

Pozulmuş və natamam verilənlərlə əməliyyatlar

Bəzən əldə etdiyimiz verilənlər natamam və düzgün olmaya bilər. Bəzən verilən data itkiyə uğrayır və bizi məqsədimizdən yayındıra bilər. Bu vəziyyətlərdə nə etməli olduğumuza nəzər yetirməyə çalışaq. Ancaq əvvəlcə natamam bir data yadaq.

```
arr = np.array([[10,20,np.nan],[3,np.nan,np.nan],[13,np.nan,4]])
arr
```

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

Göründüyü kimi, 'np.nan' kodu verilənlərdə 'Not a number' yaradır.

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

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

Və biz bu verilənlərlə DataFrame əmələ gətirdik.

Mən demirəm ki, buradan görəcəyiniz addımları mütləq istifadə edin. Amma buradakı prosedurun addımları araşdırdığınız(nəzərdən keçirdiyiniz), istədiyiniz və ya axtarılan datadan asılı olaraq dəyişə bilər. Bəzən dataların natamam olduğu yerlərdə ortalama, 0, 1 və standart kənarlaşma kimi parametrlər yerləşdirilə bilər. Bəzən isə boş məlumatlar birbaşa silinir. Nə etmək istədiyiniz tamamilə sizin seçiminizə bağlıdır.

```
df.dropna()
```

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

Məsələn, yuxarıdakı kod hər bir indeks üzrə sətirdə ən azı bir 'NaN' varsa, həmin sətri silər.

```
df.dropna(axis = 1)
```

	Column1
Index1	10.0
Index2	3.0
Index3	13.0

'Sütun'a uyğun olaraq 'Axis' parametrisini təyin etdikdə, o, bizə heç bir 'NaN' dəyəri olmadan 'Column1'i qaytarır və digər sütunları silir. Eyni zamanda, 'inplace' parametrisinin dəyərini 'True' olaraq təyin etməsəz, bu əməliyyatın nəticəsi, daimi olaraq tətbiq edilməyəcəkdir.

```
df.dropna(thresh = 2)
```

	Column1	Column2	Column3
Index1	10.0	20.0	NaN
Index3	13.0	NaN	4.0

Sizcə, burada 'thresh' parametri nəyi ifadə edir? Belə deyək, bu, " verilən ox üzrə ən azı iki tam məlumat (yəni, pozulmamış və ya null olmayan) varsa, silmə" deməkdir. Təbii ki, bu rəqəmi siz gördüyünüz işə görə təyin edirsiniz.

```
df.fillna(value = 0)
```

	Column1	Column2	Column3
Index1	10.0	20.0	0.0
Index2	3.0	0.0	0.0
Index3	13.0	0.0	4.0

Bu kod boş verilənlərə **"value"** parametrisinin aldığı dəyəri təyin edir. Nümunədə **"value"** parametrisinə '0' verilmiş və bu göstərici 'NaN' olan dəyərlərə təyin edilmişdir. Biz burada **'String'** dəyərlər də işlədə bildiyimiz üçün *fillna()* metodunun olduqca funksional və yararlı olduğunu ifadə edə bilərik.

Fərz edək ki, verilənlərin strukturu uyğundur və axtardığımız verilənlər ortalamadan asılı deyildir və biz ortalama dəyəri 'NaN' qiymətlərinə təyin edirik.

```
df.sum()
```

```
Column1    26.0
Column2    20.0
Column3     4.0
dtype: float64
```

```
df.sum().sum()
```

```
50.0
```

Siz burada, ilk kodun gördüyü kimi hər bir "Sütun" dəyərinin cəmini verdiyini, Növbəti kodun isə bu ədədlərin hamısını cəmlədiyini görə bilərsiniz.

```
df.fillna(value = (df.sum().sum())/ 5)
```

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

Bu məntiqi yanaşmanı 'value' -yə təyin edib 5-ə bölsək (ortalamanı almaq üçün), o, 'NaN' qiymətlərinin yerinə '10' təyin edəcək.

Eyni məntiqi yanaşma ilə standart sapma və variasiya kimi dəyərləri də edə bilərik. Amma bunu əvvəlki yazımızda izah etdik və kodu orada asanlıqla tətbiq edə bilərsiniz.

Bəzən daha sadə sorğular da olduqca funksional - yararlı ola bilər:

```
df.size
```

9

Burada nə qədər data olduğunu bizə göstərir.

```
df.isnull()
```

	Column1	Column2	Column3
Index1	False	False	True
Index2	False	True	True
Index3	False	True	False

'True' dəyərləri burada 'NaN' dəyərlərin olduğu mövqeyi ifadə edir

```
df.isnull().sum()
```

```
Column1    0
Column2     2
Column3     2
dtype: int64
```

Bu kod sütunlarda neçə ədəd "NaN" dəyər olduğunu ifadə edir.

```
df.isnull().sum().sum()
```

4

Bu kod ümumi verilənlərdə neçə ədəd "NaN" dəyər olduğunu ifadə edir.

Datada toplam göstəricilərin sayını və eləcə də neçə 'NaN' dəyərin olduğunu bilsək, hər bir 'Column' üzrə dolu xanaların sayını da tapa bilərik:

```
df.size - df.isnull().sum()
```

```
Column1     9
Column2     7
Column3     7
dtype: int64
```

Yuxarıdakı kodun nəticəsinə nəzər yetirsək 'Column' üzrə dolu xanaların sayını bu kodun nəticəsində görə bilərik.

GroupBy Əməliyyatları:

GroupBy əməliyyatları Sql cədvəllərində olduğu kimidir. SQL-i bilməsəniz belə rahat olun. Çətin mövzu deyil, amma vacib mövzudur.

GroupBy əməliyyatları istədiyiniz əməllərin nəticəsini qruplar formasında göstərir.

```
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]}

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

Datamızı yaratdıqdan sonra, biz onu DataFrame -ə daxil edək:

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

	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

```
SalaryGroupBy = df.groupby('Salary')
SalaryGroupBy
```

<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f8d45511790>

Biz burada 'Salary' xüsusiyyətini əsas alaraq qruplaşdırma apardıq.

SalaryGroupBy.sum()

	Job	Labouring	
Salary			
4500	Data Mining	Immanuel	
5000	Data Mining	Walker	
5250	Lawyer	Maria	
6000	Lawyer	Olivia	
30000	CEO	Jeff	
35000	CEO	Obi-Wan	

SalaryGroupBy.min()

	Job	Labouring	
Salary			
4500	Data Mining	Immanuel	
5000	Data Mining	Walker	
5250	Lawyer	Maria	
6000	Lawyer	Olivia	
30000	CEO	Jeff	
35000	CEO	Obi-Wan	

SalaryGroupBy.max()

	Job	Labouring	
Salary			
4500	Data Mining	Immanuel	
5000	Data Mining	Walker	
5250	Lawyer	Maria	
6000	Lawyer	Olivia	
30000	CEO	Jeff	
35000	CEO	Obi-Wan	

Gördüyümüz kimi hansı aqreقات funksiyaları işlədsək, nəticələr cədvəl olaraq çıxışa verilir.

df.groupby('Salary').sum()

	Job	Labouring	
Salary			
4500	Data Mining	Immanuel	
5000	Data Mining	Walker	
5250	Lawyer	Maria	
6000	Lawyer	Olivia	
30000	CEO	Jeff	
35000	CEO	Obi-Wan	

Yuxarıdakı aqreقات funksiyalardan istifadə etdikdə bu şəkildə də yazılışdan istifadə edərək kodunuzu qısalda bilərsiniz.

df.groupby('Job').sum().loc['CEO'] # CEO -nün toplam maaşı

```
Salary      65000
Name: CEO, dtype: int64
```

df.groupby('Job').count()

	Labouring	Salary	
Job			
CEO	2	2	
Data Mining	2	2	
Lawyer	2	2	

df.groupby('Job').min()

	Labouring	Salary	
Job			
CEO	Jeff	30000	
Data Mining	Immanuel	4500	
Lawyer	Maria	5250	

Numpy yazımızda paylaşdığımız funksiyalar burada da işlədilə bilər.

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

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

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

```
5250
```

Birincide “Salary” üzrə qruplaşdırdıq və ən aşağı maaşları çap etdik. Növbətidə ən az maaş alan vəkilin maaşını çap etdik.

```
df.groupby('Job').mean()['Salary']['CEO']
```

```
32500.0
```

Burada biz CEO-ların peşələrinə görə sıralanmış orta əmək haqqını tapırıq. GroupBy Əməliyyatları göründüyü kimi çətin bir mövzu deyil. Bu olduqca qısa və sadə mövzudur.

Concatenate, Merge və Join Funksiyaları:

Gəlin, ilk olaraq Concatenate funksiyası ilə başlayaq. Adətən birləşdirmə əməlini bu funksiya ilə həyata keçiririk. O eyni bir list içində çalışan ‘zip’ funksiyası kimi davranar.

```
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
```

	A	B	C
1	A1	B1	C1
2	A2	B2	C2
3	A3	B3	C3
4	A4	B4	C4

```
df2
```

	A	B	C
5	A5	B5	C5
6	A6	B6	C6
7	A7	B7	C7
8	A8	B8	C8

İki dataset yaradaq və onu aşağıdakı kimi kombinasiya edək:

```
pd.concat([df1,df2])
```

	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

Həmçinin belə də birləşdirə bilərdik:

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

	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

Burada olmayan məlumatlar təbii ki, 'NaN' olaraq təyin olunur. Bu əməliyyat yalnız 2 dataset üçün deyil, daha çox dataset olduğu hallarda da istifadə edilə bilər.

```
data1 = {
    'id': ['1', '2', '3', '4', '5'],
    'Feature1': ['A', 'C', 'E', 'G', 'I'],
    'Feature2': ['B', 'D', 'F', 'H', 'J']}

df1 = pd.DataFrame(data1, columns = ['id', 'Feature1', 'Feature2'])
df1
```

	id	Feature1	Feature2
0	1	A	B
1	2	C	D
2	3	E	F
3	4	G	H
4	5	I	J

```
data2 = {
    'id': ['1', '2', '6', '7', '8'],
    'Feature1': ['K', 'M', 'O', 'Q', 'S'],
    'Feature2': ['L', 'N', 'P', 'R', 'T']}
```

Join ilə davam edək. *'Join'* funksiyaları, belə desək, birləşmə əməliyyatı kimi düşünülə bilər. Bu funksiyaları anlamaq üçün orta məktəbdə öyrəndiyimiz *"çoxluqlar"* mövzusunı xatırlaya bilərik. Beləliklə, iki dataset yaradaq və bu əməliyyatları icra etməyə başlayaq.

```
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
```

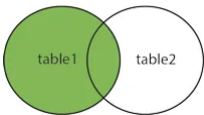
	A	B	C
1	A1	B1	C1
2	A2	B2	C2
3	A3	B3	C3
4	A4	B4	C4

```
df2 = pd.DataFrame(data2, columns = ['id', 'Feature1', 'Feature2'])
df2
```

	id	Feature1	Feature2
0	1	K	L
1	2	M	N
2	6	O	P
3	7	Q	R
4	8	S	T

'Left Join' ilə başlayaq.

LEFT JOIN



Left Join Sql-dəki 'Left Join' əməli ilə tamamilə eynidir. Onun funksiyası yuxarıdakı şəkildə verilmişdir. Biz burada "df1" -i table1, "df2" -ni table2 olaraq qəbul edək.

```
df1.join(df2)
```

	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

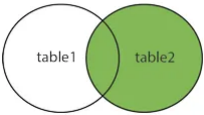
Tam tərsini edərsək sizcə nə baş verər?

```
df2.join(df1)
```

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

Right Join, ilə də eyni mühakiməni aparmaq olar. Funksiyası və kodu aşağıda verilmişdir.

RIGHT JOIN

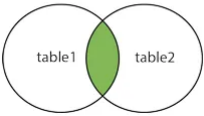


```
df1.join(df2, how = 'right')
```

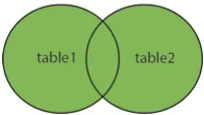
	A	B	C	id	Feature1	Feature2
0	NaN	NaN	NaN	1	K	L
1	A1	B1	C1	2	M	N
2	A2	B2	C2	6	O	P
3	A3	B3	C3	7	Q	R
4	A4	B4	C4	8	S	T

Növbəti olaraq 'inner' və 'outer' join funksiyalarına baxaq:

INNER JOIN



FULL OUTER JOIN



'Full Outer Join' yazıldığına fikir verməyin. Bu ad onun SQL-dəki adlandırılmasıdır. Burada biz, onu sadəcə 'outer' olaraq ifadə edirik və aşağıda hər ikisinə dair nümunələr verilmişdir.

```
df1.join(df2, how = 'outer')
```

	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

```
df1.join(df2, how = 'inner')
```

	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


```
df1.join(df2, sort = 'True')
```

	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

```
df1.join(df2, sort = 'False')
```

	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

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

		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	

Kod çalışıldıqdan sonra cədvəldən başqa yazıların ekrana gəlməsi bir xəbərdarlıq mesajı olub, xəta deyildir. Önemli bir şey deyildir. Bəzən, bu xəbərdarlıq meajları gələcək versiyada istifadə olunmalı olacaq kod forması haqqında məlumat verir.

Aşağıda 'Join' -in bütün parametrləri verilmişdir və onlar öyrəndiklərimizdən fərqli olaraq daha az istifadə olunur. İstəsəniz daha dərin araşdırıa bilərsiniz.

other : DataFrame, Series, or list of DataFrame

Index should be similar to one of the columns in this one. If a Series is passed, its name attribute must be set, and that will be used as the column name in the resulting joined DataFrame.

on : str, list of str, or array-like, optional

Column or index level name(s) in the caller to join on the index in other, otherwise joins index-on-index. If multiple values given, the other DataFrame must have a MultiIndex. Can pass an array as the join key if it is not already contained in the calling DataFrame. Like an Excel VLOOKUP operation.

how : {'left', 'right', 'outer', 'inner'}, default 'left'

How to handle the operation of the two objects.

- left: use calling frame's index (or column if on is specified)
- right: use other's index.
- outer: form union of calling frame's index (or column if on is specified) with other's index, and sort it. lexicographically.
- inner: form intersection of calling frame's index (or column if on is specified) with other's index, preserving the order of the calling's one.

lsuffix : str, default ''

Suffix to use from left frame's overlapping columns.

rsuffix : str, default ''

Suffix to use from right frame's overlapping columns.

sort : bool, default False

Order result DataFrame lexicographically by the join key. If False, the order of the join key depends on the join type (how keyword).

Merge əməliyyatı join əməliyyatına kifayət qədər bənzəsə də, bəzi fərqli xüsusiyyətlərə malikdir. Aşağıda daha ətraflı izah edəcəyik.

right : DataFrame or named Series

Object to merge with.

how : {'left', 'right', 'outer', 'inner', 'cross'}, default 'inner'

Type of merge to be performed.

- left: use only keys from left frame, similar to a SQL left outer join; preserve key order.
- right: use only keys from right frame, similar to a SQL right outer join; preserve key order.
- outer: use union of keys from both frames, similar to a SQL full outer join; sort keys lexicographically.
- inner: use intersection of keys from both frames, similar to a SQL inner join; preserve the order of the left keys.
- cross: creates the cartesian product from both frames, preserves the order of the left keys.

on : label or list

Column or index level names to join on. These must be found in both DataFrames. If on is None and not merging on indexes then this defaults to the intersection of the columns in both DataFrames.

left_on : label or list, or array-like

Column or index level names to join on in the left DataFrame. Can also be an array or list of arrays of the length of the left DataFrame. These arrays are treated as if they are columns.

right_on : label or list, or array-like

Column or index level names to join on in the right DataFrame. Can also be an array or list of arrays of the length of the right DataFrame. These arrays are treated as if they are columns.

left_index : bool, default False Use the index from the left DataFrame as the join key(s). If it is a MultiIndex, the number of keys in the other DataFrame (either the index or a number of columns) must match the number of levels.

right_index : bool, default False

Use the index from the right DataFrame as the join key. Same caveats as left_index. sort : bool, default False Sort the join keys lexicographically in the result DataFrame. If False, the order of the join keys depends on the join type (how keyword). suffixes : list-like, default is ("_x", "_y") A length-2 sequence where each element is optionally a string indicating the suffix to add to overlapping column names in left and right respectively. Pass a value of None instead of a string to indicate that the column name from left or right should be left as-is, with no suffix. At least one of the values must not be None.

copy : bool, default True

If False, avoid copy if possible.

indicator : bool or str, default False

If True, adds a column to the output DataFrame called "_merge" with information on the source of each row. The column can be given a different name by providing a string argument. The column will have a Categorical type with the value of "left_only" for observations whose merge key only appears in the left DataFrame, "right_only" for observations whose merge key only appears in the right DataFrame, and "both" if the observation's merge key is found in both DataFrames.

validate : str, optional

If specified, checks if merge is of specified type.

- "one_to_one" or "1:1": check if merge keys are unique in both left and right datasets.
- "one_to_many" or "1:m": check if merge keys are unique in left dataset.
- "many_to_one" or "m:1": check if merge keys are unique in right dataset.
- "many_to_many" or "m:m": allowed, but does not result in checks.

Merge prosesinin bütün parametrlərini yuxarıda görə bilərsiniz. Amma, biz onların çoxundan istifadə etməyəcəyik.

```
dataset1 = {'A': ['A1', 'A2', 'A3'], 'B': ['B1', 'B2', 'B3'], 'Key': ['K1', 'K2', 'K3']}
dataset2 = {'X': ['X1', 'X2', 'X3', 'X4'], 'Y': ['Y1', 'Y2', 'Y3', 'Y4'], 'Key': ['K1', 'K2', 'K3', 'K4']}
df1 = pd.DataFrame(dataset1, index = [1,2,3])
df2 = pd.DataFrame(dataset2, index = [1,2,3,4])
df1
```

	A	B	Key	
1	A1	B1	K1	
2	A2	B2	K2	
3	A3	B3	K3	

df2

	X	Y	Key	
1	X1	Y1	K1	
2	X2	Y2	K2	
3	X3	Y3	K3	
4	X4	Y4	K4	

Dataseti də yaratdığımız üçün artıq başlaya bilərik.

```
pd.merge(df1,df2, on = 'Key')
```

	A	B	Key	X	Y	
0	A1	B1	K1	X1	Y1	
1	A2	B2	K2	X2	Y2	
2	A3	B3	K3	X3	Y3	

Və biz artıq ilk parametrimizlə tanış olduq. Açarsözü "On" parametrinə təyin etməli - yazmalısınız. Açarsözün datasetinizdə mühüm yeri olmalıdır ki, o, əsl "açar" ola bilsin.

```
pd.merge(df1,df2, on = 'Key', how = 'left')
```

	A	B	Key	X	Y	
0	A1	B1	K1	X1	Y1	
1	A2	B2	K2	X2	Y2	
2	A3	B3	K3	X3	Y3	

'How' parametrlərini artıq Join mövzusunun bilirik. Orada gördüyünüz 'inner', 'outer', 'right' kimi parametrlərə təyin edilən dəyərlər burada da keçərlidir.

```
pd.merge(df1,df2, on = 'Key', how = 'right')
```

	A	B	Key	X	Y	
0	A1	B1	K1	X1	Y1	
1	A2	B2	K2	X2	Y2	
2	A3	B3	K3	X3	Y3	
3	NaN	NaN	K4	X4	Y4	


```
pd.merge(df1,df2, on = 'Key', how = 'outer')
```

	A	B	Key	X	Y
0	A1	B1	K1	X1	Y1
1	A2	B2	K2	X2	Y2
2	A3	B3	K3	X3	Y3
3	NaN	NaN	K4	X4	Y4

```
pd.merge(df1,df2, on = 'Key', how = 'inner')
```

	A	B	Key	X	Y
0	A1	B1	K1	X1	Y1
1	A2	B2	K2	X2	Y2
2	A3	B3	K3	X3	Y3

```
pd.merge(df1,df2, on = 'Key',how = 'right',)
```

	A	B	Key	X	Y
0	A1	B1	K1	X1	Y1
1	A2	B2	K2	X2	Y2
2	A3	B3	K3	X3	Y3
3	NaN	NaN	K4	X4	Y4

```
pd.merge(df1,df2, how = 'right', left_index=True, right_index=True)
```

	A	B	Key_x	X	Y	Key_y
1	A1	B1	K1	X1	Y1	K1
2	A2	B2	K2	X2	Y2	K2
3	A3	B3	K3	X3	Y3	K3
4	NaN	NaN	NaN	X4	Y4	K4

```
pd.merge(df1,df2, left_index=True, right_index=True, how='outer')
```

	A	B	Key_x	X	Y	Key_y
1	A1	B1	K1	X1	Y1	K1
2	A2	B2	K2	X2	Y2	K2
3	A3	B3	K3	X3	Y3	K3
4	NaN	NaN	NaN	X4	Y4	K4

```
pd.merge(df1,df2, left_index=True, right_index=True, how='inner')
```

	A	B	Key_x	X	Y	Key_y
1	A1	B1	K1	X1	Y1	K1
2	A2	B2	K2	X2	Y2	K2
3	A3	B3	K3	X3	Y3	K3

'left_index' və 'right_index' parametrləri yuxarıdakı kodlara baxdıqda başa düşüləcək qədər sadədir. Digər parametrlər demək olar ki, istifadə edilmir, amma istəsəniz Pandas -ın öz [saytı](#) -nda araşdırma edə bilərsiniz.

Xüsusi Təşəkkürlər: [Batuhan Bayraktar](#)

Yazar: [Nuhbalayev Ramazan](#)

Kod Mənbəsi:

[A'dan Z'ye Pandas Tutoriali \(Başlangıç ve Orta Seviye\)](#) və [Kaggle: batuhan35](#)