**Pandas Veri Analizi (Panel Data)**

Numpy bazı özellikleri itibari ile yetersiz kalmaktadır. Numpy üzerinde vektörel işlemleri(matematiksel ve istatistiksel) kolaylıkla yapabiliyorduk. Ancak veri analitiği, veri işleme ve veri madenciliği konularında yetersiz kalmaktadır. Numpy’de veri tiplerinin tek tipte olması gerekiyordu. Farklı türde veriler üzerinde işlem yapılmak istendiğinde pandas bu işlevi çok iyi bir şekilde karşılayacaktır. Yapılacak modelleme işlemleri pandas ile yapılmaktadır. Pandas, numpy kütüphanesinin bir alternatifi değildir. numpy özelliklerini kullanır ve bunları genişletir. Numpy dizilerine array(vektör ve matrisler) diyorduk, pandas dizileri ise tek boyutlu olanları seriler(series) iki boyutlu olanlar ilse Veri Çerçeveleri(DataFrames) olarak adlandırılır.

Pandas kütüphanesi veri analizi için yazılmış bir python kütüphanesidir. 2008 yılında geliştirilmeye başlanmıştır. Farklı veri tipleri ile çalışılabilir. Numpy arraylarinden en belirgin farkı sütunlar isimlendirilebilir, satır indis numaraları değiştirilebilir, farklı veri tiplerini içerisinde barındırabilir.

Pandas kütüphanesini projemize dahil etmek için

**import pandas as pd**

komutu kullanılmalıdır. Ancak pandas kütüphanesi ile beraber numpy kütüphanesinin de mutlaka projeye dahil edilmesi gerekmektedir. Çünkü kullanacağımız pandas serilerini genellikle numy array’lerinden üretmeye çalışacağız.

**Pandas Serisi Oluşturma**

Pandas serileri oluştururken veriler üzerinden, python dizilerinden ya da numpy array’lerinden faydalanabiliriz. Bir pandas seri’si oluşturmak için

pd.Series(["data"],["index"])

şeklinde bir tanımlama yapmak gerekmektedir. index değerleri verilmezse otomatik olarak 0,1,2 şeklinde indis değerleri atanacaktır.

**pd.Series([1,2,3,4,5,6])**

**0 1**

**1 2**

**2 3**

**3 4**

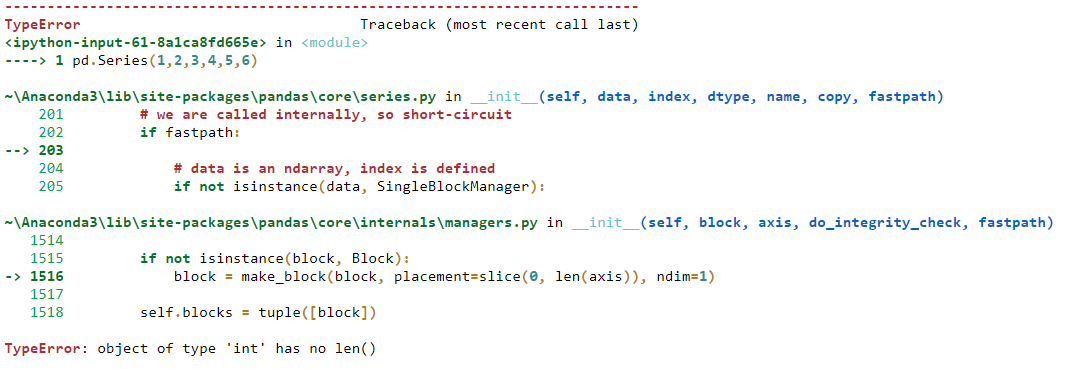
**4 5**

**5 6**

**dtype: int64**

yöntemini kullanabiliriz. Dikkat edileceği üzere aynı numpy arraylerini oluşturduğumuz gibi iç içe hem normal parantez hem de köşeli parantezi beraber kullandık. Aksi halde hata ile karşılaşacaktık. Ayrıca oluşturulan seri’nin altında veri tipi de(int 64) yazmaktadır.

**pd.Series(1,2,3,4,5,6)**



Yine numpy’de olduğu gibi ([]) parantez köşeli parantez yapısıyla beraber iç içe iki parantez (()) yapısı kullanılarak da pandas seri’si oluşturulabilir.

Serileri oluştururken sadece sayısal değerler değil string ifadelerden de faydalanabiliriz.

**seri=pd.Series(["ankara","istanbul","izmir","antalya","bursa"])**

**seri**

**0 ankara**

**1 istanbul**

**2 izmir**

**3 antalya**

**4 bursa**

**dtype: object**

**Liste üzerinden Seri Oluşturma**

Pythonda oluşturduğumuz listeleri bir pandas serisine dönüştürebiliriz. Normalde python listelerinin indis numaraları vardır ancak bunları listeleme işlemi yaptığımızda göremeyiz. Listeyi seri’ye dönüştürdüğümüzde ise indis değerleri listeyle beraber görülecektir.

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

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

**seri**

**0 1**

**1 2**

**2 3**

**3 4**

**4 5**

**5 6**

**dtype: int64**

görüldüğü üzere seri elemanları ile beraber indis değerleri de listelenmiştir.

Liste veri tipi zaten köşeli parantez içerdiği için ayrıca iç içe parantez köşeli parantez yapısı kullanmaya gerek yoktur.

**Numpy Array’i Üzerinden Seri Oluşturma İşlemi**

Bir array üzerinden seri oluşturmak için tıpkı listelerde yaptığımız işlem gibi önce array oluşturup array üzerinden seri oluşturma işlemi yapabiliriz.

**a=np.arange(1,6)**

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

**seri**

**0 1**

**1 2**

**2 3**

**3 4**

**4 5**

**dtype: int32**

**Biçimlendirme İşlemleri**

Oluşturduğumuz pandas seri’lerinin **type** fonksiyonu ile yapısına bakmak istersek,

**seri = pd.Series([1,2,3,4,5,6])**

**type(seri)**

**pandas.core.series.Series**

şeklinde bir veri tipine ait olduğunu görebiliriz.

Hatırlanacağı üzere numpy’de array ile ilgili özelliklere dtype(veri tipi), ndim(boyut sayısı), shape(boyut bilgisi) ve size(eleman sayısı) methodları ile yapıyorduk. Pandas’ta bu işlemi yapmak için de benzer metodlar kullanabilriz.

* axes metodu seri’nin özellikleri görüntülenebilir.

**seri.axes**

**[RangeIndex(start=0, stop=6, step=1)]**

Görüldüğü üzere axes metodu ile başlangıç, bitiş ve atlama değerine ulaşmış olduk.

Seri içersindenki değerlerin veri tipini öğrenmek istesek

**seri.dtype**

**dtype('int64')**

seri’nin kaç boyutlu olduğunu öğrenmek istersek ndim metodu kullanabiliriz.

**seri.ndim**

**1**

Eleman sayısına ulaşmak için size metodu kullanılabilir.

**seri.size**

**6**

Shape meto ile boyut bilgisine ulaşabiliriz.

**seri.shape**

**(6,)**

Seri üzerindeki değerleri indislerden bağımsız görüntülemek istersek values parametresi kullanlmalıyız.

**seri.values**

**y([1, 2, 3, 4, 5, 6], dtype=int64)**

head metodu sayesinde en baştan istediğimiz kadar değer görüntüleyebiliriz.

**seri.head(2)**

**0 1**

**1 2**

**dtype: int64**

tail metodu ile sondan istediğimiz kadar değer görüntüleyebiliriz.

**seri.tail(3)**

**3 4**

**4 5**

**5 6**

**dtype: int64**

eğer head ve tail fonksiyonlarına parametre verilmezse ilk 5 ya da son 5 değeri verir.

**seri.head()**

**0 1**

**1 2**

**2 3**

**3 4**

**4 5**

**dtype: int64**

**Seriler Üzerinde İndeksleme işlemleri**

Array’lar üzerinde yaptığımız indexleme işlemlerini seri’ler üzerinde de yapabiliriz.

**seri=pd.Series([1,2,3,4,5,6])**

**seri[0]**

**1**

**seri[3]**

**4**

**Seri’lerde index değerlerini değiştirme**

Pandas seri’lerinde index’lerin değerini değiştirerek istediğimiz değerleri atayabiliyoruz. Normalde biz değer vermediğmizde index değerleri otomatik olarak indis(0,1,2..) değerlerine karşılık gelmektedir.

**seri=pd.Series((1,2,3,4,5),index=(0,2,4,6,8))**

**seri**

**0 1**

**2 2**

**4 3**

**6 4**

**8 5**

**dtype: int64**

pandas’ta yapacağımız index’leme sadece integer ifadelerle sınırlı değil float hatta string değerlerle bile index’leme işlemi yapılabilir.

**seri=pd.Series((1,2,3,4,5),index=("a","b","c","d","e"))**

**seri**

**a 1**

**b 2**

**c 3**

**d 4**

**e 5**

**dtype: int64**

hatırlanacağı üzere pandas serilerinin sözlük yapılarına benzediğini söylemiştik. Aynı sözlüklerdeki gibi elemanlara erişim işlemleri yapabiliriz.

**seri["b"]**

**2**

Seri’lerin index değerlerini string değerlerle değiştirmiş olsak bile indis değerleri üzerinden elemanlarına erişebiliriz.

**seri[0]**

**1**

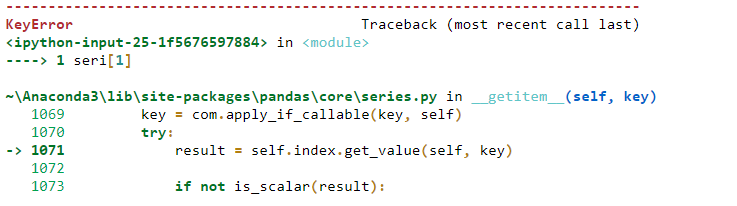
**seri[2]**

**3**

Ancak bu işlemi indis değerlerini string olarak tanımladığımız zaman gerçekleştirebiliriz.

**seri=pd.Series((1,2,3,4,5),index=(10,20,30,40,50))**

**seri[1]**



Resim 88

Bunun için ileride edetaylı deyineceğimiz loc ve iloc metodlarını kullanacağız.

**Sözlükler Üzerinden Seri Oluşturma İşlemleri**

Pandas serileri oluştururken sözlükler üzerinden seri oluşturma işlemleri sıkça kullanılmaktadır.

**sozluk={"a":1,"b":2,"c":3,"d":4,"e":5}**

**sozluk**

**{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}**

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

**seri**

**a 1**

**b 2**

**c 3**

**d 4**

**e 5**

**dtype: int64**

yine elemanlara ulaşmak istediğimizde index değerlerini yazmamız yeterlidir.

**seri["a"]**

**1**

**Seri’lere Erişim İşlemleri**

Seri’ler yapı itibariyle sözlükleri andırmaktadır. Hem index değerlerine hem de elemanlarına ulaşabiliriz. Elemanlarına ulaşmak için values metodu kullanabiliriz.

**seri=pd.Series([1,2,3,4,5,6]**

**seri.values**

**array([1, 2, 3, 4, 5, 6], dtype=int64)**

index’lerine ulaşmak için ise index metodunu kullanabiliriz.

**seri.index**

**RangeIndex(start=0, stop=6, step=1)**

Seri içerisindeki index ve değerlere ulaşabilmek için items metodu kullanılmalıdır.

**seri.items**

**<bound method Series.items of 0 1**

**1 2**

**2 3**

**3 4**

**4 5**

**5 6**

**dtype: int32>**

**Seri’lerde Dilimleme İşlemleri**

Dilimleme işlemleri yapmak istersek aynı şekilde : işareti ile başlangıç ve bitiş index isimlerini yazmamız yeterlidir.

**a=pd.Series([1,2,3,4,5,6])**

**a**

**0 1**

**1 2**

**2 3**

**3 4**

**4 5**

**5 6**

**dtype: int64**

burada dilimleme işlemi yapmak istersek

**a[0:3]**

**0 1**

**1 2**

**2 3**

**dtype: int64**

klasik dilimleme işlemlerindeki gibi başlangıç değerlerini aldı ancak bitiş değerini almadı. Şimdi seri’nin index değerlerini değiştirelim. Eğer serilerde index değerlerini farklı sayısal ifadelerle değiştirirsek, dilimleme işlemi yapmak istediğimizde boş değer döner.

**a.index=(10,20,30,40,50,60)**

**a[10:50]**

**Series([], dtype: int64)**

Bu durumda ileride daha detaylı göreceğimiz loc metodunu kullanmalıyız.

index değerine sayısal ifade değil de string değerler verirsek, dilimleme işlemini index değerlerine göre yapabiliriz.

**a.index=("a","b","c","d","e","f")**

**seri["a":"d"]**

**a 1**

**b 2**

**c 3**

**d 4**

**dtype: int64**

python ve numpy array’leri kullanırken dilimleme işlemleri yaptığımızda hatırlanacağı üzere başlangıç değeri alınıp bitiş değeri alınmıyordu, pandas’da ise indislerini değiştirdiğimiz serilerin son değeri de listelenmektedir. Yukarıda her iki örnekte de görüleceği üzere seri[10:50] işleminde “50” index değerine karşılık gelen değer, **seri["a":"d"]** işleminde denormalde biz **“d”** indexine karışık gelen değeri almayacağını beklerdik ancak burada değerin alındığını görmekteyiz.

Eğer indis değerlerinden erişmek istersek

**a[0:3]**

**seri[0:3]**

**a 1**

**b 2**

**c 3**

**dtype: int64**

görüldüğü üzere indis değerlerine göre dilimleme yapmak istersek alışık olduğumuz şekilde son değeri almayacaktır.

**Seri’lerde Birleştirme İşlemleri**

Birden fazla seriyi birleştirme işlemi yapabiliriz. Numpy’de array’leri birleştirmek için “**concatenate**” metodunu kullanıyorduk. Pandas’da ise bunun yerine “concat” metdou kullanılabilir.

**seri1=pd.Series((1,2,3,4),index=("a","b","c","d"))**

**seri2=pd.Series((5,6,7,8),index=("e","f","g","f"))**

**pd.concat([seri1,seri2])**

**a 1**

**b 2**

**c 3**

**d 4**

**e 5**

**f 6**

**g 7**

**f 8**

**dtype: int64**

ya da listelerde kullandığımız append metodunu kullanabiliriz.

**seri1.append(seri2)**

**a 1**

**b 2**

**c 3**

**d 4**

**e 5**

**f 6**

**g 7**

**f 8**

**dtype: int64**

**Seri’lerde Eleman İşlemleri**

Python’da bir değerin ya da ifadenin liste ya da karakter dizisi içerisinde olup olmadığını kontrol etmek için in operatörünü kullanıyorduk. Aynı şekilde bir değerin seri içerisinde olup olmadığını da in işleci ile kontrol edebiliriz. Burada yapılan kontrol index değerleri üzerinden yapılmaktadır.

**seri1=pd.Series((1,2,3,4),index=("a","b","c","d"))**

**"a" in seri1**

**True**

**"e" in seri1**

**False**

Eğer seride bulunan değeri değiştirmek istersek,

**seri1["a"]=5**

**seri1**

**a 5**

**b 2**

**c 3**

**d 4**

**dtype: int64**

sözlüklerde uyguladığımız metodu kullanabiliriz.

Seriler üzerinde erişim işlemlerini önceki bölümlerde görmüştük. Dilimleme işlemleri yaparken eğer index isimleri string ya da karakterlerden oluşacak şekilde değiştirildiyse yeni oluşturduğumuz index isimlerine göre değerlere erişebiliyorduk. Ancak 0,1,2 şeklinde indis değerleri görülmese de seri içerisinde bulunmaktadır.

**seri[0]**

**1**

**seri[1]**

**2**

İndex değerlerini değiştirdiğimiz serilere indis değerleri üzerinden dilimleme işlemi yaparsak alışık olduğumuz üzere yazdığımız son değer alınmamaktadır.

**seri[0:3]**

**0 1**

**1 2**

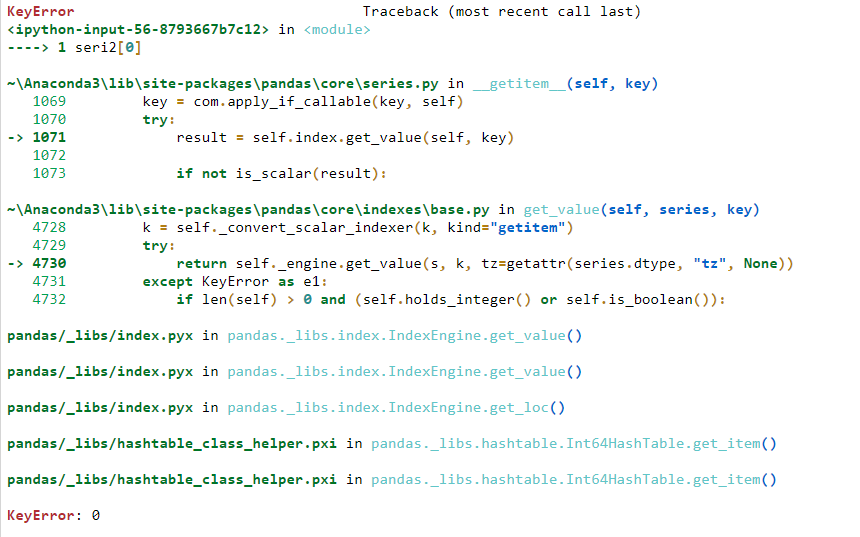
**2 3**

**dtype: int64**

**ancak index değerlerini string değil integer ifadeler olarak belirlediysek artık yeni indis değerleri bu verdiğimiz ifadeler olur ve bu şekilde indis değerlerine ulaşmak mümkün değildir.**

**seri2=pd.Series((1,2,3,4),index=(10,20,30,40))**

**seri2[0]**



Bizim elemanlara erişmemiz için verdiğimiz index değerlerini yazmamız gerekmektedir. Yani index’in tanımlandığı şekliyle çağırılması gerekmektedir.

**seri2[10]**

**1**

Aynı şekilde dilimleme yapmak istersek hata ile karşılaşmayız ancak geriye boş değer döner..

**seri2[10:30]**

**Series([], dtype: int64)**

Görüldüğü üzere dilimleme işlemini yapamadık. Oysaki seri2[10] şeklinde index değeri yazdığımızda ilgili elemana ulaşabiliyorduk. Pandas üzerinde değiştirilmiş index değerine ulaşma ve dilimle işlemleri bu yüzden farklı metodlarla yapılmaktadır. Eğer serimizi elemanlarına dilimleme işlemleri uygulamak istersek index değerleri üzerinden değil indis değerleri üzerinden erişmemiz gerekmektedir.

**seri2[0:2]**

**10 1**

**20 2**

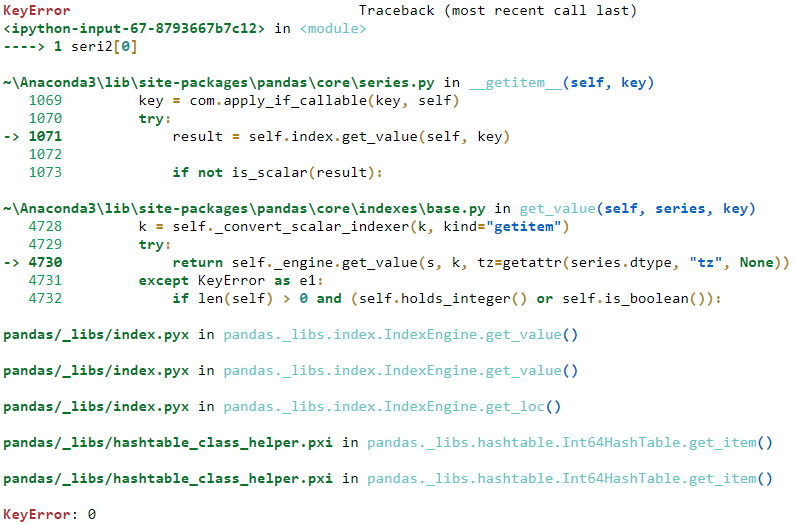
**dtype: int64**

**loc Metodu**

index değerleri değiştirdiğimiz ve sayısal ifade yaptığımız serilerde erişim ve dilimleme işlemlerini index numaraları üzerinden yapmak istersek loc(label based indexing) metodunu kullanmamız gerekmektedir. loc metodu serilerde eleman özelliklerine erişmek istediğimizde, değerin tanımlanmış şekliyle erişmemizi sağlamaktadır. Eğer index değerlerini değiştirerek indis değerleri üzerinden erişmek istersek hata ile karşılaşırız.

**seri2=pd.Series((1,2,3,4),index=(10,20,30,40))**

**seri2[0]**

****

Bu durumda yeni index değerini yazarak klasik metod ile ya da loc ifadesi ile değerlere ulaşabiliriz

**seri2.loc[10]**

**1**

**seri2[10]**

**1**

Dilimle işlemi yaparken de loc metodu kullanmazsak geriye boş değer dönderir

**seri2[10:30]**

**Series([], dtype: int64)**

**Bu durumda yine loc ifadesini kullanmalıyız**

**seri2.loc[10:30]**

**10 1**

**20 2**

**30 3**

**dtype: int64**

dikkat edilirse dilimleme işleminde verilen son değer de alınmıştır. loc metodu kullanırken index değerleri tanımlandığı şekilde çağırıldığı için dilimleme işleminde index’te olmayan aralık belirtilse bile hata ile karşılaşılmaz.

**seri2.loc[10:90]**

**10 1**

**20 2**

**30 3**

**40 4**

**dtype: int64**

**iloc metodu**

iloc metodu loc metodunun tam tersi işlem yapmaktadır. Seri’nin elemanlarına verdiğimiz indis değerlerini dikkate almadan ulaşmak istersek

**seri2=pd.Series((1,2,3,4),index=(10,20,30,40))**

**seri2.iloc[1]**

**2**

İloc metodu ile verdiğimiz index değerlerine göre ulaşmak istersek hata ile karşılaşırız.

Dilimleme işlemlerini yaparken index değerlerinin dikkate alınmadan indis değerleri üzerinden yapılmasını istersek iloc metodunu kullanmalıyız.

**seri2=pd.Series((1,2,3,4),index=(10,20,30,40))**

**seri2.iloc[0:2]**

**10 1**

**20 2**

**dtype: int64**

normal kullandığımız dilimleme metodu iloc metodu ile aynı işlemi yapmaktadır.

**seri2[0:2]**

**10 1**

**20 2**

**dtype: int64**

**Serilerde Koşullu İşlemler**

Array’lerde olduğu gibi seriler üzerinde de koşullu işlemler yapılabilmektedir.

**seri1=pd.Series((10,20,30,40,50,60))**

**seri1>30**

**0 False**

**1 False**

**2 False**

**3 True**

**4 True**

**5 True**

**dtype: bool**

görüldüğü üzere koşulu sağlayan değerler True, sağlamayan değerler ise False değeri almaktadır. Koşulu sağlayan elemanlara ulaşmak istersek,

**seri1[seri1>30]**

**3 40**

**4 50**

**5 60**

**dtype: int64**

şeklinde bir metod kullanılabilir.

Birden fazla koşulu sorgulamak istersek

**seri1[(seri1>15) & (seri1<45)]**

**1 20**

**2 30**

**3 40**

**dtype: int64**

**Serilerde Aritmetiksel İşlemler**

Array’lerde yaptığımız gibi Seri’lerle de basit aritmetiksel işlemler yapabiliriz.

**seri=pd.Series((1,2,3,4))**

**seri+5**

**0 6**

**1 7**

**2 8**

**3 9**

**dtype: int64**

**seri\*2**

**0 2**

**1 4**

**2 6**

**3 8**

**dtype: int64**

İki farklı seri üzerinde temel aritmetiksel işlemleri gerçekleştirebiliriz. Elimizde aynı meyvelerden değişik kilolarda olduğunu varsayarak bir seri oluşturalım.

**seri1=pd.Series([4,2,6,8,5],index=("elma","armut","üzüm","çilek","erik"))**

**seri2=pd.Series([5,6,1,3,7],index=("üzüm","elma","armut","erik","çilek"))**

**seri1+seri2**

**elma 9**

**armut 8**

**üzüm 7**

**çilek 11**

**erik 12**

**dtype: int64**

görüldüğü üzere index’lerin sırası aynı olmasa bile pandas aynı index’deki değerler üzerinde aritmetiksel işlemleri yapmaktadır. Bazı index değerleri birbirinden farklı olan seriler üzerinde aritmetiksel işlemler yapılmak istenirse bir seride olup diğerinde olmayan indexler NaN (Not a Number) değeri almaktadır.

**seri1=pd.Series([4,2,6,8,5],index=("elma","armut","üzüm","çilek","erik"))**

**seri2=pd.Series([5,6,1,3,7],index=("elma","armut","üzüm","çilek","kaysı"))**

**seri1+seri2**

**armut 8.0**

**elma 9.0**

**erik NaN**

**kaysı NaN**

**çilek 11.0**

**üzüm 7.0**

**dtype: float64**

NaN değerinin olma sebebi, bir değerin bir seride olup diğerinde olmamasından dolayı, seri içerisinde olan bir veri ile olmayan bir veriyi toplanmayacağından dolayı oluşmaktadır. Aslında burada akla gelen ilk soru olan değerle olmayan değeri 0 kabul edilerek toplanabileceğidir. Ancak veri analizinde eksik bilgilerin üzerinde bizden habersiz işlem yapılması hatalı sonuçlar üreteceği için bu şekilde işlem yapılmadan çıktı vermesi çok önemlidir.

**Serilerde Özel Metodlar**

Pythonda alışık olduğumuz sum() max() min() gibi fonksiyonları kullanarak veriler üzerinde çeşitli işlemler yapabiliriz.

* Sum() metodu ile toplama işlemi

Arraylerde kullandığımız gibi sum() metodu ile serinin değerlerini toplayabiliriz.

**seri=pd.Series([1,2,3,4,5,6,7,8])**

**seri1.sum()**

**25**

* Max() metodu ile serideki en büyük değeri bulabiliriz.

**seri.max()**

**8**

* Min() metodu ile serideki en küçük değeri bulabiliriz.

**seri.min()**

**1**

* mean() metodu ile serideki değerlerin ortalamasını bulabiliriz.

**seri.mean()**

**4.5**

**Read\_csv() metodu ile cvs dosyalarını okumak**

Seriler ve dataframelerle çalışırken cvs dosyaları sıkça kullanılır. CSV Virgülle ayrılmış değerler anlamına gelmektedir. Yani dosyanın içeriğinde aralarına virgül koyularak birbirinden ayrılmış metin veya rakamlar yer almaktadır. CSV uzantılı dosyalar not defteriyle de açılabilmesine rağmen onları doğru ve anlaşılır bir şekilde görüntülemek için Microsoft Excel veya Access gibi programlardan faydalanılmaktadır. Şimdi basit bir CSV dosyası oluşturalım. Bir not defteri dosyası açarak içerisine aşağıdaki değerleri alt alta yazalım, kaydetme işlemini yaparken farklı kaydet seçeneğini seçerek ismini şehirler.csv yaparak Kodlama türünü UTF-8 olarak bırakalım

Şehir

Ankara

İstanbul

İzmir

Antalya

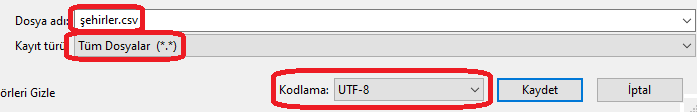
Hatay

Adana

Trabzon

Balıkesir

Aydın



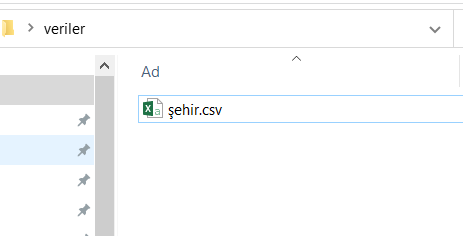
Burada ilk satırdaki değer olan “Şehir” değeri sütun ismi olacaktır. Biz burada tek sütun kullandığımız için pandas şehirlere sırasıyla indis değerleri atayacaktır. Eğer birden fazla sütun kullanırsak en soldaki sütun index değerleri olacaktır. Bunun haricinde eklediğimiz her virgülden sonraki değer yeni bir sütun olarak karşımıza çıkacaktır.

**pd.read\_csv("C:/Users/asus/Desktop/veriler/şehirler.csv")**



Resim 72

Elinizde daha önceden oluşturulmuş dosyalar olduğunu varsayarsak bunların içerikleri ile ilgili işlemler yapılabilir. Bunun için masaüstünde veriler isimli klasör içerisinde şehir.csv isim dosyanın olduğunu varsayarak

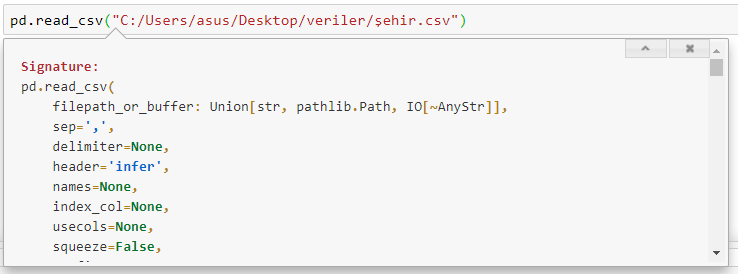


İlgili dosyaya erişmek için

**pd.read\_csv("C:/Users/asus/Desktop/veriler/şehir.csv")**



Şeklinde dosyaya erişim sağlayabiliriz. Read\_csv metodunun parametrelerini öğrenmek için parantez içinde “shift+tab” tuş kombinasyonuna basabiliriz.



squeeze parametresine True değeri verilerek değerin indis numaraları görüntülenebilir.

**pd.read\_csv("C:/Users/asus/Desktop/veriler/şehir.csv",squeeze=True)**

**0 Ankara**

**1 İstanbul**

**2 İzmir**

**3 Antalya**

**4 Hatay**

**5 Adana**

**6 Trabzon**

**7 Balıkesir**

**8 Aydın**

**9 Muğla**

**10 Kayseyi**

**11 Afyon**

**Name: Şehir, dtype: object**

Head() metodu kullanılarak üst kısımdan değerler görüntülenebilir. Eğer head() metoduna parametre girmezsek ilk 5 değeri görüntüler. Bunun için okuduğumuz dosyadan dönen değeri bir değişkene atayalım.

**veri=pd.read\_csv("C:/Users/asus/Desktop/veriler/şehir.csv",squeeze=True)**

**veri.head()**

**0 Ankara**

**1 İstanbul**

**2 İzmir**

**3 Antalya**

**4 Hatay**

**Name: Şehir, dtype: object**

Head() metodunun içerisine parametre girerek daha fazla değer görüntüleyelim.

**veri.head(8)**

**0 Ankara**

**1 İstanbul**

**2 İzmir**

**3 Antalya**

**4 Hatay**

**5 Adana**

**6 Trabzon**

**7 Balıkesir**

**Name: Şehir, dtype: object**

Tail() metodu ile head() metodunun yaptığı işlemin tam tersini yaparak sondan görüntüleme işlemi yapar.

**veri.tail()**

**7 Balıkesir**

**8 Aydın**

**9 Muğla**

**10 Kayseyi**

**11 Afyon**

**Name: Şehir, dtype: object**

Size ve len fonksiyonları ile serinin eleman sayısını öğrenebiliriz.

**len(veri)**

**12**

veri.size

**12**

Sıralama işlemi yapmak için sorted() metodunu kulanabiliriz.

**sorted(veri)**

**['Adana',**

**'Afyon',**

**'Ankara',**

**'Antalya',**

**'Aydın',**

**'Balıkesir',**

**'Hatay',**

**'Kayseyi',**

**'Muğla',**

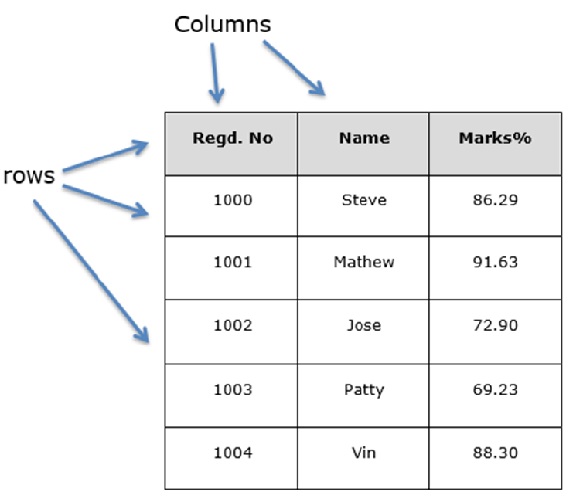
**'Trabzon',**

**'İstanbul',**

**'İzmir']**

**DataFrame ve Özellikleri**

DataFrame’ler serilerin birleşmesinden oluşan yapılardır. Genellikle farklı tipteki sütun ve satırlara sahip SQL tablolarına benzetilir. DataFrame’ler veriyi daha kolay işlememizi sağlarlar. DataFrame’ler genellikle veri tabanlarındaki tablolara benzetilmektedir. Columns(Sütunlar) ve index’lerden(satırlar) oluşmaktadır. Bu değerler üzerinden veri analizi yapılabilmektedir.



Bir DataFrame oluşturmak için,

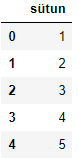
**pd.DataFrame([1,2,3,4,5])**



Resim7

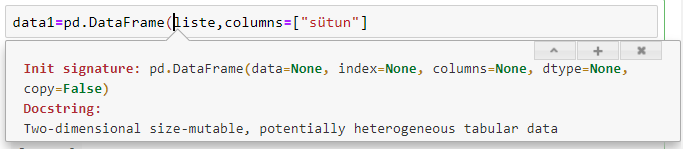
şeklinde bir tanımlama yapabiliriz. Burada istersek sütunlarımızın ismini de değiştirebiliriz.

**pd.DataFrame([1,2,3,4,5],columns=["sütun"])**



Resim 8

Görüldüğü üzere DataFrame’in sütun ismini değiştirmiş olduk. parantez içinde shift+tab tuş kombinasyonuna basarak DataFrame’in alabileceği parametreleri görebiliriz.



Resim 9

type() fonksiyonu ile DataFrame’in tipini incelersek,

**data1=pd.DataFrame([1,2,3,4,5],columns=["sütun"])**

**type(data1)**

**pandas.core.frame.DataFrame**

veri tipinin DataFrame olduğunu gördük.

axes metodu ile özelliklerini görüntülemek istersek,

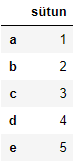
**data1.axes**

**[RangeIndex(start=0, stop=5, step=1), Index(['sütun'], dtype='object')]**

İndex değeri 0’dan başlayıp 5’te bitmiş ve birer birer artmıştır. Sütun bazında ise sütun isminin “sütun” olduğu görülmektedir.

Satır isimlerini belirlemek için ise index metodu kullanılmalıdır.

**pd.DataFrame([1,2,3,4,5],columns=["sütun"],index=["a","b","c","d","e"])**



Resim 42

Boyut sayısını öğrenmek için ndim metodu kullanabiliriz

**data1.ndim**

**2**

Boyut bilgisi için shape metodu kullanabiliriz.

**data1.shape**

(5, 1)

Eleman sayısını öğrenmek için size metodunu kullanabiliriz.(aynı işlemi len fonksiyonu ile de yapabiliriz)

**data1.size**

**5**

**Listelerden DataFrame Oluşturmak**

Elimizdeki bir listeyi kolaylıkla DataFrame’e dönüştürebiliriz.

**liste=[2,4,6,8]**

**pd.DataFrame(liste,columns=["sütun1"])**



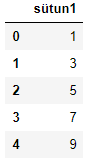
Resim 10

**Numpy Array’lerinden DataFrame Oluşturmak**

Array’leri DataFrame’lere dönüştürmek için tıpkı listelerde yaptığımız gibi array’leri de DataFrame’lere dönüştürebiliriz.

**a=np.array([1,3,5,7,9])**

**pd.DataFrame(a,columns=["sütun1"])**

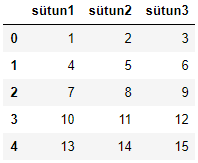


Resim 11

İki boyutlu array’i DataFrame’e dönüştürmek için 5 satır 3 sütunluk bir array oluşturalım

**b=np.arange(1,16).reshape(5,3)**

**pd.DataFrame(b,columns=["sütun1","sütun2","sütun3"])**



Resim 12

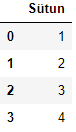
Burada array kaç sütunlu ise o kadar sütun ismi girilmesi gerekmektedir. aksi halde hata ile karşılaşırız.

**Pandas Seri’lerinden DataFrame Oluşturma**

Serilerden de DataFrame’ler oluşturabiliriz.

**data1=pd.Series([1,2,3,4])**

**pd.DataFrame(data1,columns=["Sütun"])**



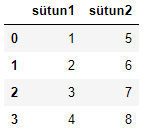
Resim 13

Birden fazla seri ile DataFrame oluşturmak istenirse sözlük yapısına benzer bir yapı kullanılmalıdır.

**data1=pd.Series([1,2,3,4])**

**data2=pd.Series([5,6,7,8])**

**pd.DataFrame({"sütun1":data1,"sütun2":data2})**

****

Resim 14

**Sözlüklerden DataFrame Oluşturma**

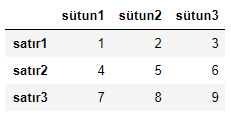
Önceki bölümde birden fazla seriden DataFrame oluştururken sözlük yapısına benzer bir yapı ile işlemimizi gerçekleştirmiştik, burada daha karışık bir yapı kullanarak DataFrame oluşturma işlemini gerçekleştirelim.

**data={"sütun1":{"satır1":1,"satır2":4,"satır3":7},**

**"sütun2":{"satır1":2,"satır2":5,"satır3":8},**

**"sütun3":{"satır1":3,"satır2":6,"satır3":9}}**

**pd.DataFrame(data)**



Resim 15

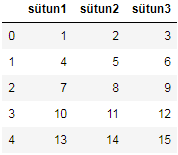
**DataFrame’deki Sütunları Değişkene Atamak**

DataFrame’lerdeki değerleri bir değişkene atayarak üzerinde işlemler yapabiliriz.

**a=np.arange(1,16).reshape(5,3)**

**df=pd.DataFrame(a,columns=["sütun1","sütun2","sütun3"])**

**df**



Resim 95

Sütun değerlerini değişkenlere atamak için

**x=df["sütun1"]**

**x**

**0 1**

**1 4**

**2 7**

**3 10**

**4 13**

**Name: sütun1, dtype: int32**

**type(x)**

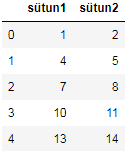
**pandas.core.series.Series**

x değişkeni seri olmuştur.

eğer bir değişkene birden fazla sütun atarsak veri türü DataFrame olur.

x=df[["sütun1","sütun2"]]

x



Resim 96

**DataFrame’lerin satır ve sütun bilgilerine erişme**

Elimizde bulunan DataFrame’e ait satır ve sütun bilgilerine ulaşabiliriz bunun için

**data=pd.DataFrame([1,2,3,4],columns=["sütun1"],index=["s1","s2","s3","s4"])**

elimizde yukarıdaki gibi bir veri seti olduğunu varsayalım. Satır bilgileri için

**data.index**

**Index(['s1', 's2', 's3', 's4'], dtype='object')**

Sütun bilgileri için

**data.columns**

**Index(['sütun1'], dtype='object')**

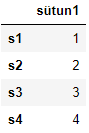
Şeklinde satır ve sütun bilgilerine erişebiliriz.

**DataFrame’in Satır ve Sütun İsimlerini Yeniden Adlandırma**

DataFrame’leri oluşturma işleminde satır ve sütun isimlerini belirlemek için

**data=pd.DataFrame([1,2,3,4],columns=["sütun1"],index=["s1","s2","s3","s4"])**

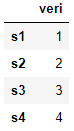
**data**



Şeklinde bir tanımlama yapmamız gerekmektedir. DataFrame oluşturduktan sonra satır ve sütun değerlerini değiştirebiliriz bunun için

**data.columns=["veri"]**

**data**



Resim 17

Satır isimlerini(index) değiştirmek için

**data.index=["satı1","satır2","satır3","satır4"]**

**data**



Resim 18

İşlemlerinin yapılması gerekmektedir.

**DataFrame Eleman İşlemleri**

**Satırlara(Gözlemlere) Erişim İşlemleri**

DataFrame’ler üzerinde indexleme işlemleri yapılabilmektedir. DataFrame’ler üzerinde bulunan satırlar gözlem olarak ifade edilmektedir.

**a=np.random.randint(10,size=6)**

**b=np.random.randint(10,size=6)**

**c=np.random.randint(10,size=6)**

**data=pd.DataFrame({"s1":a,"s2":b,"s3":c})**

**data**



Resim 19

Oluşturduğumuz DataFrame’de satır seçme işlemleri yapabilmek için,

**data[0:1]**

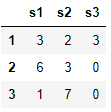


Resim 20

Yukarıdaki işlemle 1. Satırdaki verileri seçmiş olduk.

1,2 ve 3. Satırdaki değerlere ulaşmak için

**data[1:4]**



Resim 21

İşleminin yapılması gerekmektedir. Biz burada index isimlerini yazarak satırlardaki değerlere ulaşmak istersek hata ile karşılaşırız. Satırlara dilimleme işlemi yapmadan erişmek için loc ve iloc ifadelerinden faydalanmamız gerekmektedir. Eğer kendi verdiğimiz index isimleri ile ulaşmak istersek loc ifadesini orijinal indis değerleri ile ulaşmak istersek iloc ifadesini kullanmamız gerekmektedir. şimdi dataFrame’in index değerlerini değiştirelim.

**data.index=(10,20,30,40,50,70)**

**data**



Resim 44

Kendi verdiğimiz index değerlerine göre ulaşmak için

**data.loc[20]**

**s1 6**

**s2 7**

**s3 1**

**Name: 20, dtype: int32**

Varsayılan indis değerlerine göre erişebilmek için ise iloc ifadesini kullanmalıyız.

**data.iloc[1]**

**s1 6**

**s2 7**

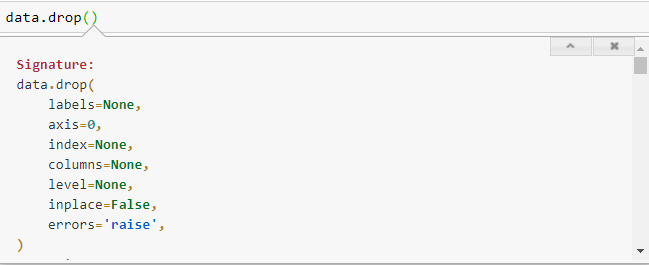
**s3 1**

**Name: 20, dtype: int32**

**Silme(drop) İşlemleri**

Oluşturduğumuz DataFrame’in satır ve sütunlarında silme işlemleri yapabiliriz. Bunun için drop parametresinin kullanılması gerekmektedir.

data.drop(“silinecek satır ya da sütun ismi”, axis=(satır silinecekse 0 sütun silinecekse 1 değeri verilmedilir)) drop metodunun aldığı parametreleri görmek için parantez içerisinde shift+tab tuş kombinasyonuna basalım



Resim 43

Drop metodunun varsayılan parametrelerine bakıldığı zaman axis değerinin 0 olduğu görülmektedir. Sadece satır ismi yazılarak istenilen satır silinebileceği gibi axis değeri 0 verilerek de silme işlemi yapılabilir.

**data.drop(2,axis=0)**



Resim 22

Görüldüğü üzere 2 numaralı satırdaki değerler silinmiştir. Sütun silme işlemi için ise

**data.drop("s2",axis=1)**



Resim 23

s2 isimli sütun silinmiştir. Burada axis değerinin doğru belirtilmesi çok önemlidir. axis değeri sütun silinmek istendiği zaman belirtilmez ya da satır silinirken 1 sütun silinirken 0 değeri verilmeye çalışılırsa hata ile karşılaşılır.

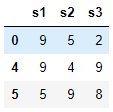


Resim 24

Eğer aynı anda birden fazla satır veya sütun silmek istersek

**a=[1,2,3]**

**data.drop(a,axis=0)**



Resim 28

Aynı işlemi sütunlar üzerinde uygulamak istersek

**a=["s2","s3"]**

**data.drop(a,axis=1)**



Resim 29

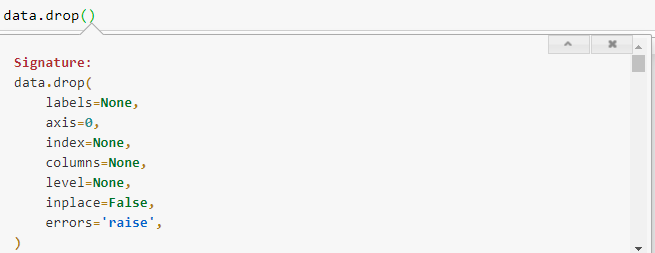
**Yapılan Değişikliklerin DataFrame’e Kaydedilmesi (inplace)**

Bu zamana kadar yaptığımız işlemlerde fark edildiği üzere yapılan işlemler DataFrame üzerinde bir değişiklik yapmamıştır. Yani drop metodu ile değerler silinmiş olsa bile DataFrame tekrar görüntülendiğinde değişikliğin DataFrame üzerinde uygulanmadığı görülecektir. Yani silme işleminden sonra DaraFrame tekrar görüntülenmek istenirse

**data**



Silme işlemleri uygulanmamıştır. Bunun için inplace metodunun kullanılması gerekmektedir. Drop metodunun alacağı parametrelere bakmak için tuş kombinasyonuna baktığımızda

****

Resim 26

İnplace değerinin varsayılan olarak False olduğu görülür. Bu özelliğin False olması çok önemlidir. çünkü elimizde bulunan DataFrame’ler üzerinde yapılan değişiklikler büyük veri kayıplarına neden olacağı için bu işlem kalıcı değişikliklerin önüne geçmektedir.

**data.drop("s2",axis=1,inplace=True)**

**data**



Resim 27

İnplace parametresine True değeri verdiğimiz için data isimli değişkende yapılan değişiklikler uygulanmış oldu.

**DataFrame’lerde Sütunlara(Değişkenlere) Erişim İşlemleri**

DataFrame’lerde sütun(değişken) ismi yazılarak değere ulaşılabilir.

**data={"sütun1":{"satır1":1,"satır2":4,"satır3":7},**

**"sütun2":{"satır1":2,"satır2":5,"satır3":8},**

**"sütun3":{"satır1":3,"satır2":6,"satır3":9}}**

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

**df["sütun1"]**

**satır1 1**

**satır2 4**

**satır3 7**

**Name: sütun1, dtype: int64**

Aynı işlemi tırnak işareti kullanmadan da yapabiliriz,

**df.sütun1**

**satır1 1**

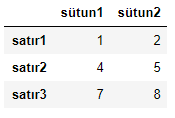
**satır2 4**

**satır3 7**

**Name: sütun1, dtype: int64**

Çoklu seçim işlemleri için sütun isimleri iç içe parantez içerisinde yazılmalıdır.

**df[["sütun1","sütun2"]]**



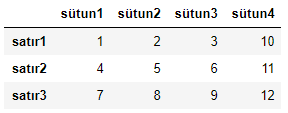
Resim 30

**DataFrame’lere Sütun(Değişken) Ekleme İşlemi**

DataFrame’lere sözlüklere eleman ekleme işlemi yaparken kullandığımız metodlardaki gibi eleman ekleyebiliriz.

**df["sütun4"]=10,11,12**

**df**

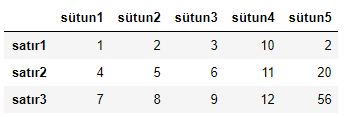


Resim 31

Bu işlemi yaparken oluşturacağımız yeni sütunu(değişkeni) diğer sütunların toplamı, farkı, çarpımı gibi aritmetiksel işlemlerin sonucu şeklinde belirleyebiliriz.

**df["sütun5"]=df["sütun1"] \* df["sütun2"]**

**df**



Resim 32

**Satır ve Sütunlara Birlikte Erişme İşlemi**

Biz daha önce satırlara erişim işlemlerini,

**a=np.random.randint(10,size=6)**

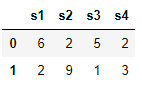
**b=np.random.randint(10,size=6)**

**c=np.random.randint(10,size=6)**

**d=np.random.randint(10,size=6)**

**data=pd.DataFrame({"s1":a,"s2":b,"s3":c,"s4":d})**

**data[0:2]**



Resim 33

Ve sütunlara erişim işlemlerini görmüştük,

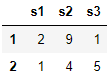
**data[["s1","s2"]]**



Resim 34

Satır ve sütunlara beraber erişmek için iloc metodu kullanılmalıdır.

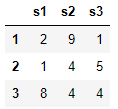
**data.iloc[1:3,0:3]**



Hatırlanacağı üzere iloc metodu bizim verdiğimiz satır ve sütun isimlerini göz ardı ederek indis değerleri üzerinden listeleme işlemleri yapıyordu. Köşeli parantez içerisinde verilen birinci parametre dilimleme yapılacak satırları virgül işaretinden sonraki ikinci parametre ise dilimleme işlemi yapılacak sütunları ifade eder.

Eğer loc ifadesi kullanılarak satır ve sütunlara birlikte erişilmek istenirse satır ve sütunlara verilen isimlere sadık kalarak yazılması gerekmektedir.

**data.loc[1:3,"s1":"s3"]**

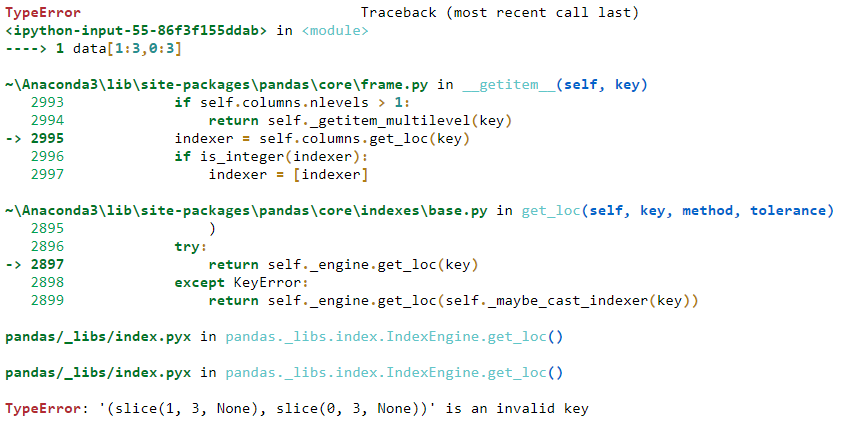


Resim 37

Hatırlanacağı üzere loc ifadesi kullanıldığı zaman yazılan başlangıç değeri ile birlikte bitiş değerini de dahil ediyordu.

Satır ve sütunlara birlikte erişilmek istendiği zaman iloc ve loc ifadesi kullanılmadan erişilmek istenirse hata ile karşılaşılır.

**data[1:3,0:3]**



Resim 38

Ya da dilimleme işlemleri yapmadan erişmek istersek

**data.loc[2,"s2"]**

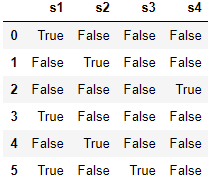
**1**

İşlemiyle satır ve sütunların kesişim yerlerine loc ifadesiyle erişebiliriz.

**DataFrame’lerde Koşullu İşlemler**

DataFrame’ler üzerinde serilerde yaptığımız gibi koşullu işlemler uygulayabiliriz. Örnek olarak DataFrame içerisinde 5’ten büyük değerler için bir koşullu ifade belirtelim

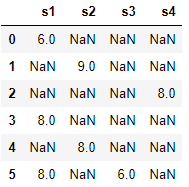
**data>5**



Resim 39

Tıpkı serilerdeki gibi

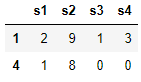
**data[data>5]**



Resim 40

Şeklinde bir işlem yapmak istersek 5’den büyük değerler görüntülenir ancak 5 ve küçük değerler NotaNumber(NaN) hatası vermektedir. Eğer seçim işlemlerinde kısıtlamaya gitmek istersek

**data[data.s2>5]**



Resim 41

Aynı işlemi

**data[data["s2"]>5]**

şeklinde de yapabiliriz

Koşul ifadesini s2 sütununda 5’den büyük değerler için belirledik, ancak DataFrame’de listeleme yaparken koşulu sağlayan tüm değerler listelemeye dahil edildiği için satır ve sütun değerleri görüntülenmiş oldu. Buna göre bir koşul daha yazarak yani sütun ismi yazarak arama işlemini daraltabiliriz.

**data[data.s2>5]["s2"]**

**1 9**

**4 8**

**Name: s2, dtype: int32**

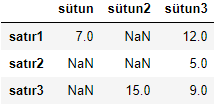
**Eksik Verilerin Tamamlanması**

Veri analizi yaparken elimize gelen veriler eksik olabilir. Bu konuyu sonraki bölümlerde detaylı inceleyeceğiz. Ancak öğrendiğimiz bilgiler eşliğinde eksik bilgileri tamamlama işlemi yapmayı deneyelim. Şimdi içerisinde eksik veriler olan 3 satır 3 sütunluk bir array oluşturalım

**a=np.array([[7,np.nan,12],[np.nan,np.nan,5],[np.nan,15,9]])**

**data=pd.DataFrame(a,index=["satır1","satır2","satır3"],columns=["sütun","sütun2","sütun3"])**

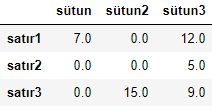
**data**

****

Resim 45

DataFrame’de işlem yapabilmek için nan olan değerlerin yerine belirlediğimiz değerleri vermemiz gerekebilir. Bu değerleri fillna metodu ile istediğimiz gibi değiştirebiliriz.

**data.fillna(value=0)**



Resim 46

Eğer nan değerlerinin yerine DataFramedeki diğer değerlerin ortalamasını bulmak yerleştirmek istersek. DataFrame’de bulunan her satırdaki değeri toplamak için sum() fonksiyonu kullanılmalıdır

**data.sum()**

**sütun 7.0**

**sütun2 15.0**

**sütun3 26.0**

**dtype: float64**

eğer

**data.sum().sum()**

**48.0**

Şeklinde bir tanımlama yaparsak tüm değerlerin toplamını bulma işlemi yaparız. Şimdi DataFrame’deki eleman sayısını bulalım,

**data.size**

**9**

Ancak buradaki değerlerin bazıları nan olduğu için değerlendirmeye katılmaması gerekmektedir. bunun için

**data.isnull().sum().sum()**

**4**

Şeklinde bir parametre kullanmamız gerekmektedir.

Şimdi ortalamayı bulmak için DataFrame’deki değerlerin toplamını eleman sayısına bölelim

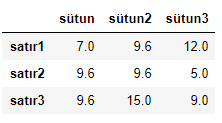
**ort=(data.sum().sum())/(data.size- data.isnull().sum().sum())**

**ort**

**9.6**

Artık bulduğumuz ortalama değerini eksik verilerin yerine koyabiliriz.

**data.fillna(value=ort)**



Resim 47

**DataFrame’leri Birleştirme İşlemleri:**

Hatırlanacağı üzere serileri birleştirmek için concat metodu kullanmıştık. DataFrame’leri birleştirmek için de concat metodu kullanabiliriz. Üç adet DataFrame oluşturalım.

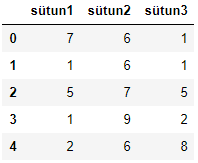
**a=np.random.randint(10,size=5)**

**b=np.random.randint(10,size=5)**

**c=np.random.randint(10,size=5)**

**data1=pd.DataFrame({"sütun1":a,"sütun2":b,"sütun3":c})**

**data1**

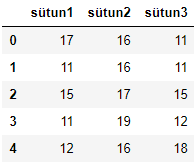
****

**Resim 73**

Data1 isimli DataFrame’den yararlanarak ikinci DataFrame’i oluşturalım.

**data2=data1+10**

**data2**



**Resim 74**

2. DataFrame 1. nin 10 fazlası oldu.

Şimdi concat metodu ile iki DataFrame’i birleştirelim, varsayılan değer axis 0 olduğu için DataFrame’leri alt alta birleştirecektir.

**pd.concat([data1,data2])**

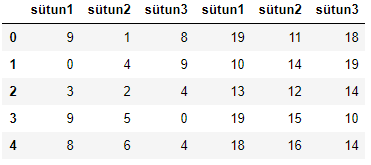


Resim 48

DataFrame’lerin her ikisinin de sütun isimleri aynı olduğu için alt alta toplama işlemi sorunsuz bir şekilde gerçekleşmiştir.

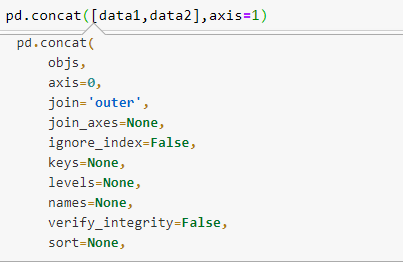
Eğer axis değerini 1 olarak belirtirsek birleştirme işlemi sütun bazında olacaktır. Her iki DataFrame’in index değerleri birbirine eşit olduğu için birleştirme işlemi sorunsuz bir şekilde gerçekleşecektir.

**pd.concat([data1,data2],axis=1)**



Resim 49

Her iki birleştirme işleminde de index ve sütun isimlerine baktığımızda birbirlerini tekrarladığını görüyoruz. Yani birleştirme işleminden sonra index veya sütun değerleri 4 den sonra 5 ile devam etmeyip tekrar 0’a dönmüştür. Bu durumun önüne geçmek için birleştirme işleminde **ignore\_index** metodu kullanılmalıdır. Concat metodunun aldığı parametrelere bakarsak **ignore\_index** parametresinin aldığı değerin False olduğunu görürüz.



Resim 50

Eğer bu parametreye True değeri verirsek

**pd.concat([data1,data2],axis=0,ignore\_index=True)**



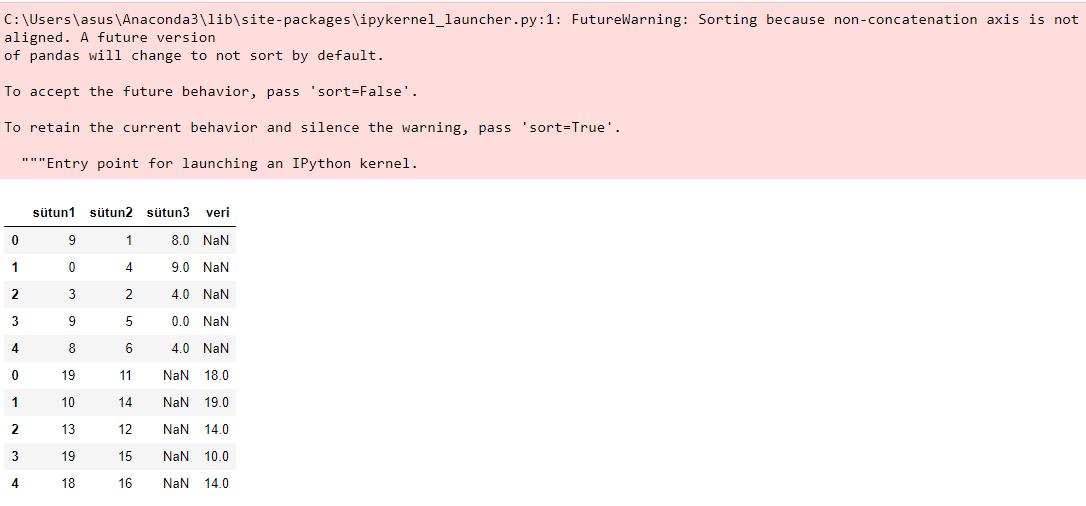
Resim 51

Burada index ve sütun isimleri aynı olduğunda sorunsuz bir şekilde birleştirme işlemini yapabildik. Eğer değişken isimlerinden bir ya da bir kaçı farklı olsaydı hata ile karşılaşacaktık

DataFrame’lerden birinin sütun isminin değişik olduğunu varsayarak tekrar işlem yapalım.

**data2.columns=["sütun1","sütun2","veri"]**

**pd.concat([data1,data2],axis=0)**



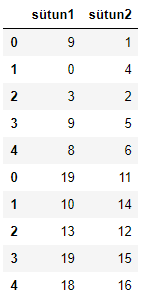
Resim 52

Sütun isimlerinin tamamı aynı olmadığı için ismi farklı olan sütun(veri) ayrı bir sütun olarak değerlendirilmiş ve DataFrame’in sonuna eklenmiştir. Ayrıca ilk değerler NaN olarak tanımlanmıştır.

**Join metodu** her iki DataFrame’de aynı isimde olan sütunları birleştirip farklı isimde olanları dikkate almaz.

**data2.columns=["sütun1","sütun2","veri"]**

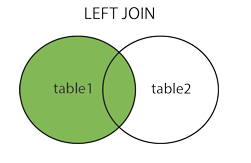
**pd.concat([data1,data2], join="inner")**



Resim 53

**Join Metodu ile Birleştirme İşlemi**

Join metodu DataFrame işlemlerinde concat metodundan ayrı olarak sıkça kullanılan metodlardan birisidir. Sql sorgularında kullanılan left join benzeri bir birleştirme işlemi yapar. Join metodu ile **index** değerlerine göre birleştirme işlemi yapabiliriz.



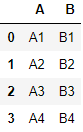
Resim 55

**data1={"A":["A1","A2","A3","A4"],**

**"B":["B1","B2","B3","B4"]}**

**df1=pd.DataFrame(data1)**

**df1**



Resim 75

**data2={"C":["C1","C2","C3"],**

**"D":["D1","D2","D3"]}**

**df2=pd.DataFrame(data2)**

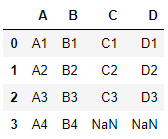
**df2**



**Resim 76**

Burada birleştirme işleminde sol taraftaki table DataFrame1’i sağ taraftakini ise DataFrame2 olarak düşünürsek, table1’deki tüm index’ler ile yeni bir DataFrame oluşmuş olacaktır. Burada data1 4 adet index değerine data2 ise 3 adet index değerine sahiptir. Join işlemi uygularsak yukarıdaki şekilde incelediğimiz gibi sol taraftaki DataFrame’in yapısına uygun bir şekilde birleştirme işlemi yapılacaktır.

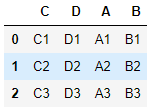
**df1.join(df2)**

****

Resim 56

Aynı işlemi tersten yapmış olsaydık, yani data2 isimli DataFrame’i sola almış olsaydık birleştirme işlemini yaparken data1’i data2 ye uyarlayacağı için en alttaki indis değerleri silinecekti

**df2.join(df1)**

****

Resim 57

**Merge Metodu ile Birleştirme İşlemi**

Merge metodu ile birleştirme işlemi SQL tablolardaki INNER JOIN işlemine benzemektedir. Her iki DataFrame’deki sütunlardan ortak olanına göre birleştirme işlemi yapılmaktadır. Elimizde ortak değişkenli iki adet DataFrame olduğunu varsayalım,

**data1=pd.DataFrame({"A":["A1","A2","A3","A4"],**

**"B":["B1","B2","B3","B4"],**

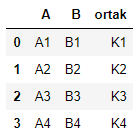
**"ortak":["K1","K2","K3","K4"]})**

**data2=pd.DataFrame({"C":["C1","C2","C3"],**

**"D":["D1","D2","D3"],**

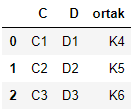
**"ortak":["K3","K4","K5"]})**

**data1**



Resim 58

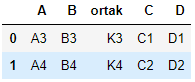
**data2**

****

Resim 59

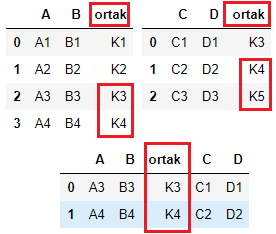
Burada ortak isimli sütuna göre birleştirme işlemi yapılacağına göre her iki DataFrame’de ortak isimli sütunda K4 değeri iki DataFrame’de de mevcut olduğundan

**pd.merge(data1,data2)**



Resim 60

Şeklinde bir birleştirme işlemi yapılacaktır. Burada birleştirme işlemini daha detaylı inceleyecek olursak,



Resim 61

Ortak olan sütunlardaki ortak olan değerlere göre birleştirme işlemi yapılmıştır. Aynı işlemi tam tersi olarak yapmak istersek, yine aynı değerlere ulaşacaktık, çünkü ortak olan değerler değişmeyecektir.

Join işlemi indexler üzerinden merge işlemi column’lar üzerinden birleştirme işlemi yapmaktadır.

**DataFrame Üzerinde Uygulanabilen Metodlar**

Veri analizinde sum , mean, median, min ve max fonkdiyonları ile özetleme yapabiliriz. Bu sayede veri hakkında genel bir kanıya ulaşılabilir.

count → Tüm elemanların sayısı

mean → Tüm elemanların ortalamalarını hesaplar

median → Tüm elemanların median’ı alır(ortadaki değer)

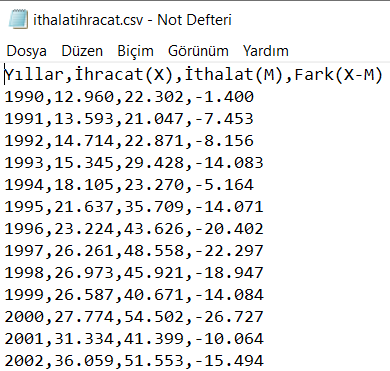
sum → Tüm elemanların toplamlarını alır

min, max → Tüm elemanların minimum ve maximum değerlerini getirir

std, var → Tüm elemanların standart sapmasını ve varyansını hesapları

prod → Tüm elamanların çarpımını hesaplar

Elimizde türliye’nin 1990 yılından bu yana ithalat ihracat rakamlarını içeren bir tablo olduğunu varsayalım

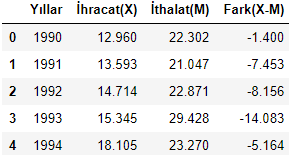


Resim 62

Dosyayı read\_csv metodu ile açarsak head metodu ile parametre girmeden ilk 5 veriye bakabiliriz. Head metodunun içerisine parametre vererek istediğimiz kadar değere bakabiliriz.

**data=pd.read\_csv("C:/Users/asus/Desktop/veriler/ithalatihracat.csv")**

**data.head()**



Resim 63

Shape metodu ile index ve column sayılarına bakabiliriz.

**data.shape**

**(28, 4)**

28 satır ve 4 sütundan oluştuğu görülmektedir.

Count metodu ile her bir sütundaki gözlem sayılarına ulaşabiliriz.

**data.count()**

**Yıllar 28**

**İhracat(X) 28**

**İthalat(M) 28**

**Fark(X-M) 28**

**dtype: int64**

sadece bir sütuna ait gözlemlerin sayısına bakmak istersek

**data["İhracat(X)"].count()**

**28**

Şeklinde bir metod kullanabiliriz.

Sum() fonksiyonu ile her sütun değeri için toplamları bulabiliriz.

**data.sum()**

**Yıllar 56098.000**

**İhracat(X) 2059.456**

**İthalat(M) 3213.457**

**Fark(X-M) -1146.040**

**dtype: float64**

mean() fonksiyonu ile ortalamalarını bulabiliriz.

**data.mean()**

**Yıllar 2003.500000**

**İhracat(X) 73.552000**

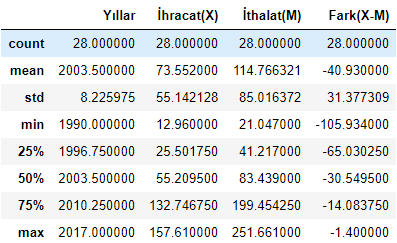
**İthalat(M) 114.766321**

**Fark(X-M) -40.930000**

**dtype: float64**

**describe** fonksiyonu ile her bir sütun için eleman sayıları **max min** ortalamaları %25-%75 lik değerlerine ulaşılabilir

**data.describe()**



Resim 64

**Group By İşlemleri**

Groupby işlemi aynı gözlem adına sahip verileri kendi arasında işlem yapmamızı sağlar. Örnek olarak elimizde futbol takımlarının isimleri ve maaşların olduğu bir DataFrame olduğunu varsayalım.

**data=pd.DataFrame({"takım":["galatasaray","fenerbahçe","fenerbahçe","galatasaray","beşiktaş"],**

**"maaş":[5000,3000,8000,4000,7000]})**

**data**



Resim 65

Group By işlemleri sql sorgularında kullanılan Group By işlemlerine benzer. Şimdi galatasaray’da oynayan oyuncuların maaşlarını takım isimlerine göre groupby işlemi yapmak istersek

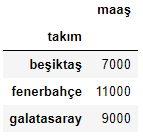
**grup=data.groupby("takım")**

**grup**

**<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000002BE99B37A88>**

Şimdi grup değişkeninin üzerinde metodları uygulamak istersek

**grup.sum()**



Resim 66

Grupların toplam değerleri bulundu.

Ortalamalarını bulmak istersek

**grup.mean()**



Resim 67

Yapılan işlemi daraltmak istersek

**data.groupby("takım").sum().loc["galatasaray"]**

**maaş 9000**

**Name: galatasaray, dtype: int64**

Şeklinde bir kullanım yapabiliriz.

Count metodu ile her bir gruptaki değerlerin sayısına ulaşılabilir.

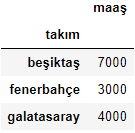
**grup.count()**



Resim 68

Her takımda alınan en düşük maaşlar

**data.groupby("takım").min()**



Resim 77

**data.groupby("takım").min()["maaş"]["galatasaray"]**

**4000**

Galatasaray takımında alınan en düşük maaşı bulmuş olduk.

Her takımda alınan maaşların ortalamasını bulmak için

**data.groupby("takım").mean()**



Resim 78

**Özel Fonksiyonlar**

DataFrame’lerde kullanabileceğimiz bazı özel fonksiyonlar bulunmaktadır.

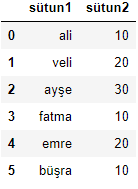
**df=pd.DataFrame({**

**"sütun1":[1,2,3,4,5,6],**

**"sütun2":[10,20,30,10,20,10],**

**"sütun3":["ali","veli","ayşe","fatma","emre","büşra"]})**

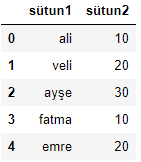
**df**



Resim 79

DataFrame’in ilk değerlerini görmek için head fonksiyonu kullanıyorduk

df.head()



Resim 80

DataFramedeki bazı değerlerin birden fazla kullanıldığı görülmektedir. Buradaki eşsiz değerleri görüntülemek için unique() fonksiyonu kullanılmalıdır.

**df["sütun2"].unique()**

**array([10, 20, 30], dtype=int64)**

Eşsiz değerlerin sayısını bulmak için nunique() fonksiyonu kullanılmalıdır.

**df["sütun2"].nunique()**

**3**

Her bir değerin kaç defa geçtiğini bulabilmek için value\_counts() methodu kullanılmalıdır.

**df["sütun2"].value\_counts()**

**10 3**

**20 2**

**30 1**

**Name: sütun2, dtype: int64**

**Pivot Tablolar**

Pivot Table(Özet Tablo) elimizdeki verilerin daha anlaşılır bir şekilde ve özet halinde sunulmasını sağlayan bir yöntemdir. MS-Excel ve MS-Acces programlarında sıklıkla kullanılır. Şimdi bir örnek üzerinden anlatmaya çalışalım.

**data=pd.DataFrame({"Aylar":["Haziran","Temmuz","Ağustos","Haziran","Temmuz","Ağustos","Haziran","Temmuz","Ağustos"],**

**"Sıcaklık":[30,34,35,28,32,34,29,35,36],**

**"Şehirler":["İstanbul","İstanbul","İstanbul","Ankara","Ankara","Ankara","Antalya","Antalya","Antalya"]})**

**data**

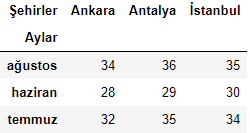
elimizde yukarıdaki gibi veri seti olduğunu varsayalım. Veri setinde 3 farklı şehre ait yaz mevsimi sıcaklık değerleri bulunmaktadır. Verileri görüntülediğimizde aşağıdaki gibi bir çıktı elde edebiliriz.



Resim 86

Data Frame’i pivot tablo ile tekrar düzenlemek istersek aşağıdaki kodları yazmamız gerekmektedir. pivot tablo hazırlarken ayları index değerler, şehirleri ise sütun değerleri olarak belirleyelim.

**data.pivot\_table(index="Aylar",columns="Şehirler",values="Sıcaklık")**



Resim 87

**Aggregate Fonksiyonu:**

Aggregate fonksiyonu ile python’da işlevsel bir şekilde toplulaştırma işlemleri yapabiliriz. Elimizde takımların

**import pandas as pd**

**import numpy as np**

**data=pd.DataFrame({"takımlar":["gs","fb","bjk","gs","fb","bjk"],**

**"attığıgoller":[188,192,176,194,190,189],**

**"yediğigoller":[70,76,68,82,70,60]})**

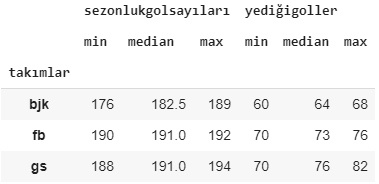
**Data**

****

**Resim 90**

Şimdi dataframedeki gözlemlere göre(takım isimleri) tablomuzu daha sade hale getirelim.

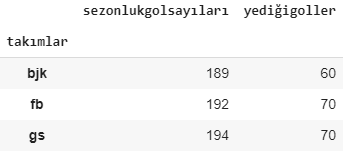
**data.groupby("takımlar").aggregate(["min",np.median,max])**



Resim 91

Hatta tabloyu biraz daha sadeleştirerek takımların attığı en fazla gol sayıları ile yediği en az gol sayılarını listelemek istersek

**data.groupby("takımlar").aggregate({"sezonlukgolsayıları":"max","yediğigoller":"min"})**



Resim 92

**Filter Fonksiyonu:**

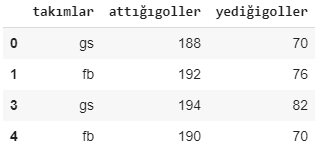
Filter fonksiyonu ile belirli kriterler belirleyerek bu kriterlere göre görüntüleme işlemleri yapabiliriz. Örnek olarak 190 dan fazla gol atan takımlara ulaşmak istediğimizde aşağıdaki gibi fonksiyon yazalım.

**def filtre(a):**

**return a["attığıgoller"].max()>190**

**şimdi fonksiyonumuzu çağıralım.**

**data.groupby("takımlar").filter(filtre)**



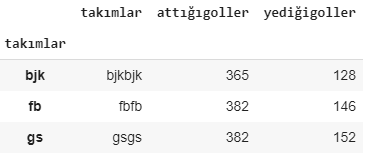
Resim 93

Burada dikkat ettiyseniz gs nin ilk değer 188 olduğu halde, farklı bir gözlemde 190 değerini geçtiği için her iki gözlem değeri de görüntülenmiştir.

**Apply Fonksiyonu:**

Apply fonksiyonu satır ve sütunlar üzerinde hesaplama işlemleri yapmamızı sağlar.

data.groupby("takımlar").apply(np.sum)



Resim 94

Görüldüğü üzere satırlar üzerinde toplama işlemi yapılmış oldu.

**Veri Okuma Kaynakları**

Önceki bölümde CSV uzantılı dosyaları okuma işlemi yapmıştık. Şimdi ise farklı dosyalar üzerinde(txt,xls) veri seti okuma işlemlerine değinelim.

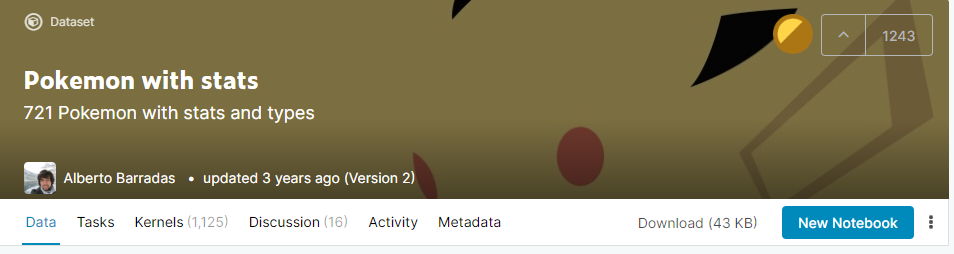
**Veri Analizi 1**

Şimdi öğrendiğimiz bilgiler doğrultusunda bir veri analizi işlemi gerçekleştirelim.

Elimizde <https://www.kaggle.com/> adresinden alınmış World Happiness Report isimli dataset’in 2019 verileri olsun dataset’i indirerek 2019 verisini açalım bunun için yukarıdaki adrese girerek datasets içerisinde pokemon araması yaptıralım ya da

<https://www.kaggle.com/abcsds/pokemon>

adresine giderek

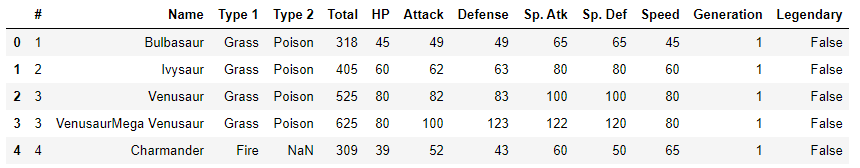


Resim 85

Dataset dosyasını indirelim. Daha sonra jupyter üzerinde dosyayı açalım.

**data=pd.read\_csv("C:/Users/asus/Desktop/veriler/pokemon.csv")**

**data** **.head()**



Resim 81

Burada pokemon’ların isimleri tipleri ile vuruş güçleri savunmaları gibi bilgiler bulunmaktadır.

Buradaki veri sayısını öğrenebilmek için,

**data.index**

**RangeIndex(start=0, stop=800, step=1)**

**Ya da**

**len(data)**

**800**

800 adet değer olduğu görülmektedir.

Yaşam güçlerinin(HP) ortalamasını bulabilmek için

**data["HP"].mean()**

**69.25875**

En yüksek yaşam gücünü bulmak için

**data["HP"].max()**

**255**

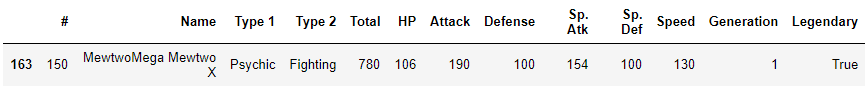
En yüksek attack gücüne sahip karakterin ismini bulalım, bunun için en yüksek attack gücüne sahip değeri bulalım,

**data["Attack"].max()**

**190**

Şimdi filtreleme işlemi yapalım

**data[data["Attack"]==190]**



Resim 82

Bu karakterin type1 ini almak istersek

**data[data["Attack"]==190]["Type 1"].iloc[0]**

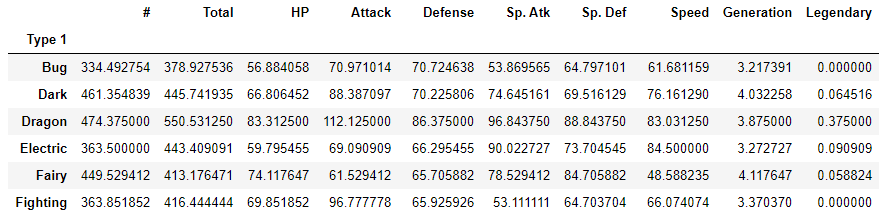
**'Psychic'**

Pikachu karakterinin Defence’ini bulalım

**data[data["Name"]=="Pikachu"]["Defense"].iloc[0]**

**40**

Type1’e göre gruplara ayırarak her bir grubun ortalama güçlerini bulalım.

data.groupby("Type 1").mean()

resim 83

toplam kaç adet tür(type1) olduğunu bulalım

**data["Type 1"].nunique()**

**18**

Her türden kaç adet karakter olduğunu bulmak için,

**data["Type 1"].value\_counts()**

**Water 112**

**Normal 98**

**Grass 70**

**Bug 69**

**Psychic 57**

**Fire 52**

**Electric 44**

**Rock 44**

**Ground 32**

**Dragon 32**

**Ghost 32**

**Dark 31**

**Poison 28**

**Fighting 27**

**Steel 27**

**Ice 24**

**Fairy 17**

**Flying 4**

**Name: Type 1, dtype: int64**