

## PANDAS

Pandas, veri dosyaları üzerinde analizler yapmamızı sağlar.

### Pandas Serileri

Etiketli verilerden oluşan, tek boyutlu veri yapılarıdır. Etiket değerine **indeks** denir. Liste, sıralı diziler ya da sözlükler ile seri oluşturabiliriz.

Seri oluşturmak için **pd.Series()** metodunu kullanırız.

```
In [ ]: import pandas as pd
#pd.Series(data, index, dtype, copy)
label_list = ['I', 'am', 'Learning', 'Data', 'Science']
data_List = [1,2,3,4,5]
pd_series = pd.Series(label_list,data_List)
print(pd_series)
```

```
1           I
2         am
3    Learning
4      Data
5   Science
dtype: object
```

Seriler arasında matematiksel işlemler yapabiliriz.

```
In [ ]: sinav1 = {'Michael': 35, 'Olivia': 85}
v1 = pd.Series(sinav1)
sinav2 = {'Michael': 44}
v2 = pd.Series(sinav2)
sinav3 = {"Luna": 99}
v3 = pd.Series(sinav3)
```

```
In [ ]: print(v1+v2)
```

```
Michael    79.0
Olivia     NaN
dtype: float64
```

**NaN** değerinin gelmesinin sebebi, Olivia'nın diğer seri içerisinde olmamasıdır. İşlem yapabilmemiz için verinin tüm serilerde bulunması gereklidir.

```
In [ ]: v4 = pd.concat([v3,v1]) #Birleştirdik
print(v4)
```

```
Luna    99
Michael  35
Olivia  85
dtype: int64
```

Seriler içerisinde bulunan değerlere, listelerde olduğu gibi direkt erişebiliriz.

```
In [ ]: print(v4["Luna"])
```

```
99
```

## DataFrame

Pandas'ın çok boyutlu veri yapılarıdır.

Sütunlar **Column / Feature**, satırlar ise **Row / Indeks** olarak adlandırılırlar.

Oluşturmak için **pd.DataFrame()** metodunu kullanınız.

```
In [ ]: from numpy.random import randn  
df = pd.DataFrame(data = randn(4,4),  
                   index = ['A','B','C','D'],  
                   columns = ['Columns1','Columns2','Columns3','Columns4'])  
df
```

```
Out[ ]:   Columns1  Columns2  Columns3  Columns4  
A    -1.234957  -0.266394   0.429624   0.699209  
B     0.480639  -0.084587  -0.906595   0.676379  
C     0.337801  -0.994191   0.224610  -0.746010  
D    -0.538235  -1.965010   2.090792   0.705179
```

```
In [ ]: df[['Columns1','Columns2']]
```

```
Out[ ]:   Columns1  Columns2  
A    -1.234957  -0.266394  
B     0.480639  -0.084587  
C     0.337801  -0.994191  
D    -0.538235  -1.965010
```

Yeni sütun eklemek istersek, direkt tanım yapabiliriz.

```
In [ ]: df["Columns5"] = pd.Series(randn(4), ['A','B','C','D'])  
df
```

```
Out[ ]:   Columns1  Columns2  Columns3  Columns4  Columns5  
A    -1.234957  -0.266394   0.429624   0.699209  -0.577381  
B     0.480639  -0.084587  -0.906595   0.676379  -0.579672  
C     0.337801  -0.994191   0.224610  -0.746010  -0.985170  
D    -0.538235  -1.965010   2.090792   0.705179   0.725850
```

Sütun ve satırlar arasında matematiksel işlemler yapabiliriz.

```
In [ ]: df["Columns6"] = df["Columns3"] / df["Columns1"]  
df
```

	Columns1	Columns3	Columns4	Columns5	Columns6
A	-1.234957	0.429624	0.699209	-0.577381	-0.347886
B	0.480639	-0.906595	0.676379	-0.579672	-1.886229
C	0.337801	0.224610	-0.746010	-0.985170	0.664919
D	-0.538235	2.090792	0.705179	0.725850	-3.884535

Ekli sütunları çıkartmak için default değeri **0** olan **axis** parametresini kullanırız.**0** satır, **1** ise sütunları simgeler.

DataFrame üzerinde yaptığımız işlemlerin kalıcı olmasını istiyorsak **inplace** parametresini **True** yapmalıyız.

```
In [ ]: df.drop('Columns2', axis = 1, inplace = True)
df
```

	Columns1	Columns3	Columns4	Columns5
A	-1.234957	0.429624	0.699209	-0.577381
B	0.480639	-0.906595	0.676379	-0.579672
C	0.337801	0.224610	-0.746010	-0.985170
D	-0.538235	2.090792	0.705179	0.725850

Seçtiğimiz bir sütunu **index** başlığı olarak atayabiliriz.

```
In [ ]: df.set_index("Columns6", inplace=True)
df
```

	Columns1	Columns3	Columns4	Columns5
Columns6				
<b>-0.347886</b>	-1.234957	0.429624	0.699209	-0.577381
<b>-1.886229</b>	0.480639	-0.906595	0.676379	-0.579672
<b>0.664919</b>	0.337801	0.224610	-0.746010	-0.985170
<b>-3.884535</b>	-0.538235	2.090792	0.705179	0.725850

Satır ve sütun isimlerine erişmek için **.names** metodunu kullanırız

```
In [ ]: df.index.names
```

```
Out[ ]: FrozenList(['Columns6'])
```

```
In [ ]: df.columns.names
```

```
Out[ ]: FrozenList([None])
```

**iloc** fonksiyonu satırın index değerine, **loc** ise indeks adına göre sütun değerlerini döndürür.

```
In [ ]: df = pd.DataFrame(data = randn(4,4),
                           index = ['A','B','C','D'],
                           columns = ['Columns1','Columns2','Columns3','Columns4'])
df
```

```
Out[ ]:   Columns1  Columns2  Columns3  Columns4
A -0.568221  0.430966 -1.460638 -1.229908
B  0.078514 -0.424473  0.238341 -2.051763
C -0.419836 -1.005222  2.304167  0.903924
D  0.142613  0.247134 -0.317689  1.780903
```

```
In [ ]: df.loc['A']
```

```
Out[ ]: Columns1    -0.568221
         Columns2     0.430966
         Columns3    -1.460638
         Columns4    -1.229908
Name: A, dtype: float64
```

```
In [ ]: df.iloc[0]
```

```
Out[ ]: Columns1    -0.568221
         Columns2     0.430966
         Columns3    -1.460638
         Columns4    -1.229908
Name: A, dtype: float64
```

```
In [ ]: df.loc['A','Columns2']
```

```
Out[ ]: 0.43096642964542553
```

```
In [ ]: df.loc[['A','D'],['Columns4','Columns2']]
```

```
Out[ ]:   Columns4  Columns2
A    -1.229908  0.430966
D    1.780903  0.247134
```

### **DataFrame Üzerinde Filtreleme İşlemleri**

```
In [ ]: df
```

```
Out[ ]:   Columns1  Columns2  Columns3  Columns4
A -0.568221  0.430966 -1.460638 -1.229908
B  0.078514 -0.424473  0.238341 -2.051763
C -0.419836 -1.005222  2.304167  0.903924
D  0.142613  0.247134 -0.317689  1.780903
```

Bütün verileri tek koşul üzerinden kontrol edebiliriz. Örneğin 1'den büyük olan değerler, **True** olurken, küçük olanlar ise **False** değerini almıştır.

```
In [ ]: df > 1
```

	Columns1	Columns2	Columns3	Columns4
A	False	False	False	False
B	False	False	False	False
C	False	False	True	False
D	False	False	False	True

Şartı sağlayan verileri görmek istersek,

```
In [ ]: df[df>0]
```

	Columns1	Columns2	Columns3	Columns4
A	NaN	0.430966	NaN	NaN
B	0.078514	NaN	0.238341	NaN
C	NaN	NaN	2.304167	0.903924
D	0.142613	0.247134	NaN	1.780903

Koşulu sağlayan değerleri yoksa, görmek istediğimiz veriler boş dönecektir.<br>  
Sadece bir ya da bir kaç sütunun koşulu sağlayıp, sağlanmadığını kontrol edelim.

```
In [ ]: df['Columns1'] > 0
```

```
Out[ ]: A    False
         B    True
         C    False
         D    True
Name: Columns1, dtype: bool
```

```
In [ ]: df[df['Columns4']> 0]
```

	Columns1	Columns2	Columns3	Columns4
C	-0.419836	-1.005222	2.304167	0.903924
D	0.142613	0.247134	-0.317689	1.780903

Koşul ve şart işaretlerini de kullanabiliriz.<br> **&** işaretini **and** anlamında ve | işaretini **or** anlamındadır.

```
In [ ]: df[(df['Columns1']> 0) & (df['Columns3']> 0)]
```

	Columns1	Columns2	Columns3	Columns4
B	0.078514	-0.424473	0.238341	-2.051763

## DataFrame'lerin Multi Index Olarak Tanımlanması

Index değerinin çok olduğu durumlarda, gruplama yapmak için kullanılır. İlk olarak iki adet indeks tanımlayacağımız, ardından **zip** fonksiyonu ile birleştiriceğiz. Bunu yaparken **Tuple** ve **Dict** veri türlerini de kullanabiliriz.

```
In [ ]: OuterIndex = ['Group1', 'Group1', 'Group1',
                   'Group2', 'Group2', 'Group2',
                   'Group3', 'Group3', 'Group3']
InnerIndex = ['Index1', 'Index2', 'Index3',
              'Index1', 'Index2', 'Index3',
              'Index1', 'Index2', 'Index3']
list(zip(OuterIndex, InnerIndex))
```

```
Out[ ]: [('Group1', 'Index1'),
          ('Group1', 'Index2'),
          ('Group1', 'Index3'),
          ('Group2', 'Index1'),
          ('Group2', 'Index2'),
          ('Group2', 'Index3'),
          ('Group3', 'Index1'),
          ('Group3', 'Index2'),
          ('Group3', 'Index3')]
```

Birleştirdiğimiz indeks tanımlarını, **pd.MultiIndex.from\_tuples** metodu yardımıyla Multi Indeks olarak tanımladık.

```
In [ ]: hierarchy = list(zip(OuterIndex, InnerIndex))
hierarchy = pd.MultiIndex.from_tuples(hierarchy)
hierarchy
```

```
Out[ ]: MultiIndex([('Group1', 'Index1'),
                     ('Group1', 'Index2'),
                     ('Group1', 'Index3'),
                     ('Group2', 'Index1'),
                     ('Group2', 'Index2'),
                     ('Group2', 'Index3'),
                     ('Group3', 'Index1'),
                     ('Group3', 'Index2'),
                     ('Group3', 'Index3')],
```

Şimdi rastgele değerler oluşturalım.

```
In [ ]: df = pd.DataFrame(randn(9,3),hierarchy,columns = ['Column1','Colum2','Column3'])
df
```

		Column1	Column2	Column3
Group	Index			
Group1	Index1	-0.376460	1.555650	-0.529534
	Index2	1.118490	-2.627686	-2.525702
	Index3	1.370861	0.408032	0.557412
Group2	Index1	0.091995	0.497143	0.871308
	Index2	0.009221	0.067685	1.876448
	Index3	-0.720830	-0.053261	0.077901
Group3	Index1	0.332184	-1.551962	0.060127
	Index2	0.504010	-1.512259	-0.446424
	Index3	-0.061763	-0.442169	0.819662

Ulaşmak istediğimiz verilere, grup, indeks, sütun bilgileri üzerinden erişebiliriz.

In [ ]: `df['Column1']`

Out[ ]: Group1 Index1 -0.376460  
 Index2 1.118490  
 Index3 1.370861  
 Group2 Index1 0.091995  
 Index2 0.009221  
 Index3 -0.720830  
 Group3 Index1 0.332184  
 Index2 0.504010  
 Index3 -0.061763  
 Name: Column1, dtype: float64

In [ ]: `df.loc[['Group1', 'Group2']]`

		Column1	Column2	Column3
Group	Index			
Group1	Index1	-0.376460	1.555650	-0.529534
	Index2	1.118490	-2.627686	-2.525702
	Index3	1.370861	0.408032	0.557412
Group2	Index1	0.091995	0.497143	0.871308
	Index2	0.009221	0.067685	1.876448
	Index3	-0.720830	-0.053261	0.077901

In [ ]: `df.loc['Group1'].loc['Index1']['Column1']`

Out[ ]: -0.37646045455297744

`.xs()` fonksiyonu `loc` ve `iloc` fonksiyonlarının işlevlerini yerine getirir. Fonksiyon, grup indeks sırası ile çalışır.

In [ ]: `df.xs('Group1') # df.xs('Group1') = df.loc['Group1']`

```
Out[ ]:      Column1    Column2    Column3
Index1 -0.376460  1.555650 -0.529534
Index2  1.118490 -2.627686 -2.525702
Index3  1.370861  0.408032  0.557412
```

İndeks bilgilerine isim atamak istersek `.index.names` fonksiyonu kullanılır.

```
In [ ]: df.index.names = ['Groups','Indexes']
df
```

```
Out[ ]:      Column1    Column2    Column3
Groups  Indexes
Group1  Index1 -0.376460  1.555650 -0.529534
Index2  1.118490 -2.627686 -2.525702
Index3  1.370861  0.408032  0.557412
Group2  Index1  0.091995  0.497143  0.871308
Index2  0.009221  0.067685  1.876448
Index3 -0.720830 -0.053261  0.077901
Group3  Index1  0.332184 -1.551962  0.060127
Index2  0.504010 -1.512259 -0.446424
Index3 -0.061763 -0.442169  0.819662
```

Seçeceğimiz indeksin grup bilgilerini getirelim.

```
In [ ]: df.xs('Index1', level = 'Indexes')
```

```
Out[ ]:      Column1    Column2    Column3
Groups
Group1 -0.376460  1.555650 -0.529534
Group2  0.091995  0.497143  0.871308
Group3  0.332184 -1.551962  0.060127
```

```
In [ ]: df.xs('Index1', level = 'Indexes')[['Column1']]
```

```
Out[ ]: Groups
Group1 -0.376460
Group2  0.091995
Group3  0.332184
Name: Column1, dtype: float64
```

## **Bozuk ve Kayıp Verilerle Uğraşmak**

Yeni bir veri oluşturup, DataFrame'e çevirelim.

```
In [ ]: import numpy as np  
arr = np.array([[10, 20, np.nan], [3, np.nan, np.nan], [13, np.nan, 4]])  
arr
```

```
Out[ ]: array([[10., 20., nan],  
               [ 3., nan, nan],  
               [13., nan, 4.]])
```

```
In [ ]: df = pd.DataFrame(arr, index = ['Index1', 'Index2', 'Index3'],  
                        columns = ['Column1', 'Column2', 'Column3'])  
df
```

```
Out[ ]:
```

	Column1	Column2	Column3
Index1	10.0	20.0	NaN
Index2	3.0	NaN	NaN
Index3	13.0	NaN	4.0

Bozuk veriler **NaN** olarak gözükmektedir. Amacımız bunlardan kurtulmak.

Bunu bazen ilgili yerleri silerek, daha fazla veri toplayarak ya da farklı hesaplamalar ile yerlerini doldurarak yaparız.

**.dropna()** metodu indeks satırında en az bir adet **NaN** değer varsa, o satırı siler.

**axis** değerini 1 yaparsak, aynı işlemi sütun için uygular. **inplace** değerini True yapmadığımız için değişiklikler yansımayacak.

```
In [ ]: df.dropna()
```

```
Out[ ]:
```

	Column1	Column2	Column3
--	---------	---------	---------

```
In [ ]: df.dropna(axis = 1)
```

```
Out[ ]:
```

	Column1
Index1	10.0
Index2	3.0
Index3	13.0

**thresh** belirleyeceğimiz adete kadar karşılık gelen düzgün veri varsa, silme işlemini yapma demektir.

```
In [ ]: df.dropna(thresh = 2)
```

```
Out[ ]:
```

	Column1	Column2	Column3
Index1	10.0	20.0	NaN
Index3	13.0	NaN	4.0

**fillna** belirleyeceğimiz değer, **NaN** olan değerler ile değiştirir.

```
In [ ]: df.fillna(value = 0)
```

```
Out[ ]:   Column1  Column2  Column3  
Index1      10.0      20.0      0.0  
Index2       3.0       0.0      0.0  
Index3      13.0       0.0      4.0
```

Sütunlarda bulunan verilerin toplamı

```
In [ ]: df.sum()
```

```
Out[ ]: Column1    26.0  
Column2    20.0  
Column3     4.0  
dtype: float64
```

Verilerin toplamı

```
In [ ]: df.sum().sum()
```

```
Out[ ]: 50.0
```

```
In [ ]: df.fillna(value = (df.sum().sum()) / 5)
```

```
Out[ ]:   Column1  Column2  Column3  
Index1      10.0      20.0     10.0  
Index2       3.0      10.0     10.0  
Index3      13.0      10.0      4.0
```

**.size** toplam veri adetini döndürür.

```
In [ ]: df.size
```

```
Out[ ]: 9
```

**.isnull()**, veri içerisindeki **NaN** değerleri True olarak gösterir.

```
In [ ]: df.isnull()
```

```
Out[ ]:   Column1  Column2  Column3  
Index1      False     False     True  
Index2      False      True     True  
Index3      False      True    False
```

**NaN** değerlerinin toplam adetini görmek istersek.

```
In [ ]: df.isnull().sum()
```

```
Out[ ]: Column1      0  
         Column2      2  
         Column3      2  
         dtype: int64
```

**Nan** olmayan veri adeti içinse,

```
In [ ]: df.size - df.isnull().sum()
```

```
Out[ ]: Column1      9  
         Column2      7  
         Column3      7  
         dtype: int64
```

### **GroupBy Operasyonları**

Verileri istediğimiz şekilde grüplamamızı sağlar.

```
In [ ]: data = {'Job': ['Data Mining', 'CEO', 'Lawyer', 'Lawyer', 'Data Mining', 'CEO'],  
              'Labouring': ['Immanuel', 'Jeff', 'Olivia', 'Maria', 'Walker', 'Obi-Wan'],  
              'Salary': [4500, 30000, 6000, 5250, 5000, 35000]}
```

```
In [ ]: df = pd.DataFrame(data)  
df
```

```
Out[ ]:
```

	Job	Labouring	Salary
0	Data Mining	Immanuel	4500
1	CEO	Jeff	30000
2	Lawyer	Olivia	6000
3	Lawyer	Maria	5250
4	Data Mining	Walker	5000
5	CEO	Obi-Wan	35000

**.groupby()** yardımıyla Salary türüne göre grüplama yaparak, toplam bilgisini alalım.

```
In [ ]: SalaryGroupBy = df.groupby('Salary')  
SalaryGroupBy
```

```
Out[ ]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x00000184750A7700>
```

```
In [ ]: SalaryGroupBy.sum()
```

```
Out[ ]: Job Labouring
```

### Salary

<b>4500</b>	Data Mining	Immanuel
<b>5000</b>	Data Mining	Walker
<b>5250</b>	Lawyer	Maria
<b>6000</b>	Lawyer	Olivia
<b>30000</b>	CEO	Jeff
<b>35000</b>	CEO	Obi-Wan

```
In [ ]: SalaryGroupBy.min()
```

```
Out[ ]: Job Labouring
```

### Salary

<b>4500</b>	Data Mining	Immanuel
<b>5000</b>	Data Mining	Walker
<b>5250</b>	Lawyer	Maria
<b>6000</b>	Lawyer	Olivia
<b>30000</b>	CEO	Jeff
<b>35000</b>	CEO	Obi-Wan

Job türüne göre gruplama yapalım.

```
In [ ]: df.groupby('Job').min()
```

```
Out[ ]: Labouring Salary
```

### Job

<b>CEO</b>	Jeff	30000
<b>Data Mining</b>	Immanuel	4500
<b>Lawyer</b>	Maria	5250

```
In [ ]: SalaryGroupBy.max()
```

```
Out[ ]:          Job  Labouring
```

**Salary**

<b>4500</b>	Data Mining	Immanuel
<b>5000</b>	Data Mining	Walker
<b>5250</b>	Lawyer	Maria
<b>6000</b>	Lawyer	Olivia
<b>30000</b>	CEO	Jeff
<b>35000</b>	CEO	Obi-Wan

```
In [ ]: df.groupby('Salary').sum()
```

```
Out[ ]:          Job  Labouring
```

**Salary**

<b>4500</b>	Data Mining	Immanuel
<b>5000</b>	Data Mining	Walker
<b>5250</b>	Lawyer	Maria
<b>6000</b>	Lawyer	Olivia
<b>30000</b>	CEO	Jeff
<b>35000</b>	CEO	Obi-Wan

```
In [ ]: df.groupby('Job').sum(numeric_only=False).loc['CEO']
```

```
Out[ ]: Labouring    Jeff
        Salary      65000
        Name: CEO, dtype: object
```

```
In [ ]: df.groupby('Job').count()
```

```
Out[ ]:          Labouring  Salary
```

Job		
<b>CEO</b>	2	2
<b>Data Mining</b>	2	2
<b>Lawyer</b>	2	2

```
In [ ]: df.groupby('Job').min()['Salary']
```

```
Out[ ]: Job
        CEO          30000
        Data Mining   4500
        Lawyer        5250
        Name: Salary, dtype: int64
```

```
In [ ]: df.groupby('Job').min()['Salary']['Lawyer']
```

```
Out[ ]: 5250
```

## **Concatenate, Merge Ve Join Fonksiyonları**

**Concatenate**, birleştirme işlemini bu fonksiyon ile yapmaktayız. Özellik olarak **zip** fonksiyonuna benzemektedir.

```
In [ ]: data = {'A': ['A1', 'A2', 'A3', 'A4'],
              'B': ['B1', 'B2', 'B3', 'B4'],
              'C': ['C1', 'C2', 'C3', 'C4']}
data1 = {'A': ['A5', 'A6', 'A7', 'A8'],
          'B': ['B5', 'B6', 'B7', 'B8'],
          'C': ['C5', 'C6', 'C7', 'C8']}
df1 = pd.DataFrame(data, index = [1,2,3,4])
df2 = pd.DataFrame(data1, index = [5,6,7,8])
df1
```

```
Out[ ]:   A   B   C
1  A1  B1  C1
2  A2  B2  C2
3  A3  B3  C3
4  A4  B4  C4
```

```
In [ ]: df2
Out[ ]:   A   B   C
5  A5  B5  C5
6  A6  B6  C6
7  A7  B7  C7
8  A8  B8  C8
```

**.concat** ile birleştirme işlemini yapalım.

```
In [ ]: pd.concat([df1,df2])
Out[ ]:   A   B   C
1  A1  B1  C1
2  A2  B2  C2
3  A3  B3  C3
4  A4  B4  C4
5  A5  B5  C5
6  A6  B6  C6
7  A7  B7  C7
8  A8  B8  C8
```

**axis** parametresini 1 yaparak, sütun birleştirmesi yapabiliriz.  
Birleştirmeden dolayı karşılığı olmayan değerler **NaN** gelecektir.

```
In [ ]: pd.concat([df1,df2], axis = 1)
```

```
Out[ ]:   A    B    C    A    B    C
1  A1  B1  C1  NaN  NaN  NaN
2  A2  B2  C2  NaN  NaN  NaN
3  A3  B3  C3  NaN  NaN  NaN
4  A4  B4  C4  NaN  NaN  NaN
5  NaN  NaN  NaN  A5  B5  C5
6  NaN  NaN  NaN  A6  B6  C6
7  NaN  NaN  NaN  A7  B7  C7
8  NaN  NaN  NaN  A8  B8  C8
```

**Join** yalnızca diğer veri kümelerinde eşleşmesi olan nesnelerin döndürüldüğü bir birleşim türü olan iç birleşim uygular.

```
In [ ]: data = {'A': ['A1', 'A2', 'A3', 'A4'],
              'B': ['B1', 'B2', 'B3', 'B4'],
              'C': ['C1', 'C2', 'C3', 'C4']}
data1 = {'A': ['A5', 'A6', 'A7', 'A8'],
          'B': ['B5', 'B6', 'B7', 'B8'],
          'C': ['C5', 'C6', 'C7', 'C8']}
df1 = pd.DataFrame(data, index = [1,2,3,4])
df2 = pd.DataFrame(data1, index = [5,6,7,8])
df1
```

```
Out[ ]:   A    B    C
1  A1  B1  C1
2  A2  B2  C2
3  A3  B3  C3
4  A4  B4  C4
```

```
In [ ]: data2 = { 'id': ['1', '2', '6', '7', '8'],
                  'Feature1': ['K', 'M', 'O', 'Q', 'S'],
                  'Feature2': ['L', 'N', 'P', 'R', 'T']}
df2 = pd.DataFrame(data2, columns = ['id', 'Feature1', 'Feature2'])
df2
```

```
Out[ ]:   id  Feature1  Feature2
```

<b>0</b>	1	K	L
<b>1</b>	2	M	N
<b>2</b>	6	O	P
<b>3</b>	7	Q	R
<b>4</b>	8	S	T

```
In [ ]: df1.join(df2)
```

```
Out[ ]:   A   B   C   id  Feature1  Feature2
```

<b>1</b>	A1	B1	C1	2	M	N
<b>2</b>	A2	B2	C2	6	O	P
<b>3</b>	A3	B3	C3	7	Q	R
<b>4</b>	A4	B4	C4	8	S	T

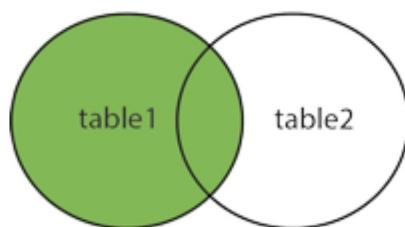
```
In [ ]: df2.join(df1)
```

```
Out[ ]:   id  Feature1  Feature2      A   B   C
```

<b>0</b>	1	K	L	NaN	NaN	NaN
<b>1</b>	2	M	N	A1	B1	C1
<b>2</b>	6	O	P	A2	B2	C2
<b>3</b>	7	Q	R	A3	B3	C3
<b>4</b>	8	S	T	A4	B4	C4

**how** parametresini değiştirerek, **Left Join**, **Right Join**, **Inner** ve **Outer** işlemlerini uygularız.

### LEFT JOIN

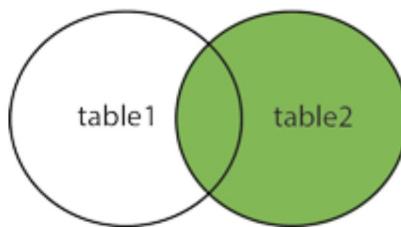


```
In [ ]: df1.join(df2, how = 'left')
```

```
Out[ ]:   A  B  C  id  Feature1  Feature2
```

1	A1	B1	C1	2	M	N
2	A2	B2	C2	6	O	P
3	A3	B3	C3	7	Q	R
4	A4	B4	C4	8	S	T

### RIGHT JOIN

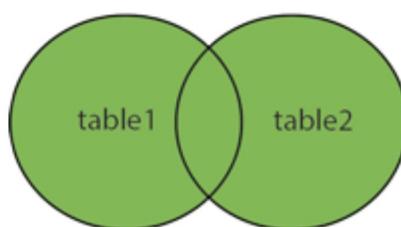


```
In [ ]: df1.join(df2, how = 'right')
```

```
Out[ ]:   A  B  C  id  Feature1  Feature2
```

0	NaN	NaN	NaN	1	K	L
1	A1	B1	C1	2	M	N
2	A2	B2	C2	6	O	P
3	A3	B3	C3	7	Q	R
4	A4	B4	C4	8	S	T

### FULL OUTER JOIN

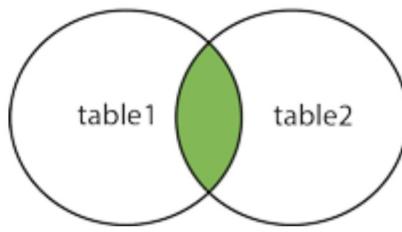


```
In [ ]: df1.join(df2, how = 'outer')
```

```
Out[ ]:   A  B  C  id  Feature1  Feature2
```

0	NaN	NaN	NaN	1	K	L
1	A1	B1	C1	2	M	N
2	A2	B2	C2	6	O	P
3	A3	B3	C3	7	Q	R
4	A4	B4	C4	8	S	T

## INNER JOIN



```
In [ ]: df1.join(df2, how = 'inner')
```

```
Out[ ]:   A   B   C   id  Feature1  Feature2
```

	A	B	C	id	Feature1	Feature2
1	A1	B1	C1	2	M	N
2	A2	B2	C2	6	O	P
3	A3	B3	C3	7	Q	R
4	A4	B4	C4	8	S	T

```
In [ ]: df1.join(df2, sort = 'True')
```

```
Out[ ]:   A   B   C   id  Feature1  Feature2
```

	A	B	C	id	Feature1	Feature2
1	A1	B1	C1	2	M	N
2	A2	B2	C2	6	O	P
3	A3	B3	C3	7	Q	R
4	A4	B4	C4	8	S	T

Hepsini birleştirelim.

```
In [ ]: frames = [df1,df2]
df_keys = pd.concat(frames, keys=['x', 'y'])
df_keys
```

```
Out[ ]:   A   B   C   id  Feature1  Feature2
```

	A	B	C	id	Feature1	Feature2
x 1	A1	B1	C1	NaN	NaN	NaN
2	A2	B2	C2	NaN	NaN	NaN
3	A3	B3	C3	NaN	NaN	NaN
4	A4	B4	C4	NaN	NaN	NaN
y 0	NaN	NaN	NaN	1	K	L
1	NaN	NaN	NaN	2	M	N
2	NaN	NaN	NaN	6	O	P
3	NaN	NaN	NaN	7	Q	R
4	NaN	NaN	NaN	8	S	T

## DataFrame Operasyonları

```
In [ ]: data = {'Column1': [1,2,3,4,5,6],  
             'Column2': [1000,1000,2000,3000,3000,1000],  
             'Column3': ['Mace Windu','Darth Vader',  
                         'Palpatine','Kylo Ren','Rey','Obi-Wan']}  
df = pd.DataFrame(data)  
df
```

```
Out[ ]:   Column1  Column2    Column3  
0         1      1000  Mace Windu  
1         2      1000  Darth Vader  
2         3      2000  Palpatine  
3         4      3000  Kylo Ren  
4         5      3000       Rey  
5         6      1000  Obi-Wan
```

**.head()**, ilk beş değeri gösterir. Parametre alırsa o kadarını listeler.

```
In [ ]: df.head()
```

```
Out[ ]:   Column1  Column2    Column3  
0         1      1000  Mace Windu  
1         2      1000  Darth Vader  
2         3      2000  Palpatine  
3         4      3000  Kylo Ren  
4         5      3000       Rey
```

```
In [ ]: df.head(n = 2)
```

```
Out[ ]:   Column1  Column2    Column3  
0         1      1000  Mace Windu  
1         2      1000  Darth Vader
```

**.info** veri seti hakkında bilgi verir.

```
In [ ]: df.info
```

```
Out[ ]: <bound method DataFrame.info of      Column1  Column2    Column3  
0         1      1000  Mace Windu  
1         2      1000  Darth Vader  
2         3      2000  Palpatine  
3         4      3000  Kylo Ren  
4         5      3000       Rey  
5         6      1000  Obi-Wan>
```

**.describe()** istatistik bilgiler verir. **.corr()** korelasyon bilgiside bunlardan biridir.

```
In [ ]: df.describe()
```

```
Out[ ]:      Column1    Column2
```

	Column1	Column2
<b>count</b>	6.000000	6.000000
<b>mean</b>	3.500000	1833.333333
<b>std</b>	1.870829	983.192080
<b>min</b>	1.000000	1000.000000
<b>25%</b>	2.250000	1000.000000
<b>50%</b>	3.500000	1500.000000
<b>75%</b>	4.750000	2750.000000
<b>max</b>	6.000000	3000.000000

```
In [ ]: df
```

```
Out[ ]:      Column1    Column2    Column3
```

	Column1	Column2	Column3
<b>0</b>	1	1000	Mace Windu
<b>1</b>	2	1000	Darth Vader
<b>2</b>	3	2000	Palpatine
<b>3</b>	4	3000	Kylo Ren
<b>4</b>	5	3000	Rey
<b>5</b>	6	1000	Obi-Wan

**.unique()** birbirinden farklı kategorilerin bilgisini verir.

```
In [ ]: df['Column1'].unique()
```

```
Out[ ]: array([1, 2, 3, 4, 5, 6], dtype=int64)
```

**.nunique()**, kaç adet eşsiz, tekrar etmeyen verinin olduğu bilgisini döndürür.

```
In [ ]: df['Column1'].nunique()
```

```
Out[ ]: 6
```

**.value\_counts()** hangi değerden kaç adet olduğunu veren fonksiyondur.

```
In [ ]: df['Column2'].value_counts()
```

```
Out[ ]: 1000    3
       3000    2
       2000    1
Name: Column2, dtype: int64
```

Tanımlayacağımız işlemleri uygulayabiliriz.

```
In [ ]: df['Column2'].apply(lambda x : x **2 )
```

```
Out[ ]: 0    1000000
         1    1000000
         2    4000000
         3    9000000
         4    9000000
         5    1000000
Name: Column2, dtype: int64
```

`.sort_values()` belli bir değere göre sıralamamızı sağlar. `ascending` parametresi `False` olursa büyükten küçüğe sıralama yapar.

```
In [ ]: df.sort_values(by=['Column1', 'Column2'])
```

```
Out[ ]:   Column1  Column2      Column3
0          1     1000  Mace Windu
1          2     1000  Darth Vader
2          3     2000  Palpatine
3          4     3000  Kylo Ren
4          5     3000      Rey
5          6     1000  Obi-Wan
```

```
In [ ]: df.sort_values('Column2', ascending = False)
```

```
Out[ ]:   Column1  Column2      Column3
3          4     3000  Kylo Ren
4          5     3000      Rey
2          3     2000  Palpatine
0          1     1000  Mace Windu
1          2     1000  Darth Vader
5          6     1000  Obi-Wan
```

### Pivot Table

```
In [ ]: df = pd.DataFrame({'Month': ['January', 'February',
                                      'March', 'January', 'February',
                                      'March', 'January', 'February',
                                      'March'],
                           'State': ['New York', 'New York',
                                     'New York', 'Texas', 'Texas',
                                     'Texas', 'Washington', 'Washington',
                                     'Washington'],
                           'moisture': [20, 25, 65, 34, 56, 85, 21, 56, 79]})
```

```
Out[ ]:   Month      State  moisture
```

0	January	New York	20
1	February	New York	25
2	March	New York	65
3	January	Texas	34
4	February	Texas	56
5	March	Texas	85
6	January	Washington	21
7	February	Washington	56
8	March	Washington	79

```
In [ ]: df.corr(numeric_only=True)
```

```
Out[ ]:    Column1  Column2
```

<b>Column1</b>	1.000000	0.380562
<b>Column2</b>	0.380562	1.000000

```
In [ ]: df.pivot_table(index = 'Month', columns = 'State', values = 'moisture')
```

```
Out[ ]:    State  New York  Texas  Washington
```

Month	New York	Texas	Washington
February	25	56	56
January	20	34	21
March	65	85	79

### **Veri Setini Okuma Yöntemleri**

**pd.read\_csv()** .csv uzantılı dosyayı okur.

```
In [ ]: dataset = pd.read_csv('Dosya_Yolu\Data_Name.csv')
```

**.to\_csv()** Datayı tekrardan csv'ye dönüştürebilmek için

```
In [ ]: dataset.to_csv('Data_Name')
```

**pd.read\_excel()** Excel Dosyasını okumak için

```
In [ ]: excelset = pd.read_excel('Excels_Name.xlsx')
```

**.to\_excel()** Excel dosyasına çevirme

```
In [ ]: excelset.to_excel('excelnewfile.xlsx')
```

**pd.read\_html()** internette ki bir veriyi okumak için

```
In [ ]: new = pd.read_html(' Datasetin urlsi.html')
```

## İletişim



## Referans

- [Pandas Tutorial](#)
- [w3schools](#)