat yazdigimiz axis'e gore birlestirir.

#axis=0 yazdigimizda vertical birlestirir concat. appendle ayni

#concat'da axis=1 yazdigimiz zaman horizontal birlestirme yapar, satirlari yanyana yazar

#concat'da inner ve outer var.

#innerj iki df’nin ortak sutunun indeksi alip getiriyor

#outerjoin ise hepsini alip getiriyor. karsiliginda deger yoksa nan atar.

#append'den farki concat'in axisle kullanilmasi

**pd.concat([one, two], ignore\_index=True, axis=1)**

#burda yanyana siraladik, kullanisli bir gorunum olmadi

#vertical daha mantikli burda

#istedigimiz satir ve sutunlari cagirtabiliyoruz iloc ile

**pd.concat([one.iloc[ :,:2], two.iloc[:,1:]])**

#one'in tum satirlari ile 2.sutuna kadar olan degerleri getir dedik

#two icin de, tum satirlari getir ve 1.sutundan sonuna kadar getir dedik

#one icin 2. indeksi dahhil etmedigimiz icin sutunda nan yapti

#two icin de 0. indeksi sutunda cagirmadigimiz icin nan yapti.

**MERGE (horizontal, HOW, ON)**

merge(), iki veya daha fazla veri kümesini, ortak bir sütuna göre birleştirme işlemidir.

Ortak sütunlardaki değerlere göre veriler birleştirilir ve tek bir veri kümesi oluşturulur.

merge(), veri kümesindeki eksik veya farklı değerleri tamamlamak veya farklı veri kaynaklarını bir araya getirmek için sıkça kullanılır.

#merge'de **how** parametresi var. **left right inner ve outer** var how’a cvp olarak

#ortak bir sutun olmasi lazim iliskilendirmesi icin

##**on** parametresine o sutunu yaziyorsun, sql'deki gibi

#merge'de **horizontal** yanyana siralama yapacak.

#bu da ortak sutun uzerinden gidecek

#kullanimi da concat gibi ama **ON** ile ortak sutunu yazarak birlestiriyoruz.

#merge yaparken foreign key gibi iliskilendirebilecegi bir sutun uzerinden birlestiriyor

#iki df birlestirdim int degerler var diyelim, nan degerler gelecegi/geldigi icin int degerler de float'a donusur.

**pd.merge(x, y, how="inner", on="key")**

#on ile key yazmasak da anliyor aslinda ortak sutunun key oldugunu

#horizontal bicimda sutunlari yanyana birlestirdi

#zaten merge vertical birlestirme yapmiyor

#how'a inner dedigimiz icin K3'u almadi

#cunku K3 ortak degil, digerleri ortakti sadece

**pd.merge(x, y, how="outer")**

#outer diyince bu sefer K3'u de getirdi, ama x'ten deger gelmedigi icin K3 sutununun karsiligna nan koydu

**pd.merge(x, y, how="left")**

#left ile birlestirince, x'i aliyor. X’i left’e yazdim cunku

#x ile y'nin ortak degerleri getiriyor bir de

**pd.merge(x, y, how="right")**

#right ile birlestirince, y'yi aliyor. Right’da y var cunku

#x ile y'nin ortak degerleri getiriyor bir de

**#left\_on, right\_on** gibi methodlari var

**pd.merge(df1, df2, left\_on="lkey", right\_on="rkey", how="inner")**

#left\_on'a ilk yazdigimiz yani soldaki df'in sutununu

#right\_on'a da 2. yani sola yazdigmiz df'teki iliski kuracagimiz sutunu yazariz

#y, a, b sutunlari ortak olmadigi icin onlari almadi

#cunku inner dedik how icin.

#x icin 3 deger var,ikisi ilkinden, biri 2.den geliyor

#x'i alir 4. deger olarak yine 3. degerini yazar. 8

#outer deseydik hepsini birlestirirdi

#merge'de sadece 2 df birlestiriyoruz.

**JOIN(Horizontal, leftjoin, rightjoin, innerjoin, outerjoin)**

**df3.join(df4)**

**#how condition'in default hali merge'de inner**

**#join'in default hali ise left**

#yani lefti alir, sagdakiyle ortak olani da alir.

#horizontal sekilde birlestirir

**leftjoin** 1.nin hepsini getiriyor.

#**rightjoin** 2.nin hepsini getiriyor. ilkinin sadece ortak olanlarini getiriyor

#**innerjoin** sadece ortak olani getiriyor

#**outer join** hepsini getiriyor.

#ortak sutunlar varsa isimlerini A1 gibi farkli veriyor.

#outer yapsaydik, hepsini birlestirecekti.

#ama karsiligina value denk gelmezse nan doldurur.

**df3.join(df4, how="right")**

#right yapinca da sagin(df4) hepsini alir, ortak olanlari da alir, ortak olanlari getrir

#bos olan yerlere nan doldurur

**df3.join(df4, how="outer")**

#outer yapinca tum satirlari alir

**df3.join(df4, how="inner")**

#inner'da her zaman kesisimini getirdigi icin hic null deger gelmez

**df4.join(df3)**

#boyle yaparsam da once C ve D gelir sonra A ve B

#yazim sirasina bagli

**#df5.join(df6)** boyle yazarsak hata verir. ortak sutunlar var der joinle birlestirirken

#lsuffix ve rsuffix leri kullanarak birlestirebilirim:

**df5.join(df6, lsuffix= "\_left", rsuffix="\_right")**

**df5.set\_index("key")**

#set index napiyordu. bir sutnu alip df'in indexi yapiyordu

#key sutunun yazarsam onu index yapar, boylece sorunsuz birlestirebilirim.

**df6.set\_index("key")**

#ikisine de ayni islemi yapinca artik sorunsuz birlestirebiliyorum

**TEXT AND TIME DATA**

**Text Data - String Methods**

|  |
| --- |
| **\*\*str.lower():\*\* Bir stringi küçük harfe dönüştürür**  **\*\*str.upper():\*\* Bir stringi büyük harfe dönüştürür**  **\*\*str.capitalize():\*\* İlk karakteri büyük harfe dönüştürür**  **\*\*str.title():\*\* Her kelimenin ilk karakterini büyük harfe dönüştürür**  **\*\*str.swapcase():\*\* Büyük harfleri küçük, küçük harfleri büyük harfe dönüştürür**  **\*\*str.isalpha():\*\* Tüm karakterlerin alfabede yer alıp almadığını kontrol ederek True veya False değeri döndürür**  **\*\*str.isnumeric():\*\* Tüm karakterlerin sayısal değer içerip içermediğini kontrol ederek True veya False değeri döndürür**  **\*\*str.isalnum():\*\* Tüm karakterlerin harf veya sayısal değer içerip içermediğini kontrol ederek True veya False değeri döndürür**  **\*\*str.endswith():\*\* Bir stringin belirtilen değerle bittiğini kontrol ederek True veya False değeri döndürür**  **\*\*str.startswith():\*\* Bir stringin belirtilen değerle başladığını kontrol ederek True veya False değeri döndürür**  **\*\*str.contains():\*\* Bir alt dizinin bir dizede bulunup bulunmadığını kontrol ederek her bir öğe için True veya False değeri döndürür**  **\*\*str.strip():\*\* Bir stringin başındaki ve sonundaki boşlukları temizler ve kırpılmış bir versiyonunu döndürür**  **\*\*str.replace():\*\* Belirtilen bir değeri başka bir değerle değiştirilmiş bir string döndürür**  **\*\*str.split():\*\* Bir stringi belirtilen ayraçta böler ve bir liste döndürür**  **\*\*str.find():\*\* Bir stringde belirtilen bir değeri arar ve bulunduğu konumun indeksini döndürür**  **\*\*str.findall():\*\* Bir desendeki tüm örneklerin bir listesini döndürür**  **\*\*str.join():\*\* Bir iterable ın öğelerini bir stringe dönüştürür** |

**#butun df uzerinde islem yapiyorsak str. olarak yaziyoruz fonksiyonu**

#yani df'in tum satir ve sutunlarinda islem yapmak istedigimde str kullanmam gerek.

**df.job.str.isalpha()**

#alfebedeki degerlerden mi olusuyor diye kontrol ediyor

#bosluk oldugu icin False verdi

**df.age.str.isnumeric()**

#neden NaN geldi, age sutununda **-** isareti oldugu icin, age sutununu object kategorisinde goruyor.

#o yuzden int gormedigi icin nan verdi.

#ben age sutunun kategorisini object'den str'ye cevirirsem, o zaman check edebilirim

**df.age.astype("string").str.isnumeric()**

# **“-”** bir str ifade oldugu icin, sutunu int'e degistiremedik.

#o yuzden sutunu komple str'ye cevirdik. Sonra **str.isnumeric** mi diye check ettik

**df.sales.str.isalnum()**

#hem alpha hem numeric birlikte kontrol eder isalnum

#icerdigi karakterler hem numeric hem alfabetikse True doner

#$ isareti dolayisisyla False verdi digerlerine.

#$'i ozel karakter olarak aliyor alfabetik olmayan

#sadece space tespit eden **isspace** var mesela

**df.sales.isascii()**

#ascii karaktere sahip olanlara True dondurur bu da

**mystr = 'ABC'**

**print(mystr.isascii())**

(Adem hocamin netten buldugu kullanim. Str’siz kullanilmis.

str.isascii yapinca hata veriyor)

**df.job.str.startswith("f")**

#neyle basladigini kontrol etmek icin kullanilir

#mesela f harfi ile baslayanlari getir gibi

#bool type donduruyor hep bunlar

**df.job[df.job.str.startswith("data")]**

#boyle yazarsak da bool type degil dogrudan degerleri verir

**df.job.str.endswith("st")**

#st ile bitenleri getir

**df.job.str.contains("data")**

#icinde data contain edenleri verir

**df[df.job.str.contains("data")]**

#koseli paranteze alip basina df yazarsan, o kosulu saglayan degerleri getirir.

#datayi temizlemek icin kullandiklarim: strip, lstrip, rstrip

**df.sales.str.strip("$, dolar")**

#bu mesela $ ve dolar ifadelerini temizledi.

#yukardakinin aynisini yapiyor.

**df.sales.str.rstrip("dolar").str.lstrip("$")**

#sales sutununda, sag tarafta isime yaramayan dolari, sol taraftaki $ isaretini temizliyor

**ya da**

**df.sales.str.strip("dolar$")**

#tum bunlar dolar ve $’dan kurtariyor datalari

**df.sales.str.strip("dolar$").replace(",", "")**

#replace yazarsam boyle str yazmadan, aradan "," secip silmez.

#bazilarinda , var bazilarinda yok ya hepsini tek tip hale getirmeye calisiyorum ki int'a cevireyim

**df.sales.str.strip("dolar$").str.replace(",", "")**

#str.replace yazarak, dogrudan datanin icine girip duzenleme yapabildim.

#artik virgullerden kurtulmus oldum

#ama dtype hala object bunu astype ile float'a cevirebilirim.

**df.sales=df.sales.str.strip("dolar$").str.replace(",", "").astype("float")**

**df**

#bu sekilde temizlenmis ve float'a cevrilmis datayi yeni bir degiskene atadim

#boylece degismis oldu df.

#birden cok method girince icine, inplace=True olmasina ragmen kalici degisiklik yapmaz.

##ozetle dolar ve $ yazilarini attim, virgulleri attim, floata cevirdim, yeni bir degiskene atayarak kalici hale getirdim.

#age'deki - isaretinden de kurtulmak istiyorum ki float'a cevirebileyim

#-'yi null yaparsam float'a cevirebilecegim. Cunku NaN’in tipi float.

**df.age.replace("-", np.nan)**

**df.age=df.age.replace("-", np.nan)**

#str.replace df’in tum datalari uzerinde islem yapar.

#yazdigimiz kosulu tum df'nin datalarina uygulatabiliriz. Datanin icine girerek donusturtebiliriz.

#replace ise tek bir satir, ya da tek bir deger uzerinde islem yapar. Komple degistirir.

#mesela arada virgul var diyelim, bunu replace ile yapamam.

#ama tek bir deger var ve onu komple baska bir seye degistirmek istiyoprum o zaman ancak replace yapabilirim.

**df.name.str.title().str.split().str[0]**

#bunu title'a cevirip strip ile ayirip, 0. index'teki elemani alirsam isim kismini

#boylece ad soyad olarak ayirabilirim

**df.name.str.title().str.split().str[1]**

#1. indeksi alirsam da soyadini alir.

**df["first\_name"]=df.name.str.title().str.split().str[0]**

**df["last\_name"]=df.name.str.title().str.split().str[1]**

#Kalici olmasi icin yeni degerlere atadim

#artik name sutununu dusurmem gerek

**df.drop("name", axis=1, inplace=True)**

#artik kalici olarak name droplandi

**df.job.str.find("developer")**

#icine yazdigimiz seyi arayacak ve degerin index degerini dondurur

#bulamadiklari icin -1 dondurur

#buldugu yerin indexini verir.

#9 demek d"nin ilk basladigi yerin indeksi demek

#frontend developer kelimesinde 9. indekste basliyor "developer"

**df.job.str.findall("er")**

#finall ise, indexi deger degeri donduruyor,

#yani buldugu ifadeyi liste icinde bize getirdi

#yoksa bos liste getiriyor.

**df.job.str.findall("d")**

#find index getirirken findall deger getiriyor liste icinde

**df.job.str.findall("d").count()**

#burda count kac tane satir varsa onu saydi, d”yi saydirmadi.

#peki ben kac tane d oldugunu nasil bulacagim

#count, len gibi ifadeler hepsini sayar.

#ama apply'i veya transformu kullanirsak, herbir satirda islem yapacagi icin

#herbir satirda d'yi kac tane bulduysa onu yazar apply ve transform(len) ile

**df.job.str.findall("d").apply(len)**

#apply olmadan len veya count kullanirsak tum satirlari sayar getirir

#ama ben satirlar bazinda calissin istiyorsam, her satirda ne kadar var bana onu getirsin istiyorsam **apply ile len** kullanirim.

**df.skills.str.join(",")**

#herbirini alsin virgulle ayirsin liste disinda birlestirsin istiyorum

#artik liste icinde degil, disari cikardi

#kelimeler , ile birbiriyle birlesmis halde cikti.

#df'de bazi sutunlarda liste icinde bazi sutunlarda ayri ayri tableu, ptyhon, sql var.

#tableu, ptyhon, sql ayri bir sutun olsun istiyorum:

#liste ise onu al join kullan , ile birlestirerek listeden cikar,

#degilse aynen birak diyecegim.

**df.Skills.apply(lambda item: ",".join(item) if type(item)==list else item)**

#if yapisi kullanarak, dedim ki tipi listeyse , ile birlestirerek listeden cikar

#listede degilse(else) aynen getir

#tableu, ptyhon, sql ayri bir sutun olsun istiyorum nasil yaparim?

#df'de Skills sutununda liste icinde ve disinda datalar var.

#oncelikle onlari tek tipe cevirmem lazim

#liste ise onu al join kullan , ile birlestirerek listeden cikar,

#degilse aynen birak diyecegim.

**df.Skills.apply(lambda item: ",".join(item) if type(item)==list else item)**

#if yapisi kullanarak, dedim ki tipi listeyse , ile birlestirerek listeden cikar

#listede degilse(else) aynen getir

**DUMMY OPERATIONS**

**get\_dummies(), makine öğrenimi ve veri analizinde yaygın olarak kullanılan bir işlevdir.**

**Genellikle kategorik değişkenleri veya text datalarını, makine öğrenimi algoritmaları tarafından kolayca işlenebilen sayısal veya binary dönüştürmek için kullanılır.**

#get\_dummies()

#kategorik sutunlari sayisala donusturmek icin kullanilir.

#makine ogrenmesinde kullaniyoruz daha cok

#makineye bilgileri verip, anlamli sonuclar cikarmasini saglamak icin kullaniyoruz

#kategorik degerleri alir tek basina sutun yapar

#daha sonra da bunu numerik hale cevirir

#M'nin oldugu yere 1 yazdi, olmayan yerlere 0 yaziyor

#bunu L ve S icin de yapiyor.

#binary degerler 0 yok 1 var anlaminda, buna donusturmus oluyor

**pd.get\_dummies(df.dept)**

Dept sutununda, HR ve IT vardi 5 satir seklinde. Bunu HR ve IT sutun olacak sekilde bir df yapip, HR’in bulundugu satirlara 1, bulunmadiklarina 0 yazdi. Aynisini IT icin de yapti. Boylece kategorik bir degere donusturmus oldu.

**pd.get\_dummies(df.dept, drop\_first=True)**

#get\_dummies’lerde 2 kategorik data varsa, bunu 1’e cevirmeliyim. 3 varsa 2’ye.

#o yuzden ilk satiri dusuruyoruz.

**df.Skills.str.get\_dummies(sep=",")**

#sonra Skills sutununa geldi, ona get dummies yapti.

#, separatorunu kullanarak da, Python, SQL, Tableu gibi skillsleri ayri sutunlar halinde gelmesini sagladi.

#Orjinal halinde, tabloda yanyana virgulle siralanmis halde bulunuyordu bazilari (Python, Tableau, SQL) gibi

**skills\_dummy = df.Skills.str.get\_dummies(sep=",").add\_prefix("skills\_")**

**skills\_dummy**

#Skills sutununa git, datalarindan get dummies ile yapay kategorik veri olustur.

#aralari virgulle ayrili birden fazla data varsa, onlari ayri sutunlar olarak separate et.

#add\_prefix ile de “skills\_” ismini koy baslarina sutunlarda.

#bunu da skills\_dummy gibi yeni bir degiskene atayarak kalici hale getir.

**df\_final=df[["dept", "job", "sales", "Skills"]]**

#df\_final ile birlestirmemiz gerekecek simdi skill\_dummy'i:

**df\_final=df\_final.join(skills\_dummy)**

#Skills sutunun get dummies ile kategorik veriye donusturdugum icin

#artik onu droplamam gerekecek:

**df\_final.drop("Skills", axis=1, inplace=True)**

#Skills’in ilk sutununu drop first True ile dusurmus olduk.

#df\_final ile skills\_dummy’i join ile birlestirmis olduk.

#df\_final olarak yeni bir degiskene atayarak da kalici hale getirdik

#df\_final'i get dummies yapiyoruz simdi:

**pd.get\_dummies(df\_final, drop\_first=True)**

#sales sutunu sayisal oldugu icin onu kategorik data haline getiremedi.

**IMPORT AND EXPORT DATAFRAME**

**Simdi bu uzerinde calistigim temizleyip kategorik hale getirdigim df\_final df’ini csv dosyasi olarak kaydediyorum. Export:**

**df\_final.to\_csv("df\_final.csv")**

#burdaki df'yi export ederek, csv olarak kaydettim.

#daha sonra buna kolayca ulasabilecegim

**df\_new=pd.read\_csv("df\_final.csv")**

**df\_new.head()**

#read ile okuttum ve ilk 5 satirini cagirttim

#unnamed olarak gelen indexleri (sutun) dan kurtulmam gerek

**df\_new=pd.read\_csv("df\_final.csv", index\_col=0)**

**df\_new.head()**

#bu sekilde ilk index'teki index numaralarini gosteren sutundan kurtulmus oldum

#index\_col'dan sonra hangi index nosunu yazarsam, o sutunu atar. ayni seyi dropla da yapabilirim.

**df\_html=pd.read\_html("https://en.wikipedia.org/wiki/List\_of\_countries\_by\_population\_(United\_Nations)")**

#bu da bir html dosyasinin icindeki datayi okuyup getiriyor.

#ama anlamli uzerinde calisabilecegim bir formatta getirmiyor bu

**table=pd.read\_html("https://en.wikipedia.org/wiki/List\_of\_countries\_by\_population\_(United\_Nations)")**

#bu sekilde table’a atarsam table olarak getirir.

**df\_country=table[0]**

#bu sayfa icindeki ilk tabloyu alir

**#1 yazarsam varsa 2. tabloyu getirir**

**df\_country.head()**

#df\_country tablosunu bu wikipedia adresinden cekmis bir df olusturmus oldum

#head ile de ilk bes satirni getirdim tablonun

**Time Data**

#zaman serilerinde kullanacagiz daha cok

**df=pd.read\_csv("time\_exercise.csv")**

**df.head()**

**df["entry\_date"]=pd.to\_datetime(df["entry\_date"])**

#datetime modulu sadece gunler uzerinde islem yapiyor

#tipi objecti, datetime'a cevirdi tipini

#artik tarihler uzerinde islem yapabilirim.

**df["order\_date"]=pd.to\_datetime(df["order\_date"])**

#order\_date’e de aynisini uyguladik

**df.entry\_date.min()**

#ilk tarihi gostermis oldu. depoya ilk giren

**df.entry\_date.max()**

#depoya son giren urunun tarihini bulabilirsin

**df.entry\_date.max()-df.entry\_date.min()**

#Max ve min tarihlerin farklini aliyor

**df.entry\_date.dt.year**

#dt-sadece yillari gostersin istersem **dt.year** yaparim

#day var, week var farkli kullanimlari var

**df.entry\_date.dt.month**

**Not:#max baslibasina bir method yaninda () kullandik o yuzden**

**#ama month year'i dt ile kullandigimiz icin parantez kullanmadim**