Pandas Nədir?

Pandas istifadəsi rahat və yüksək peformanslı data struktur və data analizi alətləri təklif edən açıq mənbəli bir python kitabxanasıdır. Pandas, 'csv', 'excel', 'feather', 'fwf', 'gbq', 'hdf', 'html', 'json', 'orc', 'parquet', 'pickle', 'sas', 'spss', 'sql', 'sql_query', 'sql_table', 'stata', 'table', 'xml' fayllarını açmaq və clipboard -dan dataları kopyalamaq eləcə də, asanlıqla datanı oxumaqla arzulanan nəticəyə çatmaq üçün istifadə edilir. Digər bir ifadə ilə, müxtəlif faylları açmaq və onun daxilində sətir və ya sütun seçməklə onların üzərində işlədiyimiz üçün Pandas -a təşəkkür edə bilərik. Digər bir xüsusiyyət, Numpy -da yaradılmış datanın formalaşdırılması prosesini daha detallı apara bilərik. Pandas ilə, ağlınıza gələ biləcək bir çox şeyi uğurla həm də, bir neçə sətir kodla edə bilərsiniz. Misal üçün, ilk qara ddəliyin görüntüsünün datası Pandas ilə işlənilmişdir. Pandas həmçinin sürətli olması üçün optimizasiya edilmişdir və çox sürətlə çalışa bilər.Pandas -ı import etməklə başlayaq:

```
import numpy as np
import pandas as pd
```

	BandName	WavelengthMax	WavelengthMin
0	CoastalAerosol	450	430
1	Blue	510	450
2	Green	590	530
3	Red	670	640
4	NearInfrared	880	850
5	ShortWaveInfrared_1	1650	1570
6	ShortWaveInfrared_2	2290	2110
7	Cirrus	1380	1360

Pandas Series -ları:

Seriallar numpy massivlərinə çox bənzəyir, çünki onlar Numpy sekansları əsasında yaradılıblar. Seriya etiketlənmiş (adlandırılmış) verilənlərdən ibarət birölçülü data strukturudur. Etiket dəyərlərinə indeks deyilir. Məlumatın özü ədədlərdən, stringlər və ya digər Python obyektlərindən ibarət ola bilər. Series -ləri yaratmaq üçün list -lər, sequences və ya dictionary -lardan istifadə edilə bilər. Pandas series -lər 5 parametr qəbul edə bilir. Bunlar data, index, dtype, copy və name -dir. Bunlardan dtype və copy sizə tanış gəlməyə bilər. dtype data tipi üçün istifadə edilir - əgər dəyəri None olarsa, data tipi təxmini olaraq avtomatik təyin ediləcək. Copy data -nı kopyalamaq üçündür)). Copy -dən istifadə önəmlidir. Data -da dəyişiklik edəcəksinizsə, onun bir nüsxəsini yaradın. Mövcud datada dəyişikliklər etməniz sizdə bəzən data itkilərinə və zaman israflarına yol açа bilər.

Parameter	Description
data	array-like, Iterable, dict, or scalar value.Contains data stored in Series. If data is a dict, argument order is maintained.
index	array-like or Index (1d).Values must be hashable and have the same length as `data`. Non-unique index values are allowed. Will default to RangeIndex (0, 1, 2, ..., n) if not provi
dtype	str, numpy.dtype, or ExtensionDtype, optional. Data type for the output Series. If not specified, this will be inferred from `data`. See the :ref:`user guide ` for more usages.
copy	bool, default False. Copy input data. Only affects Series or 1d ndarray input. See examples.

Bir serie yaradaq:

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

```
I           1
am          2
Learning    3
Data        4
Science     5
dtype: int64
```

Bunu qısaca belə də edə bilərsiz:

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

```
100    a
101    b
102    c
```

```
103    d
dtype: object
```

Üç ədəd series yaradaq

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

```
A
```

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

```
B
```

```
Michael_s exam result    44
dtype: int64
```

```
C
```

```
Darth_Vader_s exam result    99
dtype: int64
```

Bu series -lər üzərində əməliyyat aparaq.

```
A + B
```

```
Michael_s exam result    79.0
Olivia_s exam result      NaN
dtype: float64
```

Bu əməliyyatlardan ilki: iki series arasında dörd və daha çox əməliyyat edilə bilər. Bununla belə, bilməli olduğunuz bir şey var: 'Michael' və 'Olivia' dəyərləri 'A' dəyişənində mövcud olsa da, 'B' dəyişəninin daxilində yalnız 'Michael'in imtahan nəticəsi vardır . İki dəyişənlə işlədiyimiz zaman yalnız uyğun gələn dəyərlər emal edilir (burada, toplanır). Uyğun olmayan dəyər varsa (bu misaldakı 'Olivia' adlı şəxs), bu dəyər 'NaN' kimi görünəcək. Burada, 'NaN' , 'Not a Number' deməkdir

```
DataDict4 = pd.concat([C,A],axis=0)
DataDict4
```

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

Yuxarıda gördüyümüz kimi biz "DataDict4" dəyişəni yaratdıq, "A" dəyişənini "C" dəyişəni ilə birləşdirdik (append metodunun istifadəsi dayandırılıb). Nəticədə biz, 3 şəxsin də imtahan nəticəsini görə bilirk. Əgər “Darth Vader” belə yüksək bal toplayıbsa, bu, yəqin ki, “gücün qaranlıq tərəfi” ilə bağlı imtahandır. :)

```
A['Michael_s exam result']
```

```
35
```

```
C['Darth_Vader_s exam result']
```

```
99
```

Yuxarıda olan kodu açıqlayım: ‘A’ dəyişənində ‘Michael’ və ‘Olivia’ adlı şəxslərin imtahan nəticələri vardı. Tutaq ki, biz, ‘Michael’ -ın imtahan nəticəsini bilmək istəyirik. Bu vəziyyətdə ilk kodu yazmağımız lazımdır. Eynisi 'Darth Vader' adlı şəxs üçün də labüddür. Bu şəxslərin aldığı imtahan nəticələri və ya digər bir series -lərdə ədədi dəyişənin dəyər olaraq təyin edilməsi məcburi deyil (int veya float). ‘String’ yəni söz və ya mətn simvollarından ibarət olan bir data da ola bilərdi.‘Series’ mövzusunu əslində bu qədər qısa ifadə edə bilərik.))

DataFrame -in yaradılması və onun funksiyaları ilə iş:

DataFrames, Pandas kitabxanasında əsas işlərin görüldüyü yerdir və biz bu hissədə bir çox əməliyyatlar edəcəyik. Burada sütunlar 'Column' və ya 'Feature', sətirlər isə 'Row' və ya 'Index' adlanır. Hər şeydən əvvəl özümüz üçün data yaradaq və sütunları seçməyi öyrənək:

```
from numpy.random import randn
df = pd.DataFrame(data = randn(5,5),
                  index = ['A','B','C','D','E'],
                  columns = ['Columns1','Columns2','Columns3','Columns4','Columns5'])
df
```

	Columns1	Columns2	Columns3	Columns4	Columns5
A	-0.828852	0.654459	-0.508973	-1.139647	-0.538566
B	1.003648	-0.254032	-1.005248	-1.624998	-2.634857
C	-1.929049	-0.051853	-0.738681	0.109531	-0.738103
D	0.518918	0.541574	-1.034584	0.841593	0.148018
E	1.081142	0.961920	-0.556315	0.290454	-0.715656

Əgər "randn" funksiyasını ilk dəfə görürsünüzsə, Numpy haqqında məqaləni oxuya bilərsiniz.

İşlədiyiniz verilənlərdə sizə lazım olan sütunu aşağıdakı üsulla əldə edə bilərsiniz.

```
df['Columns1']
```

```
A   -0.828852
B    1.003648
C   -1.929049
D    0.518918
E    1.081142
Name: Columns1, dtype: float64
```

Üstəlik, eyni üsulla təkcə 1 sütun deyil, həm də birdən çox sütun əldə edə bilərsiniz.

```
df[['Columns1','Columns5']]
```

	Columns1	Columns5
A	-0.828852	-0.538566
B	1.003648	-2.634857
C	-1.929049	-0.738103
D	0.518918	0.148018
E	1.081142	-0.715656

Yeni bir sütun belə əlavə edilir:

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

	Columns1	Columns2	Columns3	Columns4	Columns5	Columns6
A	-0.828852	0.654459	-0.508973	-1.139647	-0.538566	0.740678
B	1.003648	-0.254032	-1.005248	-1.624998	-2.634857	1.122299
C	-1.929049	-0.051853	-0.738681	0.109531	-0.738103	1.661071
D	0.518918	0.541574	-1.034584	0.841593	0.148018	-1.276802
E	1.081142	0.961920	-0.556315	0.290454	-0.715656	-0.328035

İstədiyiniz sütun və ya sətirə istənilən əməliyyatı da tətbiq edə bilərsiniz.

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

	Columns1	Columns2	Columns3	Columns4	Columns5	Columns6	Columns7
A	-0.828852	0.654459	-0.508973	-1.139647	-0.538566	0.740678	-0.334320
B	1.003648	-0.254032	-1.005248	-1.624998	-2.634857	1.122299	-5.960870
C	-1.929049	-0.051853	-0.738681	0.109531	-0.738103	1.661071	52.703553
D	0.518918	0.541574	-1.034584	0.841593	0.148018	-1.276802	1.822696

Məsələn, yuxarıda 'Column7' dəyərini yaratdıq və onu müxtəlif sütunların yaratdığı dəyərlə riyazi olaraq formalaşdıraraq, belə bir cədvəl yaradıldı.İndi "Column" əlavə edə bilirik, bəs necə silək?

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

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7
A	-0.828852	-0.508973	-1.139647	-0.538566	0.740678	-0.334320
B	1.003648	-1.005248	-1.624998	-2.634857	1.122299	-5.960870
C	-1.929049	-0.738681	0.109531	-0.738103	1.661071	52.703553
D	0.518918	-1.034584	0.841593	0.148018	-1.276802	1.822696
E	1.081142	-0.556315	0.290454	-0.715656	-0.328035	0.647001

'Axis' -in haqqında Numpy mövzusunda danışmışdıq. Qısaca desək, onun 'defolt' dəyəri 0-dır və '0' sətirləri '1' sütunlarını təmsil edir. Burada 'inplace' parametri vacibdir, o, əməliyyatın qalıcı olub olmadığını təyin edir və biz True dedikdə onu daimi olaraq təyin edir. Biz bir Column -ı indeks başlığına dəyər olaraq aşağıdakı kimi təyin edə bilərik:

df.set_index('Columns name', inplace = True)

Aşğıdakı üsulla 'index' və 'Column'ların adlarını öyrənə bilirik.

```
df.index.names
```

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

```
df.columns.names
```

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

İndi isə **loc** və **iloc** xüsusiyyətini işləyək:

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

```
Columns1    -1.929049
Columns3    -0.738681
Columns4     0.109531
Columns5    -0.738103
Columns6     1.661071
Columns7    52.703553
Name: C, dtype: float64
```

Yuxarıda verdiyimiz kodu istifadəsi nəticəsində o bizə, C sətiri üzrə Sütunlardakı dəyərləri verir.

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

```
Columns1   -0.828852
Columns3   -0.508973
Columns4   -1.139647
Columns5   -0.538566
Columns6    0.740678
Columns7   -0.334320
Name: A, dtype: float64
```

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

```
Columns1   -0.828852
Columns3   -0.508973
Columns4   -1.139647
Columns5   -0.538566
```

```
Columns6    0.740678
Columns7   -0.334320
Name: A, dtype: float64
```

İndi, bu iki kod arasındakı əlaqə haqqında 10 saniyə düşünməyə vaxt ayıraq. Onların hər ikisinin olduqca oxşar olduğunu görərik. 'iloc' xüsusiyyəti sətirin indeksinə uyğun olaraq 'Sütun' dəyərlərini çıxarır və bildiyiniz kimi, sətirlərin indeksləri 0-dan başlayır. 'A' indeksi 0-dır və 'iloc' funksiyasına '0'dəyərini təyin etmişik. Bu halda, bu iki dataframe xüsusiyyəti əslində eyni şeyi təmsil edir. 'loc' xüsusiyyətinsdə adı, 'iloc'da isə indeks nömrəsini göstərməliyik.

Bəzən daha da irəli getmək - sütun və sətir kəsişməsini tapmaq lazım gəlir. Məsələn, aşağıdakı kimi:

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

```
-0.5385656027534138
```

```
df
```

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7
A	-0.828852	-0.508973	-1.139647	-0.538566	0.740678	-0.334320
B	1.003648	-1.005248	-1.624998	-2.634857	1.122299	-5.960870
C	-1.929049	-0.738681	0.109531	-0.738103	1.661071	52.703553
D	0.518918	-1.034584	0.841593	0.148018	-1.276802	1.822696
E	1.081142	-0.556315	0.290454	-0.715656	-0.328035	0.647001

```
df.at['B','Columns5']
```

```
-2.6348569009460046
```

"at" xüsusiyyəti **Label - Başlıq** üzrə verilmiş sətir və sütun kəsişməsindəki vahid dəyəri verir.

```
df.iat[1,3]
```

```
-2.6348569009460046
```

"iat" xüsusiyyəti **Tam ədədi yeri - Index lokasiyası** üzrə verilmiş sətir və sütun kəsişməsindəki vahid dəyəri verir.

DataFrame Filtrləmə Əməliyyatları:

Adından da göründüyü kimi, biz əldə etdiyimiz dataların müəyyən aralıqlarda dəyərini tapmaq və ya həmin dataları əldə etmək istədikdə ondan istifadə edirik. İndi, yuxarıdakı dataya baxaq və sadə əməliyyatlarla başlayaq.

```
df
```

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7
A	-0.828852	-0.508973	-1.139647	-0.538566	0.740678	-0.334320
B	1.003648	-1.005248	-1.624998	-2.634857	1.122299	-5.960870
C	-1.929049	-0.738681	0.109531	-0.738103	1.661071	52.703553
D	0.518918	-1.034584	0.841593	0.148018	-1.276802	1.822696
E	1.081142	-0.556315	0.290454	-0.715656	-0.328035	0.647001

Kiçik əməliyyatlar ilə başlayaq:

```
df > 0.2
```

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7
A	False	False	False	False	True	False
B	True	False	False	False	True	False

```
booleanDF = df > 0
booleanDF
```

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7
A	False	False	False	False	True	False
B	True	False	False	False	True	False
C	False	False	True	False	True	True
D	True	False	True	True	False	True
E	True	False	True	False	False	True

'Boolean'ı aşağıdakı kimi izah edə bilərik: Əgər tələb olunan dəyər (burada '0'dan böyükdür mü?) şərti ödənirsə, "True", əks halda "False" qaytarır.

```
df[booleanDF]
```

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7
A	NaN	NaN	NaN	NaN	0.740678	NaN
B	1.003648	NaN	NaN	NaN	1.122299	NaN
C	NaN	NaN	0.109531	NaN	1.661071	52.703553
D	0.518918	NaN	0.841593	0.148018	NaN	1.822696
E	1.081142	NaN	0.290454	NaN	NaN	0.647001

Burada, 'NaN' dəyərləri yuxarıdakı BooleanDF dəyişənindəki 'False' dəyərləri təmsil edir.

```
df[df > 0.5]
```

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7
A	NaN	NaN	NaN	NaN	0.740678	NaN
B	1.003648	NaN	NaN	NaN	1.122299	NaN
C	NaN	NaN	NaN	NaN	1.661071	52.703553
D	0.518918	NaN	0.841593	NaN	NaN	1.822696
E	1.081142	NaN	NaN	NaN	NaN	0.647001

Burada 'df'-da '0.5'-dən böyük olan dəyərləri göstərir. 'False' olanlar bu halda da 'NaN' olaraq göstərilir. Bunu daha sonra daha yaxşı nümunələrlə izah edəcəyik.

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

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

Yuxarıda, müəyyən bir 'Column' dəyərinin filtrasiyasını görürük.

Aşağıdakı növbəti 4 nümunədə bu şərtin hər bir "Column" dəyərinə tətbiq edildiyini görə bilərsiniz. Bəzilərinin fərqli nəticələri var. Bəzilərində 4 sətir, bəzilərində isə 2 sətir var, sizcə niyə?

```
df[df['Columns3']> 0]
```

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

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7
C	-1.929049	-0.738681	0.109531	-0.738103	1.661071	52.703553
D	0.518918	-1.034584	0.841593	0.148018	-1.276802	1.822696
E	1.081142	-0.556315	0.290454	-0.715656	-0.328035	0.647001

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

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7
D	0.518918	-1.034584	0.841593	0.148018	-1.276802	1.822696

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

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7
B	1.003648	-1.005248	-1.624998	-2.634857	1.122299	-5.960870
D	0.518918	-1.034584	0.841593	0.148018	-1.276802	1.822696
E	1.081142	-0.556315	0.290454	-0.715656	-0.328035	0.647001

Qısaca ümumiləşdirsək, "True" olan dəyərlər göstərilir. Məsələn, 'Sütun4' 4 sıradan ibarətdir, onun elementlərindən biri olan E indeksli 5-ci sətir 'True'dir.

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

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7

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

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7

Yuxarıda müxtəlif filtrləmə əməliyyatları görürsünüz. Onlar da daha əvvəl qeyd etdiyimiz qaydalara tabedirlər. '&' işarəsi 'AND' operatoru, '|' işarəsi isə 'OR'operatoru ilə eyni mənada işlənir. Bu işarələrin yerinə 'AND' və ya 'OR' istifadə etsəniz, xəta alacaqsınız. '&' işarəsi 'SHIFT' və '7' düymələri ilə, '|' işarəsi isə 'ALT GR' və '-' işarələri ilə ekrana verilir.

```
df
```

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7
A	-0.828852	-0.508973	-1.139647	-0.538566	0.740678	-0.334320
B	1.003648	-1.005248	-1.624998	-2.634857	1.122299	-5.960870
C	-1.929049	-0.738681	0.109531	-0.738103	1.661071	52.703553
D	0.518918	-1.034584	0.841593	0.148018	-1.276802	1.822696
E	1.081142	-0.556315	0.290454	-0.715656	-0.328035	0.647001

```
df['Columns6'] = ['NewValue1','NewValue2','NewValue3','NewValue4','NewValue5']
df
```

	Columns1	Columns3	Columns4	Columns5	Columns6	Columns7
A	-0.828852	-0.508973	-1.139647	-0.538566	NewValue1	-0.334320
B	1.003648	-1.005248	-1.624998	-2.634857	NewValue2	-5.960870
C	-1.929049	-0.738681	0.109531	-0.738103	NewValue3	52.703553
D	0.518918	-1.034584	0.841593	0.148018	NewValue4	1.822696
E	1.081142	-0.556315	0.290454	-0.715656	NewValue5	0.647001

Gördüyünüz kimi verdiyimiz sütuna **string** dəyər təyin etmək mümkündür.

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

Columns6	Columns1	Columns3	Columns4	Columns5	Columns7
NewValue1	-0.828852	-0.508973	-1.139647	-0.538566	-0.334320
NewValue2	1.003648	-1.005248	-1.624998	-2.634857	-5.960870
NewValue3	-1.929049	-0.738681	0.109531	-0.738103	52.703553
NewValue4	0.518918	-1.034584	0.841593	0.148018	1.822696
NewValue5	1.081142	-0.556315	0.290454	-0.715656	0.647001

Burada, daimi olaraq indeks adını 'Sütun6' olaraq təyin etdik.

```
df.index.names
```

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

Burada isə indeks başlığını sorğulayırıq.

```
df.columns.names
```

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

Bu kodda eyni məntiqlə sütun adını sorğulayırıq və biz onu təyin etmədiyimiz üçün 'None' çıxır.

DataFrame -ləri Multi Indeks kimi təyin etmək:

Multi indeks, adından göründüyü kimi, indekslərin sayının çox olduğu və əsasən də, qruplaşdırıldığı hallarda istifadə olunur.

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

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

Biz 2 indeks yaratdıq və onları 'zip' funksiyası ilə birləşdirdik. Bunu etmək üçün list -dən istifadə etdik, lakin siz "Tuple" və "Dict"dən də istifadə edə bilərsiniz.

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

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

Həmin yuxarıdakı ziplə hazırlanmış indeksi 'hierarchy' ilə bərabərləşdirdik. Sonra 'pd.MultiIndex.from_tuples()' xassəsi ilə Multi indeks yaratdıq.

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

		Column1	Colum2	Column3
Group1	**Index1**	0.304639	1.401268	1.094050
	Index2	1.038416	0.738979	1.122265
	Index3	-0.740849	0.297538	-1.125289
Group2	**Index1**	-0.512095	1.249443	-0.597438
	Index2	-0.156956	-0.569734	-1.939492
	Index3	-0.614910	-1.511630	-2.013855
Group3	**Index1**	0.347343	0.873495	-1.036048
	Index2	-0.377797	0.045367	-0.478578
	Index3	-1.323564	1.804778	0.993988

'rand()' funksiyası ilə təsadüfi qiymətlər təyin etdik və Column datalarına onları verdik. Gördüyünüz kimi, biz 'Group' 'Index' 'Column' dəyərlərinə malik Multi-Index DataFrame əldə etdik.

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

```
Group1  Index1    0.304639
        Index2    1.038416
        Index3   -0.740849
Group2  Index1   -0.512095
        Index2   -0.156956
        Index3   -0.614910
Group3  Index1    0.347343
        Index2   -0.377797
        Index3   -1.323564
Name: Column1, dtype: float64
```

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

	Column1	Colum2	Column3
Index1	0.304639	1.401268	1.094050
Index2	1.038416	0.738979	1.122265
Index3	-0.740849	0.297538	-1.125289

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

		Column1	Colum2	Column3
Group1	**Index1**	0.304639	1.401268	1.094050
	Index2	1.038416	0.738979	1.122265
	Index3	-0.740849	0.297538	-1.125289
Group2	**Index1**	-0.512095	1.249443	-0.597438
	Index2	-0.156956	-0.569734	-1.939492
	Index3	-0.614910	-1.511630	-2.013855

Yuxarıda gördüyünüz kimi, datanın müəyyən bir hissəsini bu şəkildə çağıra bilərsiniz.

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

	Column1	Colum2	Column3
Index1	0.304639	1.401268	1.094050
Index2	1.038416	0.738979	1.122265
Index3	-0.740849	0.297538	-1.125289

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

		Column1	Colum2	Column3
Group1	**Index1**	0.304639	1.401268	1.094050
	Index2	1.038416	0.738979	1.122265
	Index3	-0.740849	0.297538	-1.125289
Group2	**Index1**	-0.512095	1.249443	-0.597438
	Index2	-0.156956	-0.569734	-1.939492
	Index3	-0.614910	-1.511630	-2.013855

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

```
Column1    0.304639
Colum2     1.401268
Column3    1.094050
Name: Index1, dtype: float64
```

'Qrup', 'Sütun' və 'İndeks' kimi dəyərlərə bu formada nəzər yetirə bilərsiniz. Siz həmçinin datalara nəzər yetirmək üçün çoxsaylı və müxtəlif növdə metodlar istifadə edə bilərsiniz.

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

```
0.30463940296837966
```

Dəqiq və spesifik dataları əldə etmək üçün işinizi yuxarıdakı kimi kodlarla yerinə yetirə bilərsiniz.

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

		Column1	Colum2	Column3
Groups	**Indexes**			
Group1	**Index1**	0.304639	1.401268	1.094050
	Index2	1.038416	0.738979	1.122265
	Index3	-0.740849	0.297538	-1.125289
Group2	**Index1**	-0.512095	1.249443	-0.597438
	Index2	-0.156956	-0.569734	-1.939492
	Index3	-0.614910	-1.511630	-2.013855
Group3	**Index1**	0.347343	0.873495	-1.036048
	Index2	-0.377797	0.045367	-0.478578
	Index3	-1.323564	1.804778	0.993988

Bu kodla Index adlarını öyrənərək eyni anda sütun adlarını da sorğu edə bilərik.

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

	Column1	Colum2	Column3
Indexes			
Index1	0.304639	1.401268	1.094050
Index2	1.038416	0.738979	1.122265
Index3	-0.740849	0.297538	-1.125289

xs funksiyası iloc və loc kimi funksiyaların işini yerinə yetirə bilər. Lakin onun bir üstün cəhəti də var:

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

```
Column1    0.304639
Colum2     1.401268
Column3    1.094050
Name: Index1, dtype: float64
```

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

```
0.30463940296837966
```

Birinci kodda əvvəlcə 'Group' dəyəri, sonra isə 'Index' dəyəri verilir. Çünki 'xs' funksiyası 'Group', 'Index' sırası ilə işləyir. Bu vəziyyət ikinci kodu da izah edir. 'Deməli, biz bunu hər dəfə belə edəcəyik? Bəs, Bunun üstünlüyü haradadır?

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

	Column1	Colum2	Column3
Groups			
Group1	0.304639	1.401268	1.094050
Group2	-0.512095	1.249443	-0.597438
Group3	0.347343	0.873495	-1.036048

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

```
Groups
Group1    0.304639
Group2   -0.512095
Group3    0.347343
Name: Column1, dtype: float64
```

Burada görünür ki, istədiyimiz səviyyəni verməklə onun alt səviyyəsinə baxış edə bilirik.

Xüsusi Təşəkkürlər: Batuhan Bayraktar
Yazar: Nuhbalayev Ramazan
Kod Mənbəsi:
A'dan Z'ye Pandas Tutoriali (Başlangıç ve Orta Seviye) və Kaggle: batuhan35