

★ **Jump-start your best year yet:** Become a member and get 25% off the first year



A'dan Z'ye Pandas Tutorialı (Başlangıç ve Orta Seviye)



Batuhan Bayraktar · Follow

Published in Deep Learning Türkiye

22 min read · Oct 20, 2019

Share

More



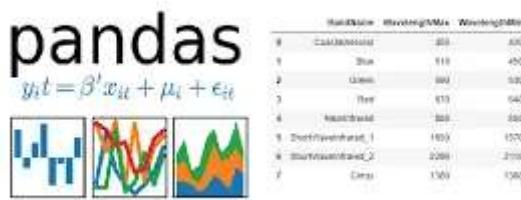
Pandas, Python programlama dili için yüksek performanslı, kullanımı kolay veri yapıları ve veri analiz araçları sağlayan açık kaynaklı bir kütüphanedir. '.csv' ve '.txt' dosyalarını açmak ve içerisinde bulunan verileri okuyarak istenen sonuca kolayca ulaşmak için kullanılmaktadır. Yani Pandas sayesinde bir excel dosyasını açarak içerisinde bulunan bir sütunu veya satırı seçip işlem yapabiliriz. Numpy kütüphanesinde yapılan verilerin şekillendirilmesi işlemi daha detaylı bir biçimde kullanılabilmektedir.

Pandas ile akılınıza gelebilecek pek çok işlemi başarıyla ve birkaç satır kodla yapabilirsiniz. Örneğin, İlk karadelgin görüntüsünün verisi pandas ile düzenlenmiştir. Ayrıca Pandas hız konusunda da optimize edilmiştir ve çok hızlı bir şekilde işlem yapabilmektedir. Başlamadan önce Numpy hakkındaki [şu yazıyı](#) okumanızı şiddetle tavsiye ederim. Çünkü bu iki konu birbirinden ayrılması imkansız olan konulardır ve bu ders için temel hazırlamasına yardımcı olacaktır.

Öncelikle ‘Seriler’ konusunu işleyeceğiz. Ardından’ DataFrame oluşturma ve işlevleri’ne değineceğiz. Çeşitli ‘Filtreleme işlemleri’nden sonra ‘DataFramelerin multi index olarak tanımlanması’ konusunu görüp, ‘Bozuk veya Hatalı veri durumunda neler yapabiliriz?’ sorusuna yanıt arayacağız. ‘GroupBy Sorguları’ ile belli bir değere göre sınıflamayı öğreneceğiz ve son olarak ‘Concatenate merge ve join fonksiyonları’ konusuna deðindikten sonra yazımızı sonlandıracagız. Aşağıdaki kodla Pandas kütüphanemizi ‘import’ edelim ve Pandas’a ilk adımıımızı atalım.

```
import pandas as pd
```

Ve şimdi ‘Pandas Serileri’ne geçmeye hazırız.



Pandas Serileri

Seriler Numpy dizileri baz alınarak oluşturuldukları için onlara çok benzerler. Seri, etiketli verilerden oluşan tek boyutlu bir veri yapısıdır. Etiket değerlerine ise indeks denir. Verinin kendisi sayılar, dizeler veya başka Python objelerinden oluşabilir. Serileri oluşturmak için ise listeler, sıralı dizeler ya da sözlükler kullanılabilir.

Pandas Serileri aşağıdaki değişkenleri alırlar.

```
pandas.Series( data, index, dtype, copy)
```

The parameters of the constructor are as follows –

Sr.No	Parameter & Description
1	data data takes various forms like ndarray, list, constants
2	index Index values must be unique and hashable, same length as data. Default np.arange(n) if no index is passed.
3	dtype dtype is for data type. If None, data type will be inferred
4	copy Copy data. Default False

Burada da ‘Bir seri nasıl oluşturulur?’ sorusunun cevabını görüyorsunuz.

Input:

```
Label_list = ['I', 'am', 'Learning', 'Data', 'Science']
Data_List = [1,2,3,4,5]
Pd_Series1 = pd.Series(Data_List, Label_list)
```

Ama illaki bu yolla ya da bu uzunlukta olmak zorunda değildir, Şu şekilde de olabilir.

Input:

```
data = np.array(['a', 'b', 'c', 'd'])
Series = pd.Series(data, [100, 101, 102, 103])
```

Output:

```
100    a
101    b
102    c
103    d
```

dtype: object

Şimdi 3 tane Seri oluşturalım ve bu seriler üzerinde bir takım işlemler uygulayalım.

```

Input:
DataDict = {'Michael_s exam result': 35, 'Olivia_s exam result': 85}
A = pd.Series(DataDict)
Output:
Michael_s exam result    35
Olivia_s exam result    85
dtype: int64
Input:
DataDict2 = {'Michael_s exam result': 44}
B = pd.Series(DataDict2)
Output:
Michael_s exam result    44
dtype: int64
Input:
DataDict3 = {'Darth_Vader_s exam result' :99}
C = pd.Series(DataDict3)
Output:
Darth_Vader_s exam result    99
dtype: int64

```

Görmüş olduğunuz gibi 3 adet seri oluşturduk. Sıra kullanabileceğimiz bazı durumlarda.

```

Input:
A + B
Output:
Michael_s exam result    79.0
Olivia_s exam result      NaN
dtype: float64

```

Bu durumlardan ilki: iki seri arasında dört işlem ve daha fazlası yapılabilir. Ancak dikkat etmeniz gereken bir husus var: ‘A’ değerinin içinde ‘Michael’ ve ‘Olivia’ kişilerinin sınav sonuçları varken, ‘B’ değerinde sadece ‘Michael’ kişisinin sınav sonucu var. İki değeri işleme soktuğumuzda sadece eşleşen değerler işleme tabii olur. Eşleşmeyen değer olursa (Bu örnekte ‘Olivia’ kişi), o değer ‘NaN’ olarak görünecektir.

‘Peki NaN nedir?’ diye soruyorsanız, ‘Not a number’ın kısaltılmış halidir ve bu veri tipinin sayı olmadığını ifade eder.

```

Input:
DataDict4 = C.append(A)
Output:

```

```
Darth_Vader_s exam result    99
Michael_s exam result      35
Olivia_s exam result       85
dtype: int64
```

Yukarıda görüldüğü gibi ‘DataDict4’ diye bir değer oluşturup, ‘C’ değerine ‘A’ değerini ekledik ve üç kişinin de sınav sonuçlarını görebildik. Gerçi ‘Darth Vader’ bu kadar yüksek not almışsa muhtemelen ‘Güçün karanlık tarafı’ ile ilgili bir sınavdır. :)

```
Input:
A['Michael_s exam result']
Output:
35
Input:
C['Darth_Vader_s exam result']
Output:
99
```

Yukarıda bulunan kodu hemen açıklayayım. ‘A’ değerinde ‘Michael’ ve ‘Olivia’ kişilerinin sınav sonuçları vardı. Diyelim ki biz ‘Michael’ kişisinin sınav sonucunu öğrenmek istiyoruz. Bu durumda ilk kodu yazmamız gereklidir. Aynısı ‘Darth Vader’ karakteri için de geçerlidir. Bu illaki sayısal bir değer olmak zorunda değildir (int veya float). ‘String’ yani kelimelerden oluşan bir veri de olabilir.

‘Series’ konusu aslında bu kadar kısa ve öz şimdî ‘DataFrame’ konusuna geçelim.

Pandas DataFrame

DataFrameler, Türkçesiyle ‘Veri Çerçeveleri’ Pandas kütüphanesinde asıl olayın döndüğü kısımdır ve pek çok işlemi bu kısımda yapacağız. Burada sütunlar ‘Column’ ya da ‘Feature’ olarak satırlar ise ‘row’ ya da ‘indeks’ olarak adlandırılır.

Öncelikle kendimize bir veri oluşturmalıyız.

```
from numpy.random import randn

Input:
df = pd.DataFrame(data = randn(5,5), index = ['A','B','C','D','E'],
columns = ['Columns1','Columns2','Columns3','Columns4','Columns5'])
Output:
```

	Columns1	Columns2	Columns3	Columns4	Columns5
A	-0.405682	-0.896915	0.220989	1.130962	0.626333
B	0.083131	-0.619950	-1.450517	-0.971936	-0.233941
C	0.809041	1.564836	1.157813	-1.063345	-0.260387
D	1.103567	-1.323272	1.372780	1.035036	1.334846
E	0.026578	-1.397457	1.735202	1.722989	-0.732212

Eğer 'randn' fonksiyonunu ilk defa görüporsanız 'Numpy' hakkında olan yazıyı okuyabilirsiniz.

Çalıştığınız veride size gerekli olan sütunları şu yöntemle alabilirsiniz.

```
Input:  
df['Columns1']  
Output:  
A      -0.405682  
B      0.083131  
C      0.809041  
D      1.103567  
E      0.026578  
Name: Columns1, dtype: float64
```

Üstelik sadece 1 sütun değil aynı yöntemle birden fazla sütunu da alabiliyorsunuz.

```
Input:  
df[['Columns1','Columns5']]  
Output:
```

	Columns1	Columns5
A	-0.405682	0.626333
B	0.083131	-0.233941
C	0.809041	-0.260387
D	1.103567	1.334846
E	0.026578	-0.732212

Yeni bir sütun ekleme işlemini ise şöyle yapıyoruz.

```
Input:  
df['Columns6'] = pd.Series(randn(5),['A','B','C','D','E'])  
Output:
```

	Columns1	Columns2	Columns3	Columns4	Columns5	Columns6
A	-0.405682	-0.896915	0.220989	1.130962	0.626333	0.034836
B	0.083131	-0.619950	-1.450517	-0.971936	-0.233941	1.377102
C	0.809041	1.564836	1.157813	-1.063345	-0.260387	-0.616719
D	1.103567	-1.323272	1.372760	1.035036	1.334846	0.221872
E	0.026578	-1.397457	1.735202	1.722989	-0.732212	-0.963177

Bununla da sınırlı değil, istediğiniz sütuna ya da satıra istediğiniz işlemi de uygulayabilirsiniz.

Input:

```
df['Columns7'] = (df['Columns6'] + df['Columns4'] - df['Columns1']) / df['Columns2'] * df['Columns3']
```

Output:

	Columns1	Columns2	Columns3	Columns4	Columns5	Columns6	Columns7
A	0.868092	-0.207914	-0.155522	-0.475097	-0.825116	-0.228605	-1.175716
B	1.506686	1.147269	-0.141114	-0.017588	-0.575107	-0.815875	0.287838
C	0.236537	0.441746	-1.595101	3.393466	2.040532	0.446495	-13.011599
D	0.705563	-0.325302	-1.087014	2.178489	-1.148871	-1.080769	1.310412
E	0.505845	-0.700655	0.466766	-1.686732	1.416919	0.064606	1.417624

Örneğin, yukarıda ‘Column7’ değerini oluşturup onu çeşitli sütunların işlemler sonucu oluşturduğu bir değere eşitledik ve böyle bir tablo oluşturduk.

Peki ‘Column’ ekleriz de çıkaramaz mıyız ?

Input:

```
df.drop('Columns2', axis = 1, inplace = True)
```

Output:

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7
A	0.868092	-0.155522	-0.475097	-0.825116	-0.228605	-1.175716
B	1.506686	-0.141114	-0.017588	-0.575107	-0.815875	0.287838
C	0.236537	-1.595101	3.393466	2.040532	0.446495	-13.011599
D	0.705563	-1.087014	2.178489	-1.148871	-1.080769	1.310412
E	0.505845	0.466766	-1.686732	1.416919	0.064606	1.417624

‘Axis’ in ne olduğunu merak edenler için kendisinin ‘default’ değeri 0'dır ve ‘0’ satırları ‘1’ sütunlarını simgeler. Buradaki ‘inplace’ parametresi önemlidir, bu

işlemin kalıcı olup olmadığını belirler ve ‘True’ dediğimiz zaman kalıcı olarak atama yapar.

Bir ‘Column’ değerini index başlığı olarak şöyle atayabiliyoruz.

```
df.set_index('Column ismi', inplace = True)
```

Aşağıdaki şekilde de ‘index’ ve ‘Column’ların isimlerini öğrenebiliyoruz.

Input:

```
df.index.names
```

Output:

```
FrozenList([None])
```

Input:

```
df.column.names
```

Output:

```
FrozenList([None])
```

Şimdi ise ‘loc’ ve ‘iloc’ durumunu işleyelim.

Input:

```
df.loc['C']
```

Output:

Columns1	0.236537
Columns3	-1.595101
Columns4	3.393466
Columns5	2.040532
Columns7	-13.011599
Name: C, dtype:	float64

Bu kodu kullandığımızda, bize ‘C’ satırında ki ‘Column’ değerlerini vermektedir.

Input:

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

Output:

Columns1	0.868092
Columns3	-0.155522
Columns4	-0.475097
Columns5	-0.825116
Columns7	-1.175716
Name: A, dtype:	float64

Input:

```
df.iloc[0]
```

Output:

Columns1	0.868092
Columns3	-0.155522
Columns4	-0.475097
Columns5	-0.825116
Columns7	-1.175716
Name: A, dtype: float64	

Bu ikisi arasındaki bağlantıyı oturup bir 10 saniye düşünmenizi istiyorum. Sunu farkedeceksiniz ki ikisi de oldukça benzer şeyler.

'iloc' fonksiyonu, satırın indeksine göre 'Column' değerlerini çıkartır ve satırların indeksleri bildiğiniz gibi 0'dan başlayarak ilerliyor. 'A'nın indeksi 0, ve 'iloc' fonksiyonuna '0' verilmiş. Bu durumda bu iki fonksiyon aslında aynı şeyi temsil ediyor. 'loc' fonksiyonunda isim belirtmeniz gerekirkken, 'iloc' da indeks belirtmeniz gerekiyor.

Bazen daha ileri gitmek gereklidir, mesela aşağıdaki gibi.

Input:

```
df.loc['A','Columns5']
```

Output:

```
-0.8251161336118147
```

İstenilen veri özel bir veri ya da nokta atışı yapılması gereken bir veri ise bu kullanılır.

[Open in app ↗](#)



Search



A

	Columns1	Columns3	Columns4	Columns5	Columns7
B	1.506686	-0.141114	-0.017588	-0.575107	0.287838
Columns4	NaN	NaN	NaN	NaN	NaN

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:1: FutureWarning:  
Passing list-like to .loc or [] with any missing label will raise  
KeyError in the future, you can use .reindex() as an alternative.  
  
See the documentation here:  
https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#deprecate-loc-reindex-listlike  
    """Entry point for launching an IPython kernel.
```

Eğer bu işlemleri ‘Kaggle’ ortamında yapıyorsanız yukarıdaki gibi bir şey çıkarsa endişelenmeyin. Bu birbhata değildir kodunuz sorunsuz çalışacaktır.

Anlaşıldığı üzere yukarıdaki ‘or , ‘and’ gibi ifadelerde kullanılabilmektedir.

Input:

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

Output:

	Columns4	Columns2
A	-0.475097	NaN
D	2.178489	NaN

Yukarıda da görüldüğü gibi birden fazla sütun ve satırı çağrııp bakabiliyoruz. Buradaki konumuz bitti, sırada ‘DataFrame Filtreleme İşlemleri’ var.

DataFrame Filtreleme İşlemleri

Adından da anlaşıldığı gibi, elimizde bulunan verinin belli aralıklardaki değeri bulmak ya da almak istediğimizde kullanmaktayız. Dilerseniz Verimizi hatırlayalım ve basit işlemlerle başlayalım.

Input:

```
df
```

Output:

	Columns1	Columns3	Columns4	Columns5	Columns7
A	-0.433733	2.177240	1.570476	1.506423	44.084240
B	1.428484	-0.436005	2.484519	0.781167	-0.410048
C	0.958244	-0.783307	-0.854804	1.397072	2.161001
D	-0.790159	-0.452584	-0.127712	0.815334	-3.347594
E	-0.553580	-0.629420	0.825197	0.051979	-0.228027

Ve basit işlemlere geçelim.

Input:

`df > 0.2`

Output:

	Columns1	Columns3	Columns4	Columns5	Columns7
A	False	True	True	True	True
B	True	False	True	True	False
C	True	False	False	True	True
D	False	False	False	True	False
E	False	False	True	False	False

Input:

`booleanDF = df > 0`

Output:

	Columns1	Columns3	Columns4	Columns5	Columns7
A	False	True	True	True	True
B	True	False	True	True	False
C	True	False	False	True	True
D	False	False	False	True	False
E	False	False	True	True	False

‘Boolean’ı şöyle açıklayabiliriz. İstenilen değer (burada ‘0’dan büyük olma şartı ile), koşulu sağlıyorsa ‘True’ değilse ‘False’ değerini döndürür.

Input:

`df[booleanDF]`

Output:

	Columns1	Columns3	Columns4	Columns5	Columns7
A	NaN	2.17724	1.570476	1.506423	44.084240
B	1.428484	NaN	2.484519	0.781167	NaN
C	0.958244	NaN	NaN	1.397072	2.161001
D	NaN	NaN	NaN	0.815334	NaN
E	NaN	NaN	0.825197	0.051979	NaN

Burada ise ‘NaN’ değerleri üstteki Boolean değerindeki ‘False’ değerlerini temsil etmekte.

Input:
`df[df > 0.5]`
Output:

	Columns1	Columns3	Columns4	Columns5	Columns7
A	NaN	2.17724	1.570476	1.506423	44.084240
B	1.428484	NaN	2.484519	0.781167	NaN
C	0.958244	NaN	NaN	1.397072	2.161001
D	NaN	NaN	NaN	0.815334	NaN
E	NaN	NaN	0.825197	NaN	NaN

Burada ise ‘df’ içinde ‘0.5’ten büyük olanları gösterir. ‘False’ olanlar yine ‘NaN’ olarak gösteriliyor. Bunu daha ileride daha iyi örneklerle açıklayacağız.

Input:
`df['Columns1'] > 0`
Output:

```
A    False
B    True
C    True
D    False
E    False
Name: Columns1, dtype: bool
```

Yukarıda ise belli bir ‘Column’ değerinin filtrelemesini görüyorsunuz. Aşağıda ise bunu her ‘Column’ değerine uygulanışını görüyorsunuz. Bazlarında sonuç farklı. Bazlarında 4 satır varken bazılarında 2 satır var, sizce neden olabilir ?

Input:
`df[df['Columns3'] > 0]`
Output:

	Columns1	Columns3	Columns4	Columns5	Columns7
A	-0.433733	2.17724	1.570476	1.506423	44.08424

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

Output:

	Columns1	Columns3	Columns4	Columns5	Columns7
A	-0.433733	2.177240	1.570476	1.506423	44.084240
B	1.428484	-0.436005	2.484519	0.781167	-0.410048
E	-0.553580	-0.629420	0.825197	0.051979	-0.228027

Input:

```
df[df['Columns5'] > 0]
```

Output:

	Columns1	Columns3	Columns4	Columns5	Columns7
A	-0.433733	2.177240	1.570476	1.506423	44.084240
B	1.428484	-0.436005	2.484519	0.781167	-0.410048
C	0.958244	-0.783307	-0.854804	1.397072	2.161001
D	-0.790159	-0.452584	-0.127712	0.815334	-3.347594
E	-0.553580	-0.629420	0.825197	0.051979	-0.228027

Input:

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

Output:

	Columns1	Columns3	Columns4	Columns5	Columns7
B	1.428484	-0.436005	2.484519	0.781167	-0.410048
C	0.958244	-0.783307	-0.854804	1.397072	2.161001

Kısaca özetlemek gerekirse, ‘True’ olan değerler gösterilir. Örneğin, ‘Column4’te 4 satır bulunmakta demek ki 5. satır ‘True’ değerinde.

Input:

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

Output:

	Columns1	Columns3	Columns4	Columns5	Columns7

Input:

```
df[(df['Columns1'] > 0) | (df['Columns5'] > 1)]
```

Output:

	Columns1	Columns3	Columns4	Columns5	Columns7
A	-0.433733	2.177240	1.570476	1.506423	44.084240
B	1.428484	-0.436005	2.484519	0.781167	-0.410048
C	0.958244	-0.783307	-0.854804	1.397072	2.161001

Yukarıda çeşitli filtreleme alıştırmalarını görüp sunuyoruz. Önceden bahsettiğimiz kurallara tabiidirler. '&' işaretini 'and' anlamında ve '|'' işaretini 'or' anlamındadır. Bu işaretler yerine 'and' ya da 'or' kullanır iseniz hata alırsınız. '&' işaretini 'SHIFT' ve '6' tuşları ile '|'' işaretini ise 'ALT GR' ve '-' ile yapılımaktadır.

Input:

```
df
```

Output:

	Columns1	Columns3	Columns4	Columns5	Columns7
A	-0.433733	2.177240	1.570476	1.506423	44.084240
B	1.428484	-0.436005	2.484519	0.781167	-0.410048
C	0.958244	-0.783307	-0.854804	1.397072	2.161001
D	-0.790159	-0.452584	-0.127712	0.815334	-3.347594
E	-0.553580	-0.629420	0.825197	0.051979	-0.228027

Input:

```
df['Columns6'] =
```

```
['newValue1', 'newValue2', 'newValue3', 'newValue4', 'newValue5']
```

Output:

	Columns1	Columns3	Columns4	Columns5	Columns2	Columns6
A	0.008692	1.183470	-0.182408	-0.656253	-0.049831	NewValue1
B	-0.604517	0.382120	-2.271736	1.506358	0.335323	NewValue2
C	-0.201642	0.892372	0.611166	-2.113499	-0.724640	NewValue3
D	0.578574	-0.779907	-0.496607	-0.573673	0.835551	NewValue4
E	-0.037768	-0.780190	-0.950621	0.987784	-0.602855	NewValue5

Verimizi güncellerken 2. fotoğrafta anlaşıldığı üzere verilen değer 'String' olsa bile direk atandığıdır.

Input:

```
df.set_index('Columns6' , inplace = True)
```

Output:

	Columns1	Columns3	Columns4	Columns5	Columns2
Columns6					
NewValue1	0.008692	1.163470	-0.182408	-0.656253	-0.049831
NewValue2	-0.604517	0.362120	-2.271736	1.506358	0.335323
NewValue3	-0.201642	0.892372	0.611166	-2.113499	-0.724640
NewValue4	0.578574	-0.779907	-0.496607	-0.573673	0.835551
NewValue5	-0.037768	-0.780190	-0.950621	0.967784	-0.602855

Input:

```
df.index.names
```

Output:

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

Input:

```
df.columns.names
```

Output:

```
FrozenList([None])
```

Yukarıda ilk satırda kalıcı bir şekilde index ismini 'Column6' olarak atadık. 2. satırda index başlığını sorguladık. Son kodda ise aynı mantıkla sütun ismini sorguluyor ve biz atamadığımız için 'None' çıkıyor.

DataFramelerin Multi Index Olarak Tanımlanması

Multi index, adından da anlaşılacağı gibi index sayısının çok olduğu durumlarda kullanılır ve temel olarak gruplanır.

Input:

```
OuterIndex =
['Group1', 'Group1', 'Group1', 'Group2', 'Group2', 'Group2', 'Group3', 'Gro
up3', 'Group3']
InnerIndex =
['Index1', 'Index2', 'Index3', 'Index1', 'Index2', 'Index3', 'Index1', 'Ind
ex2', 'Index3']
list(zip(OuterIndex, InnerIndex))
```

Output:

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

Input:

```
hierarchy = list(zip(OuterIndex, InnerIndex))
hierarchy = pd.MultiIndex.from_tuples(hierarchy)
```

Output:

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

Input:

```
df = pd.DataFrame(randn(9,3),hierarchy,columns =
['Column1','Column2','Column3'])
```

Output:

		Column1	Column2	Column3
Group1	Index1	0.104384	-0.862022	-0.793392
	Index2	0.170685	-0.094821	-0.902629
	Index3	-1.295203	-0.013739	0.412205
Group2	Index1	0.126634	0.835351	1.024142
	Index2	-0.056693	-2.421660	-1.474845
	Index3	-0.202806	-1.070895	-0.854948
Group3	Index1	-1.130723	-1.302862	1.184730
	Index2	-0.221858	0.102638	-1.467781
	Index3	-1.532283	0.848873	0.592044

1.görselde 2 adet index oluşturup ‘zip’ fonksiyonu ile birleştirdik. Bunu yaparken liste kullandık ama siz ‘Tuple’ ve ‘Dict’ de kullanabilirsiniz. 2. görselde bunu ‘hierarchy’e eşitledik. Sonra ise ‘pd.MultiIndex.from_tuples()’ özelliği ile Multi index’i oluşturduk. En son görselde ise ‘rand()’ fonksiyonu ile rastgele değerler atayıp sütun bilgilerini verdik. Ve görüldüğü üzere ‘Group’ ‘Index’ ‘Column’ değerlerine sahip Multi index DataFrame elde ettik.

Input:

```
df['Column1']
```

Output:

Group1	Index1	0.104384
	Index2	0.170685
	Index3	-1.295203
Group2	Index1	0.126634
	Index2	-0.056693
	Index3	-0.202806
Group3	Index1	-1.130723
	Index2	-0.221858
	Index3	-1.532283
Name: Column1, dtype: float64		

Input:

```
df.loc['Group1']
```

Output:

	Column1	Colum2	Column3
Index1	0.104384	-0.862022	-0.793392
Index2	0.170685	-0.094821	-0.902629
Index3	-1.295203	-0.013739	0.412205

Input:

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

Output:

		Column1	Column2	Column3
	Index1	0.104384	-0.862022	-0.793392
Group1	Index2	0.170685	-0.094821	-0.902629
	Index3	-1.295203	-0.013739	0.412205
	Index1	0.126634	0.835351	1.024142
Group2	Index2	-0.056693	-2.421660	-1.474845
	Index3	-0.202806	-1.070895	-0.854948

Yukarıda gördüğünüz gibi belli bir kısmı bu şekilde çağrılabilirsiniz.

Input:

```
df.loc['Group1']
```

Output:

	Column1	Column2	Column3	
	Index1	0.104384	-0.862022	-0.793392
	Index2	0.170685	-0.094821	-0.902629
	Index3	-1.295203	-0.013739	0.412205

Input:

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

Output:

	Column1	Column2	Column3	
	Index1	0.104384	-0.862022	-0.793392
Group1	Index2	0.170685	-0.094821	-0.902629
	Index3	-1.295203	-0.013739	0.412205
	Index1	0.126634	0.835351	1.024142
Group2	Index2	-0.056693	-2.421660	-1.474845
	Index3	-0.202806	-1.070895	-0.854948

Input:

```
df.loc['Group1'].loc['Index1']
```

Output:

```
Column1    0.104384
Colum2    -0.862022
Column3    -0.793392
Name: Index1, dtype: float64
```

‘Group’, ‘Column’ ve ‘Index’ gibi değerleri bu şekilde çağrılabılır ve birden fazla ve farklı türden de çağrıma yapabilirsiniz.

Input:

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

Output:

```
0.10438364748875473
```

Nokta atışı veriler için yukarıdakine benzer kodarla işinizi halledebilirsiniz.

Input:

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

Output:

		Column1	Column2	Column3
Groups	Indexes			
Group1	Index1	0.104384	-0.862022	-0.793392
	Index2	0.170685	-0.094821	-0.902629
	Index3	-1.295203	-0.013739	0.412205
Group2	Index1	0.126634	0.835351	1.024142
	Index2	-0.056693	-2.421660	-1.474845
	Index3	-0.202806	-1.070895	-0.854948
Group3	Index1	-1.130723	-1.302882	1.184730
	Index2	-0.221858	0.102638	-1.467781
	Index3	-1.532283	0.848873	0.592044

Index isimlerini öğrenme aynı şekilde sütunları da sorgulayabilirsiniz.

Input:

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

Output:

	Column1	Column2	Column3
Indexes			
Index1	0.104384	-0.862022	-0.793392
Index2	0.170685	-0.094821	-0.902629
Index3	-1.295203	-0.013739	0.412205

‘xs’ fonksiyonuna biraz değinelim. Yukarıda bildiğimiz bir koda eşit olduğunu görüyorsunuz. ‘xs’ fonksiyonu ‘loc’, ‘iloc’ gibi işlevsel fonksiyonları yerine getirebilir ve bize güzel avantajlar da sağlar.

Input:

```
df.xs('Group1').xs('Index1')
```

Output:

```
Column1    0.104384
Column2   -0.862022
Column3   -0.793392
Name: Index1, dtype: float64
```

Input:

```
df.xs('Group1').xs('Index1').xs('Column1')
```

Output:

```
0.10438364748875473
```

İlk fotoğrafta önce ‘Group’ sonra ‘Index’ değeri veriliyor. Çünkü ‘xs’ fonksiyonu ‘Group’, ‘Index’ sırasıyla çalışmaktadır. Bu durum 2. fotoğrafı da açıklamaktadır.

‘Peki her seferinde böyle uğraşacak mıyız? Nerede bunun avantajı?’ diyorsanız aşağıdaki durumlara bakabilirsiniz.

Input:

```
df.xs('Index1', level = 'Indexes')
```

Output:

	Column1	Column2	Column3
Groups			
Group1	0.104384	-0.862022	-0.793392
Group2	0.126634	0.835351	1.024142
Group3	-1.130723	-1.302862	1.184730

Input:

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

Output:

```

Groups
Group1    0.104384
Group2    0.126634
Group3   -1.130723
Name: Column1, dtype: float64

```

Bozuk Ve Kayıp Verilerle Ugraşabilmek

Bazen almış olduğumuz veriler tam, düzgün çıkmayabiliyor. Bazen veriler kayıp olup, bizi amacımıza uzaklaştırabiliyor. Bu durumlarda ne yapılması gerektiğini anlatacağız. Ama önce kayıp bir veri oluşturalım.

Input:

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

Output:

```

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

```

‘np.nan’ kodu anlaşıldığı üzere veride ‘Not a Number’ oluşturuyor.

Input:

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

Output:

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

Ve bu veri ile bu şekilde bir DataFrame'ye eşitledik.

Buradan sonra göreceğiniz adımları kesin kullanın demiyorum. Ama incelediğiniz veri, istenilen veya aranan veriye göre adımlar değişimlere tabi. Bazı veride kayıp olan yerlere ortalama, 0,1, standart sapma gibi parametreler konulabiliyor. Bazense direkt siliniyor. Bu tamamen sizin ne yapmak istediğinizde kalmıştır.

Input:

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

Output:

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

Input:

```
df.dropna()
```

Output:

	Column1	Column2	Column3

Mesela yukarıda ki kod, 'Index' satırında en az bir 'NaN' var ise siler.

Input:

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

Output:

Column1	
Index1	10.0
Index2	3.0
Index3	13.0

‘Axis’ parametresini ‘Column’a göre ayarladığımızda ise içinde hiç ‘NaN’ değeri olmayan ‘Column1’i bize döndürür ve diğerlerini siler. Aynı zamanda ‘inplace’ değerini ‘True’ vermediğiniz müddetçe kalıcı değildir.

Input:

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

Output:

	Column1	Column2	Column3
Index1	10.0	20.0	NaN
Index3	13.0	NaN	4.0

Sizce ‘thresh’ parametresi nedir? Hemen söyleyelim. ‘En az iki düzgün veri var ise silme’ anlamına gelmektedir. Elbette ki bu sayıyı siz belirliyorsunuz.

Input:

```
df.fillna(value = 0) # '' ile birlikte string değer de yazılabilir.
```

Output:

	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 içine aldığı ‘value’ değerini boş veriye atar. Örnekte, ‘0’ değeri verilmiş ve ‘NaN’ değerlerine atanmış. ‘String’ değer verilebildiği gibi oldukça da işlevseldir.

Verinin yapısının uygun olduğunu ve bizim aradığımız verinin ortalamadan bağımsız olduğunu varsayıyalım ve ‘NaN’ değerleri yerine ortalamayı atayalım.

Input:

```
df.sum()
```

Output:

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

Input:

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

Output:

```
50.0
```

İlk kodun, görüldüğü üzere her ‘Column’ değerinin toplamını, Sonraki kodun ise hepsinin toplamını verdiğini farketmişsinizdir.

Input:

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

Output:

	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 mantığı ‘value’ değerine atarsak ve 5'e bölersek (ortalama almak için), ‘NaN’ değerleri yerine ‘10’ atayacaktır.

Aynı mantık ile standart sapma, varyans gibi değerleri de yapabiliriz. Ama bunu daha önceki yazımızda açıklamıştık ve ordaki kodu buraya rahatlıkla uygulayabilirsiniz.

Bazen daha basit sorgularda oldukça işlevseldir.

Input:

```
df.size
```

Output:

9

Kaç adet veri olduğunu sorgular.

Input:

```
df.isnull()
```

Output:

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

‘True’ olan değerler ‘NaN’ı temsil etmektedir.

Input:

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

Output:

Column1	0
Column2	2
Column3	2
dtype:	int64

‘Hangi ‘Column’da kaç adet ‘NaN’ var?’ sorunun cevabı bu koddur.

Input:

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

Output:

4

Bu kod ise toplam kaç adet ‘NaN’ değeri olduğunu verir.

Eğer biz toplam kaçar adet veri olduğunu biliyorsak ve yukarıdaki kod ile de kaç adet ‘NaN’ değerini olduğunu da biliyorsak, Her ‘Column’da kaç adet düzgün veri olduğunu da bulabiliriz.

Input:
`df.size - df.isnull().sum()`
Output:

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

İşte bu kodda bahsettiğimiz düzgün veri miktarının 'Column'lara göre dağılımıdır.

GroupBy Operasyonları

GroupBy Operasyonları, Sql tablolarındankiyle tamamen aynı. Sql bilmiyor iseniz bile rahat olun. Zor bir konu değil ancak önemli bir konu. İstenilen durumu gruplar gösterir.

Input:
`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]}`
Output:

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

Verimizi oluşturuktan sonra, DataFrame'e atıyoruz.

Input:
`df = pd.DataFrame(data)`
Output:

	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

Input:

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

Output:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fc9810785f8>
```

Ve 'Salary' özelliğine göre grüplamış olduk.

Input:

```
SalaryGroupBy.sum()
```

Output:

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

Input:

```
SalaryGroupBy.min()
```

Output:

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

Input:

```
SalaryGroupBy.max()
```

Output:

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

Gördüğünüz gibi hangi fonksiyonu kullanırsanız kullanın, tablo halinde yansıtacaktır.

Input:

```
df.groupby('Salary').sum()# Bu şekilde de işlem kısaltılabilir.
```

Output:

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

Son iki fotoğraftaki işlemi yukarıdaki kod tek başına yapmaktadır.

Input:

```
df.groupby('Job').sum().loc['CEO']
```

Output:

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

'CEO'ların maaşlarının toplamı.

Input:

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

Output:

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

Input:

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

Output:

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

Numpy yazısındaki fonksiyonlar burada da kullanılabilmektedir.

Input:

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

Output:

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

Input:

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

Output:

```
5250
```

İlkinde 'Salary'e göre gruplayıp en düşük maaşları bastırdık. Sonrakinde ise en düşük maaşlı avukatın, maaşını bastırdık.

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

32500

Burada ise, mesleğe göre sıralayıp Ceo'ların ortalama maaşını bulduk. GroupBy Operasyonları, görüldüğü gibi zor bir konu değil. Oldukça kısa ve öz bir konu.

Concatenate Merge Ve Join Fonksiyonları

İlk olarak Concatenate ile başlayalım. Temel olarak birleştirme işlemini bu fonksiyon ile yapmaktayız. Aynı listelerdeki ‘zip’ fonksiyonu gibidir.

Input:
`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`
Output:

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

Input:
`df2`
Output:

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

İki tane veri seti oluşturalım. Ve aşağıdaki gibi birleştirelim.

Input:

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

Output:

	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

Şöyledir de birleştirilebilirdi.

Input:

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

Output:

	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 veriler elbetteki 'NaN' olarak atanıyor. Bu işlem yalnızca 2 değil daha fazla sayıda olan durumlarda da kullanılabilir.

```
In [1]: df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
...:                         'B': ['B0', 'B1', 'B2', 'B3'],
...:                         'C': ['C0', 'C1', 'C2', 'C3'],
...:                         'D': ['D0', 'D1', 'D2', 'D3']},
...:                        index=[0, 1, 2, 3])
...:

In [2]: df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
...:                         'B': ['B4', 'B5', 'B6', 'B7'],
...:                         'C': ['C4', 'C5', 'C6', 'C7'],
...:                         'D': ['D4', 'D5', 'D6', 'D7']},
...:                        index=[4, 5, 6, 7])
...:

In [3]: df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
...:                         'B': ['B8', 'B9', 'B10', 'B11'],
...:                         'C': ['C8', 'C9', 'C10', 'C11'],
...:                         'D': ['D8', 'D9', 'D10', 'D11']},
...:                        index=[8, 9, 10, 11])
...:

In [4]: frames = [df1, df2, df3]

In [5]: result = pd.concat(frames)
```

df1				
	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2				
	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3				
	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Result

Result				
	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Join, ile devam edelim. ‘Join’ fonkiyonları adı üstünde birbirlerine katılma işlemleri olarak düşünülebilir. Ve lisedeki ‘kümeler’ konusu olarak da düşünülebilir. Öyleyse iki veri oluşturup bu işlemleri yapmaya başlayalım.

Input:

```
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

Output:

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

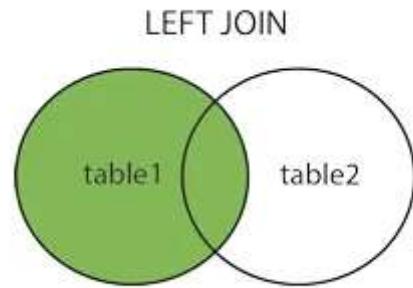
Input:

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

Output:

	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' ile başlayalım.



Left Join, Sql'deki 'Left Join' ile birebir aynıdır. İşlevi ise yukarıda verilmiştir. Siz burada 'df1'i table1 olarak, 'df2'yi table2 olarak düşünmelisiniz.

Input:

df1.join(df2)

Output

	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 tersini yaparsak sizce ne olur ?

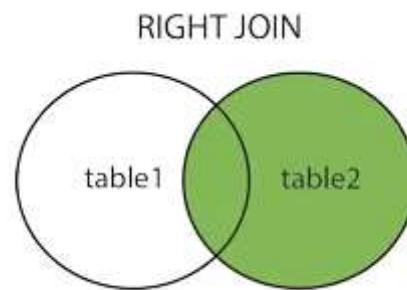
Input:

df2.join.(df1)

Output:

	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, ile de aynı mantık yürütülebilir. İşlevi ve kodu aşağıda verilmiştir.



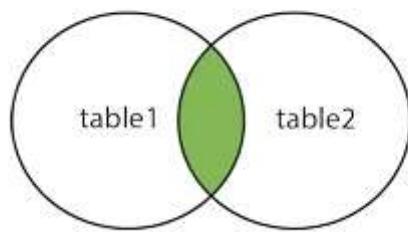
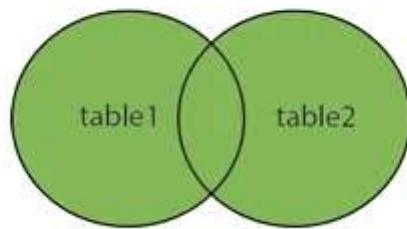
Input:

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

Output:

	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

Sırada 'inner' ve 'outer' join fonksiyonları var.

INNER JOIN**FULL OUTER JOIN**

‘Full Outer Join’ yazdığını bakmayın. O Sql’deki ismi, burada sadece ‘outer’ olarak geçiyor ve aşağıda ikisinin de örnekleri var.

Input:

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

Output:

	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

Input:

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

Output:

	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

Input:`df1.join(df2, sort = 'True')`**Output:**

	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

Input:`df1.join(df2, sort = 'False')`**Output:**

	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

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

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:2: FutureWarning: Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.
```

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

Siyah olan yazı hata değil bir uyarı mesajıdır. Çok dikkate almayınız. Bazen sürümle alakalı uyarı verebiliyor.

Aşağıda ise ‘Join’in tüm parametreleri yer almaktır ve öğrendiklerimize kıyasla daha az kullanılmaktadır. İsterseniz araştırabilirsiniz.

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).

Parameters:

Merge işlemi, join işlemi ile oldukça benzemekle beraber bazı farklı özelliklere sahiptir. Aşağıda daha detaylı anlatacağız.

right : *DataFrame or named Series*

Object to merge with.

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

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 : *tuple of (str, str), default ('_X', '_Y')*Suffix to apply to overlapping column names in the left and right side, respectively.
To raise an exception on overlapping columns use (False, False).**copy** : *bool, default True*

If False, avoid copy if possible.

indicator : *bool or str, default False*

If True, adds a column to output DataFrame called "_merge" with information on the source of each row. If string, column with information on source of each row will be added to output DataFrame, and column will be named value of string. Information column is Categorical-type and takes on a value of "left_only" for observations whose merge key only appears in 'left' DataFrame, "right_only" for observations whose merge key only appears in 'right' DataFrame, and "both" if the observation's merge key is found in both.

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.

New in version 0.21.0.

Yukarıda Merge işleminin tüm parametrelerini görmektesiniz. Ama pek çogunu kullanmayacağız.

Input:

```
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
```

Output:

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

Input:

df2

Output:

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

Verimizi de oluşturduğumuza göre başlayabiliriz.

Input:

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

Output:

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

Ve ilk parametremizle tanıştık bile. ‘On’ parametresine anahtarınız gerekmektedir. Anahtar sizin verinizde önemli yeri olmalıdır ki gerçek manada ‘anahtar’ olabilsin.

Input:

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

Output:

	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

‘How’ parametresine zaten Join konusundan alışğız. Orada gördüğünüz ‘inner’, ‘outer’, ‘right’ değerleri burada da geçerli.

Input:

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

Output:

	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

Input:

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

Output:

	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

Input:`pd.merge(df1,df2, on = 'Key', how = 'inner')`**Output:**

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

Input:`pd.merge(df1,df2, on = 'Key', how = 'right',right_index=True)`**Output:**

	A	B	Key	X	Y
1.0	A1	B1	K1	X1	Y1
2.0	A2	B2	K2	X2	Y2
3.0	A3	B3	K3	X3	Y3
NaN	NaN	NaN	K4	X4	Y4

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

	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

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

	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' ve 'right_index' parametreleri ise fotoğraflardan anlaşılacak kadar basit. Diğer parametreler hemen hemen kullanılmaz ama isterseniz Pandasın kendi sitesinden araştırabilirsiniz.

DataFrame Operasyonları Ve Pivot Table

Burada küçük ama işlevsel fonksiyonları gördükten sonra 'Pivot Table' konusunu işleyeceğiz. Önce verimizi oluşturalım ve başlayalım.

Input:

```
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

Output:

	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

Ve küçük ama işlevsel fonksiyonlarımıza gelelim.

Input:

df.head()

Output:

	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

Input:`df.head(n = 2) # df.head(2) = df.head(n = 2)`**Output:**

	Column1	Column2	Column3
0	1	1000	Mace Windu
1	2	1000	Darth Vader

‘head()’ ilk 5 değeri göstermekte ama içine aldığı değer kadar da gösterebilir.

Örnekte 2 verilmiş. Aynı şekilde ‘tail()’de son 5 değeri gösterir. O da ‘head()’ gibi değer alabilir.

Input:`df.describe()`**Output:**

	A	B	Key
count	3	3	3
unique	3	3	3
top	A1	B3	K3
freq	1	1	1

Input:`df.info`**Output:**

```
<bound method DataFrame.info of      A   B Key>
1  A1  B1  K1
2  A2  B2  K2
3  A3  B3  K3>
```

‘describe()’ bazı istatistiksel bilgiler verirken, ‘corr()’ korelasyon bilgisini verir. Ama bu data ile uyumlu olmadığı için aşağıda açıklayacağız. ‘info()’ ise adından da anlaşılacağı gibi temel bilgiler veriyor. Bu fonksiyonlar veriye ilk bakışınız olacağı için kullanmanızı tavsiye ederiz.

Input:

```
df['Column2'].unique()
```

Output:

```
array([1000, 2000, 3000])
```

Input:

```
df['Column2'].nunique()
```

Output:

3

‘unique()’ verilen verinin kaç adet ‘eşsiz’ verisi olduğunu bize verir. Aşağıdaki kod ise kaç adet ‘eşsiz’ değer olduğunu veriyor.

Input:

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

Output:

1000	3
3000	2
2000	1

```
Name: Column2, dtype: int64
```

‘value_counts()’ ise hangi değerden kaç adet olduğunu veren bir fonksiyon.

Input:

```
df[df['Column1'] >= 2]
```

Output:

	Column1	Column2	Column3
1	2	1000	Darth Vader
2	3	2000	Palpatine
3	4	3000	Kylo Ren
4	5	3000	Rey
5	6	1000	Obi-Wan

Input:`df[(df['Column1'] >= 2) & (df['Column2'] == 300)]`**Output:**

	Column1	Column2	Column3
1	2	300	Dart Vader
3	4	300	Thanos
4	5	300	Harley Quinn

Anlaşıldığımız üzere ‘DataFrame Filtreleme’ işlemlerini de burada kullanabiliyoruz.

Input:`def square(x):
 return x ** 2``square(2)`**Output:**

4

Burada bir Fonksiyon oluşturup içine aldığı değerin karesini vermesini sağladık.

Input:`df['Column2'].apply(square)`**Output:**

```
0    40000
1    90000
2    40000
3    90000
4    90000
5   160000
Name: Column2, dtype: int64
```

Bu fonksiyonu ‘apply()’ ile verimize attığımızda bize karelerini verdi.

Bu özel durumu projelerinizde kullanabilirsiniz. Örneğin, aracın motor kısmının verisini inceliyorsunuz ve ekibiniz sizden motora ait bir bulgu istedi. Onlardan gerekli olan matematiksel formülü alıp aynı bu örnekteki gibi kullanıp rahatlıkla işinizi yapabilirsiniz.

Ama bunun daha kısa bir yolu var.

Input:

```
lambda x : x **2
```

Output:

```
<function __main__.<lambda>(x)>
```

Input:

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

Output:

0	40000
1	90000
2	40000
3	90000
4	90000
5	160000

name: Column2, dtype: int64

'lambda' genelde daha kısa olmasından ötürü daha çok tercih edilir .

```
# df['Column2'] = df['Column2'].apply(square) # Güncelleyebilmek  
için bunu kullanıyoruz. Ayrıca lambda'yıda kullanabirdik.
```

Ama bu işlemleri yapınca veriniz otomatik olarak güncellenmiyor. Sizin güncellemeniz gerekiyor, onu da yukarıdaki gibi yapıyoruz.

Input:

```
df['Column3']
```

Output:

```

0      John Wick
1      Dart Vader
2      Joker
3      Thanos
4      Harley Quinn
5      Andrew NG <3
Name: Column3, dtype: object

```

Input:`df['Column3'].apply(len) # String'in uzunluğunu alma`**Output:**

```

0      9
1     10
2      5
3      6
4     12
5     12
Name: Column3, dtype: int64

```

Input:`df.drop('Column3', axis = 1)`**Output:**

	Column1	Column2
0	1	40000
1	2	90000
2	3	40000
3	4	90000
4	5	90000
5	6	160000

Input:`df.index`**Output:**

```
Index(['Column1', 'Column2', 'Column3'], dtype='object')
```

Input:`df.index.names`**Output:**

```
FrozenList([None])
```

Yukarıdaki fonksiyonların bazılarını daha önce işlemiştik. Muhtemelen tek anlatmadığım ‘df.index’ fonksiyonu. O da şöyle ‘0’dan başlamış ‘6’ya kadar ‘1’er ‘1’er ilerlemiş. Altındaki anlamı bu kadar basit. Şimdi ise ‘sort_values()’e geçelim.

Aşağıda aldığı parametreleri görürorsunuz. ‘by’ neye göre alınacağını belirtirken, ‘ascending’ sıralamasını neye göre olacağını belirtir. Küçükten büyüğe ya da büyükten küçüğe olarak sıralayabilirsiniz. ‘Axis’ ve ‘Inplace’ parametrelerini zaten biliyorsunuz. ‘kind’ parametresi aşağıda yazan 3 adete göre verinizi düzenleyebilirsiniz. Çok kullanılmaz ama isterseniz araştırabilirsiniz. ‘na_postion’ ise ‘NaN’ değerlerinin pozisyonunu belirtmenizi sağlar.

Parameters:	<p>by : str or list of str Name or list of names to sort by. • if axis is 0 or ‘index’ then by may contain index levels and/or column labels • if axis is 1 or ‘columns’ then by may contain column levels and/or index labels <small>Changed in version 0.23.0: Allow specifying index or column level names.</small></p> <p>axis : {0 or ‘index’, 1 or ‘columns’}, default 0 Axis to be sorted.</p> <p>ascending : bool or list of bool, default True Sort ascending vs. descending. Specify list for multiple sort orders. If this is a list of bools, must match the length of the by.</p> <p>inplace : bool, default False If True, perform operation in-place.</p> <p>kind : {‘quicksort’, ‘mergesort’, ‘heapsort’}, default ‘quicksort’ Choice of sorting algorithm. See also ndarray.np.sort for more information. <small>mergesort</small> is the only stable algorithm. For DataFrames, this option is only applied when sorting on a single column or label.</p> <p>na_position : {‘first’, ‘last’}, default ‘last’ Puts NaNs at the beginning if <i>first</i>; <i>last</i> puts NaNs at the end.</p>
Returns:	sorted_obj : DataFrame or None DataFrame with sorted values if <i>inplace=False</i> , None otherwise.

Input:

df

Output:

	Column1	Column2	Column3
0	1	40000	John Wick
1	2	90000	Dart Vader
2	3	40000	Joker
3	4	90000	Thanos
4	5	90000	Harley Quinn
5	6	160000	Andrew NG <3

Input:

```
df.sort_values(by=['Column1', 'Column3']) # Birden fazla değerde  
'Sort' edilebilir.
```

Output:

	Column1	Column2	Column3
0	1	200	John Wick
1	2	300	Dart Vader
2	3	200	Joker
3	4	300	Thanos
4	5	300	Harley Quinn
5	6	400	Andrew NG <3

Bu fonksiyon belli bir değere göre sıralamamızı sağlar. Biz burada 'Column2'ye göre sıralamasını sağladık. Yukarı da ise 'Column1' ve 'Column3'e sıraladık ki farkı görebilesiniz.

Input:

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

Output:

	Column1	Column2	Column3
0	1	40000	John Wick
1	2	90000	Dart Vader
2	3	40000	Joker
3	4	90000	Thanos
4	5	90000	Harley Quinn
5	6	160000	Andrew NG <3

Input:

```
df.sort_values('Column2', ascending = True) # Küçükten Büyüge
```

Output:

	Column1	Column2	Column3
0	1	40000	John Wick
2	3	40000	Joker
1	2	90000	Dart Vader
3	4	90000	Thanos
4	5	90000	Harley Quinn
5	6	180000	Andrew NG <3

Input:`df.sort_values('Column2', ascending = False) # Büyüktен küçüğe`**Output:**

	Column1	Column2	Column3
5	6	180000	Andrew NG <3
1	2	90000	Dart Vader
3	4	90000	Thanos
4	5	90000	Harley Quinn
0	1	40000	John Wick
2	3	40000	Joker

Input:`df.sort_values('Column1', kind = 'heapsort')`**Output:**

	Column1	Column2	Column3
0	1	40000	John Wick
1	2	90000	Dart Vader
2	3	40000	Joker
3	4	90000	Thanos
4	5	90000	Harley Quinn
5	6	180000	Andrew NG <3

Input:`df.sort_values('Column1', kind = 'mergesort')`**Output:**

	Column1	Column2	Column3
0	1	40000	John Wick
1	2	90000	Dart Vader
2	3	40000	Joker
3	4	90000	Thanos
4	5	90000	Harley Quinn
5	6	160000	Andrew NG <3

Input:`df.sort_values('Column1', kind = 'quicksort')`**Output:**

	Column1	Column2	Column3
0	1	40000	John Wick
1	2	90000	Dart Vader
2	3	40000	Joker
3	4	90000	Thanos
4	5	90000	Harley Quinn
5	6	160000	Andrew NG <3

Input:`df.sort_values('Column1', na_position = 'first')`**Output:**

	Column1	Column2	Column3
0	1	40000	John Wick
1	2	90000	Dart Vader
2	3	40000	Joker
3	4	90000	Thanos
4	5	90000	Harley Quinn
5	6	160000	Andrew NG <3

Input:`df.sort_values('Column1', na_position = 'last')`**Output:**

	Column1	Column2	Column3
0	1	40000	John Wick
1	2	90000	Dart Vader
2	3	40000	Joker
3	4	90000	Thanos
4	5	90000	Harley Quinn
5	6	180000	Andrew NG <3

Verimiz bazı parametreler için (kind ve na_position) için pek de uygun değil. Bazıları aynı sonucu verdi. Ama biz nasıl yapıldığını da göstermek istedik. Dediğimiz gibi çok kullanılmaz, isterseniz araştırabilirsiniz.

Gelelim 'Pivot Table' konusuna. Excel biliyorsanız korkacak bir şey yok. Çünkü tamamıyla aynı. Bilmiyorsanız da endişelenmeyin. Çünkü çok basit bir kullanımı var.

Input:

```
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]})  
df
```

Output:

	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

Başlamadan önce 'corr()'dan bahsediceğimizi söylemiştık. Burada onu anlatalım(her ne kadar data uygun olmasa da). Önceden bahsettiğimiz gibi korelasyonu anlatmaktadır.

Input:
`df.corr()`
Output:

moisture	
moisture	1.0

3 farklı şehirde, 3 farklı aydaki, 3 yüzdesel nem oranımız var diyelim.

Input:
`df.pivot_table(index = 'Month', columns = 'State', values = 'moisture')`
Output:

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

Input:
`df.pivot_table(index = 'State', columns = 'Month', values = 'moisture')`
Output:

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

Yukarıdaki gibi pivot table oluşturabilir. İşte bu kadar basit.

Veri Setini Okuma Yöntemleri

Bu konu aslında aşağıdaki ile açıklanabilir.

```
dataset = pd.read_csv('Dosya_Yolu\Data_Name.csv') # Pc'de IDE'nizde
bu şekilde okuyabilirsiniz.

df=pd.read_csv("../input/Data_Name.csv") # Kaggle kernel yazmak
için( ama güncelleme geldi bazen çalışmayıor)

dataset.to_csv('Data_Name') # Datayı tekrardan csv'ye
dönüştürebilmek için

excelset = pd.read_excel('Excels_Name.xlsx') # Excel Dosyasını
okumak için

excelset.to_excel('excelnewfile.xlsx') # Excel dosyasına çevirme

New = pd.read_html('Datasetin urlsi.html') # internette ki bir
veriyi okumak için
```

Normalde burada bitirecektim ama son zamanlarda oldukça güncel ve epey tutacağımı düşündüğüm bir özelliği paylaşmak istiyorum. İsmi ‘Pandas_Profiling()’.

```
from sklearn.datasets import load_boston

boston = load_boston()

df = pd.DataFrame(data = boston.data, columns = boston.feature_names)

df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Öncelikle datamızı import etmemiz, internet ortamında değilsek (offline ise) ‘path’ vermemiz gereklidir.

In [11]: `import pandas_profiling as pp`
`pp.ProfileReport(df)`
`# Returns Pandas Profiling Report which then can be accessed for our data analysis`

Note: df is our dataframe

Pandas Profiling Report

Overview Variables Correlations Missing values Sample

Overview

Returns Pandas Profiling Report

Daha sonra yukarıda ki gibi işlemler yapılır sonra ise...

The screenshot shows a Jupyter Notebook interface with the following sections:

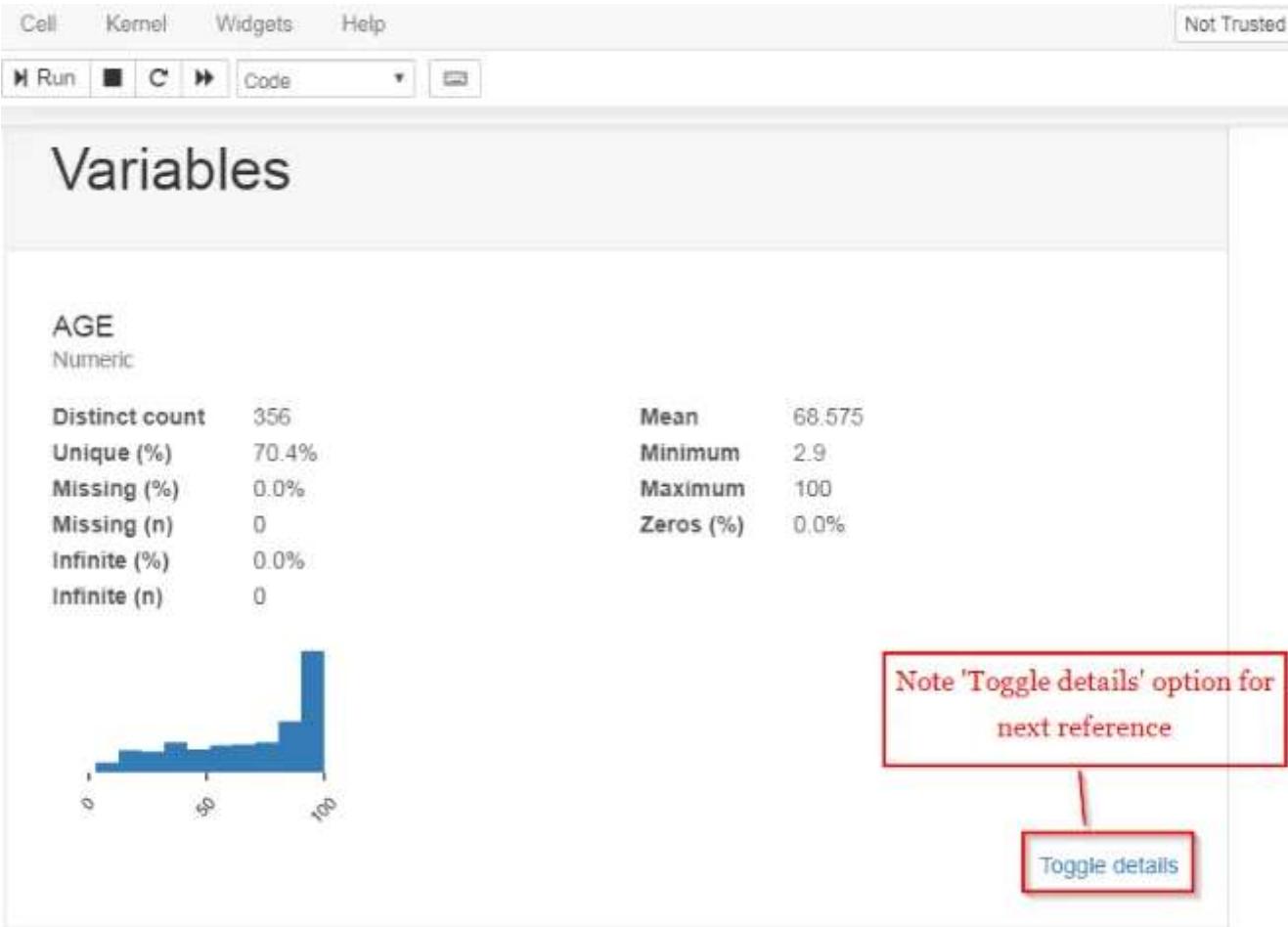
- Toolbar:** Insert, Cell, Kernel, Widgets, Help, Run, Stop, Kernel, Code, Cell Type.
- Section 1: Dataset info**

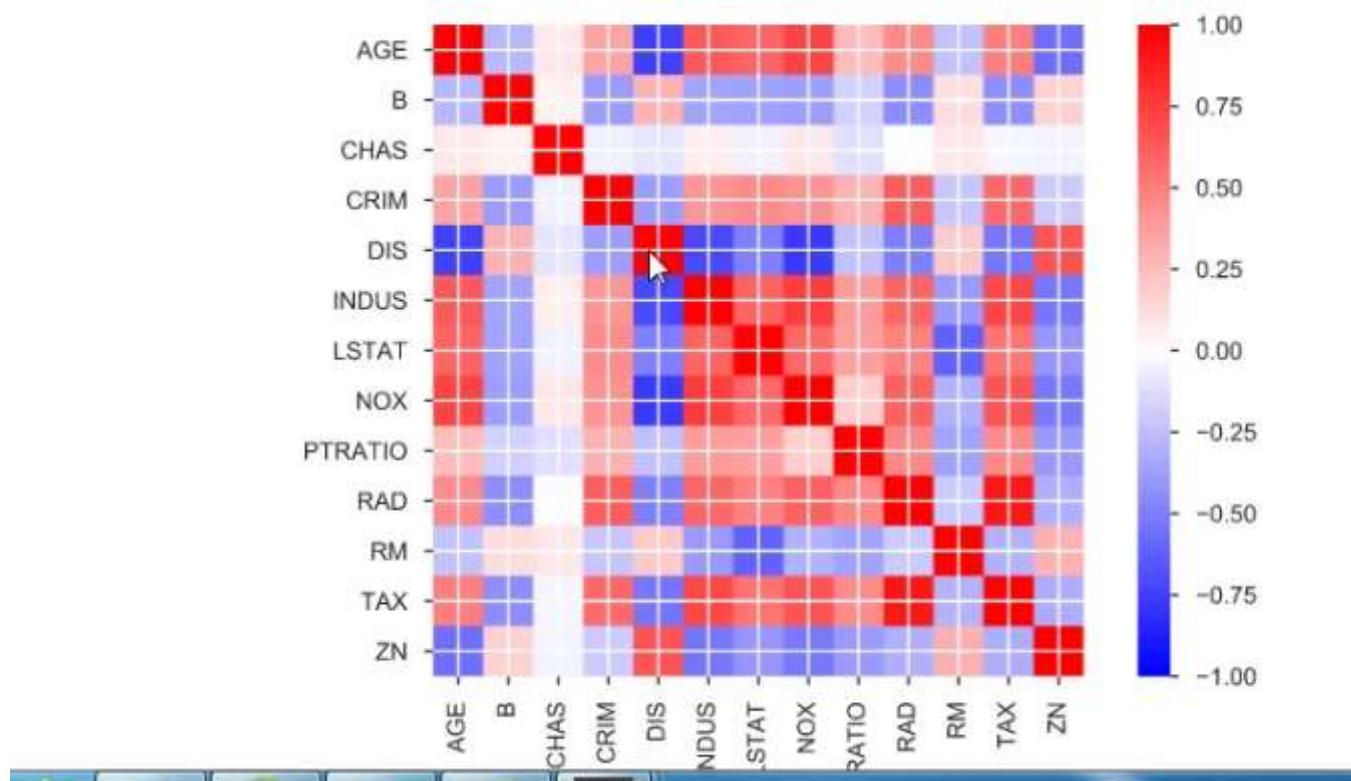
Number of variables	13
Number of observations	506
Missing cells	0 (0.0%)
Duplicate rows	0 (0.0%)
Total size in memory	51.5 KiB
Average record size in memory	104.2 B
- Section 2: Variables types**

Numeric	11
Categorical	0
Boolean	1
Date	0
URL	0
Text (Unique)	0
Rejected	1
Unsupported	0
- Section 3: Warnings**

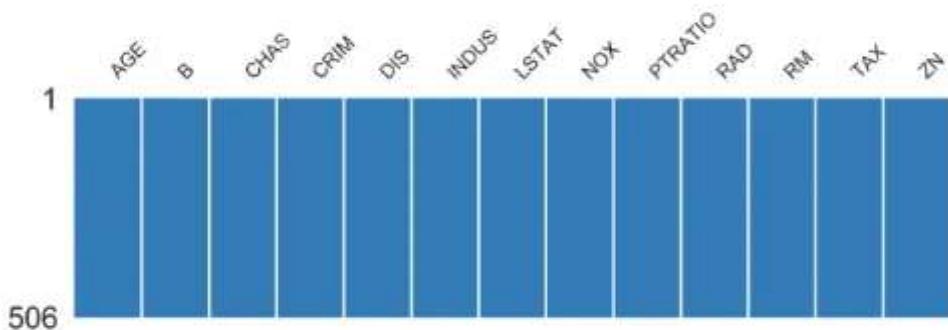
TAX is highly correlated with RAD ($\rho = 0.91023$) Rejected

ZN has 372 (73.5%) zeros Zeros



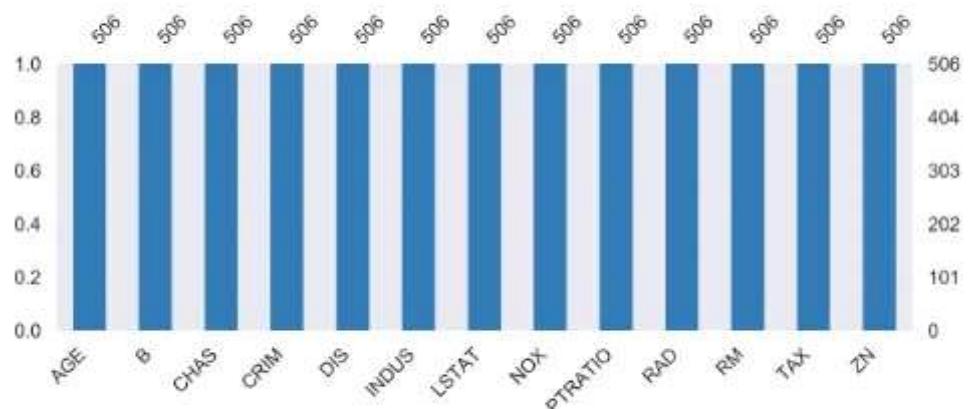
[Pearson](#)[Spearman](#)[Kendall](#)[Phi_k](#)

Matrix



Matris Grafiği

Count



Grafik Say

Yukarıda görüldüğü gibi pek çok özelliği içinde barındırmaktadır. Ayrıca Daha çok güncel bir konu olduğu için deneyimleyemedim ve bu fotoğrafları yabancı kaynaklardan almak zorunda kaldım.

Son Söz

Evet arkadaşlar buraya kadar geldiyseniz ve gerçekten hakkını verdiyseniz kendiniz için 'İyi seviyede Pandas biliyorum' diyebilirsiniz. Ama bununla yetinmemeli pek çok farklı kaynağı araştırmalı ve çalışmalısınız. Zamanı geldikçe yazı güncellenecektir. Daha detaylı halini ilerleyen zamanlarda Kaggle sayfamda paylaştım buradan ulaşabilirsınız.

Yazımı okuduğunuza ve kıymetli zamanınızı ayırdığınız için müteşekkirim.
Sağlıcakla kalın ve öğrenmeyi kesmeyin.

```
print("İyi Günler !")
```

[Data Science](#)[Data](#)[Pandas](#)[Tutorial](#)[Türkçe](#)[Follow](#)

Written by Batuhan Bayraktar

223 Followers · Writer for Deep Learning Türkiye

Üniversite öğrencisi, bilim ve sanat düşkünu, yazılım meraklısı ve amatör girişimci

More from Batuhan Bayraktar and Deep Learning Türkiye

More thorough example:

```
>>> a = [3,6,9]
>>> b = np.array(a)
>>> b.T
array([3, 6, 9])          #Here it didn't transpose because 'a' is 1 dimensional
>>> b = np.array([a])
>>> b.T
array([[3],
       [6],
       [9]])                  #Here it did transpose because a is 2 dimensional
```

Use numpy's `shape` method to see what is going on here:

```
>>> b = np.array([10,20,30])
>>> b.shape
(3,)
```

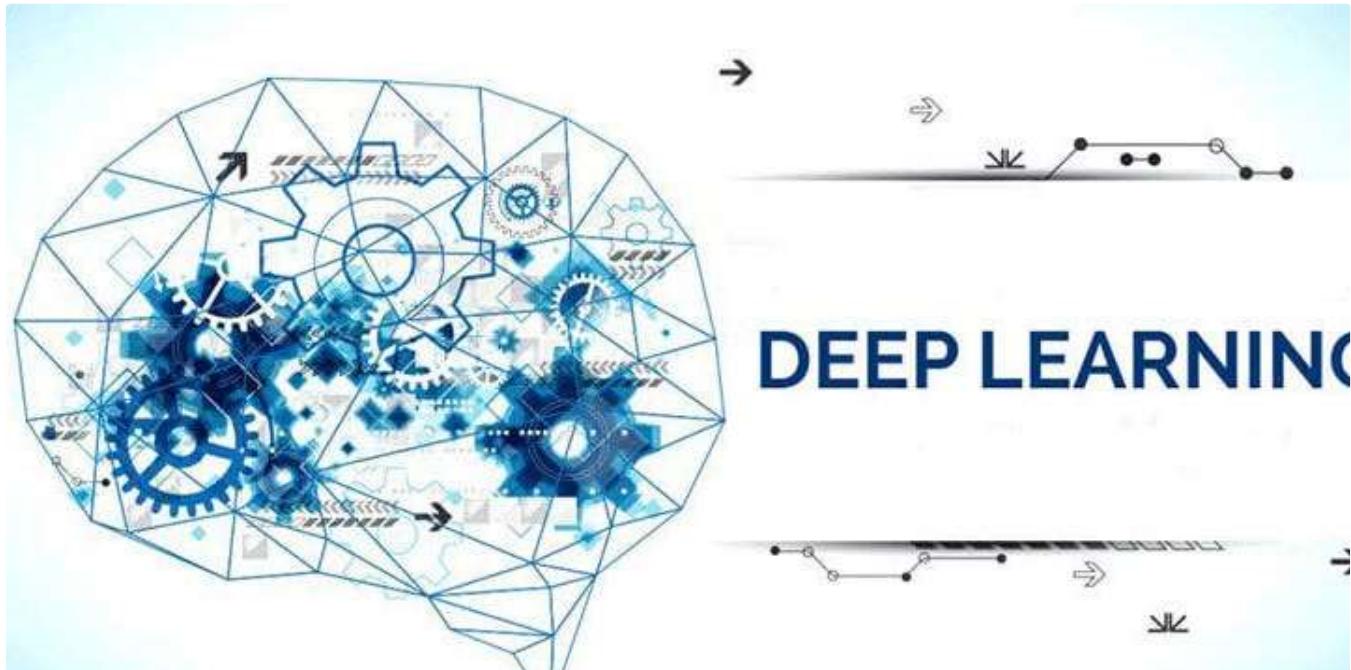
 Batuhan Bayraktar in Deep Learning Türkiye

Türkiye'nin En Kapsamlı Numpy Tutorialı

NumPy, Python programlama dili için bir kütüphanedir; büyük, çok boyutlu diziler ve matrisler için destek ekler ve bu dizilerde çalışacak...

12 min read · Oct 5, 2019

👏 558 🎧 2



 Necmettin Çarkacı in Deep Learning Türkiye

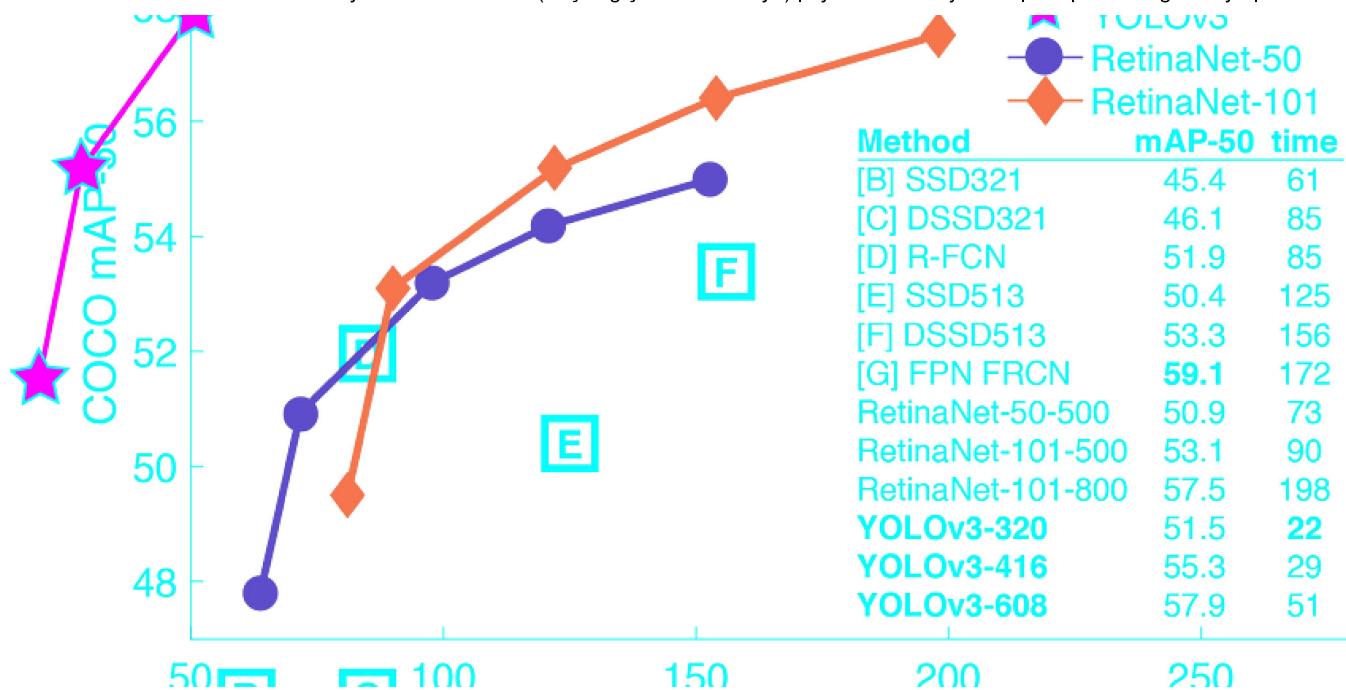
Derin Öğrenme Uygulamalarında En Sık kullanılan Hiper-parametreler

Derin Öğrenme Uygulamalarında En Sık kullanılan Hiper-parametreler ve Bu Parametrelerin Bazı Temel Özellikleri.

14 min read · Jan 22, 2018

👏 2.9K 🎧 13





Yiğit Mesci in Deep Learning Türkiye

YOLO Algoritmasını Anlamak

Son yıllarda nesne tespiti alanında revaçta olan YOLO (You Only Look Once) algoritmasını ister istemez duymuşsunuzdur. Bu algoritma neden...

7 min read · Apr 29, 2019

610 4

...



Batuhan Bayraktar in Türkçe Yayın

Elon Musk ve Başarı İlişkisi

Selamlar arkadaşlar bugün sizlerlerle Elon Musk'ın Girişimlerini ve Başarılarını konuşacağız ve önemli dersler çıkaracağız. Çocukluğundan...

2 min read · Jan 9, 2020

👏 86

🗨 1



...

See all from Batuhan Bayraktar

See all from Deep Learning Türkiye

Recommended from Medium



Sheila Teo in Towards Data Science

How I Won Singapore's GPT-4 Prompt Engineering Competition

A deep dive into the strategies I learned for harnessing the power of Large Language Models (LLMs)

⭐ · 24 min read · Dec 29, 2023

9.8K

119



...



Anmol Tomar in CodeX

Say Goodbye to Loops in Python, and Welcome Vectorization!

Use Vectorization—a super-fast alternative to loops in Python

◆ 5 min read · Dec 28, 2023

3.3K

39



...

Lists



Predictive Modeling w/ Python

20 stories · 824 saves



Practical Guides to Machine Learning

10 stories · 953 saves



data science and AI

39 stories · 49 saves



Coding & Development

11 stories · 398 saves



 James Presbitero Jr. in Practice in Public

These Words Make it Obvious That Your Text is Written By AI

These 7 words are painfully obvious. They make me cringe. They will make your reader cringe.

4 min read · Jan 1

 23K  638



 Ignacio de Gregorio in Towards AI

Is Mamba the End of ChatGPT As We Know It?

The Great New Question

◆ 8 min read · Jan 11

👏 5.3K ⚡ 61



A screenshot of the GitHub Copilot interface. On the left, there's a sidebar with various icons for file operations like copy, paste, and search. The main area shows a conversation between GitHub Copilot and a user (@monalisa). Copilot says, "Hi @monalisa, how can I help you? I'm powered by AI, so surprises and mistakes are possible. Make sure to verify any generated code or suggestions, and share feedback so that we can learn and improve." Below this, there's a large text input field containing a snippet of Python code:

```
1 import datetime
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```

At the bottom of the interface, there's a footer bar with the text "Ask a question or type '?' for commands".

⌚ Jacob Bennett in Level Up Coding

The 5 paid subscriptions I actually use in 2024 as a software engineer

Tools I use that are cheaper than Netflix

◆ 5 min read · Jan 4

👏 4K ⚡ 51



 Barr Moses

Top 10 Data & AI Trends for 2024

From LLMs transforming the modern data stack to data observability for vector databases, here are my predictions for the top data...

6 min read · 4 days ago

 494 9

...

[See more recommendations](#)