

## PYTHON'A GİRİŞ

- Jupyter Notebook'ta 'Hello, World!' ile Python alıştırmalarına başlanır.

In [1]: `print('Hello, World!')`

Hello, World!

- Python'da otomatik tamamlama özelliği vardır.

In [12]: `def func_with_keywords(abra=1, abbra=2, abbbra=3):  
 return abra, abbra, abbbra`

In [ ]: `func_with_keywords(ab)`

abbra=  
abbra=  
abra=  
abs

In [1]: `b = [1, 2, 3]`

In [5]: `b?`

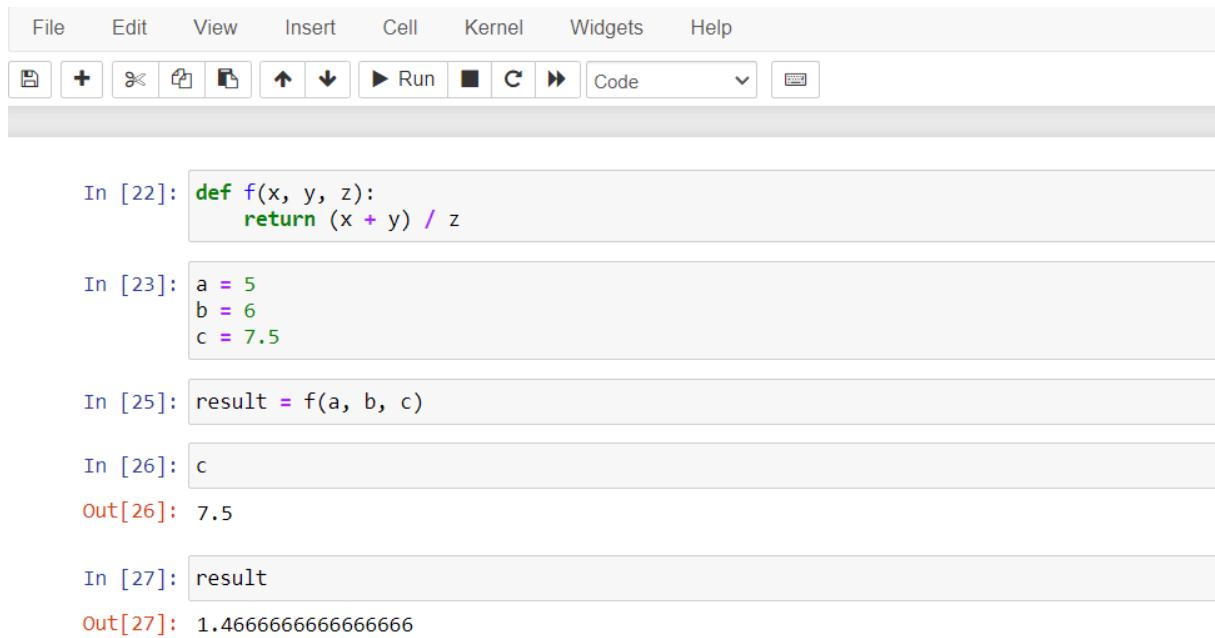
In [4]: `print?`

In [ ]:

Type: list  
String form: [1, 2, 3]  
Length: 3  
Docstring:  
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list.  
The argument must be an iterable if specified.

- Soru işaretü ilgili nesne hakkında genel bilgiler verir.



The screenshot shows a Jupyter Notebook interface with the following code execution history:

```

In [22]: def f(x, y, z):
           return (x + y) / z

In [23]: a = 5
           b = 6
           c = 7.5

In [25]: result = f(a, b, c)

In [26]: c
Out[26]: 7.5

In [27]: result
Out[27]: 1.4666666666666666
  
```

- Fonksiyon tanımlamaları

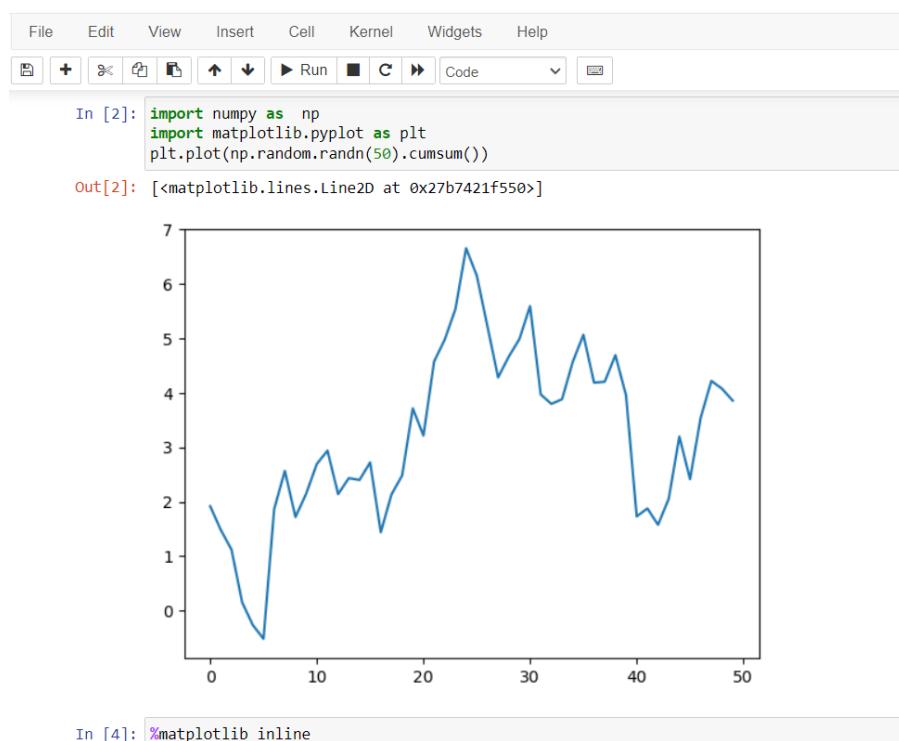
## Python'da Bazı Kısıyollar ve İşlevleri

Keyboard shortcut	Description
Ctrl-P or up-arrow	Search backward in command history for commands starting with currently entered text
Ctrl-N or down-arrow	Search forward in command history for commands starting with currently entered text
Ctrl-R	Readline-style reverse history search (partial matching)
Ctrl-Shift-V	Paste text from clipboard
Ctrl-C	Interrupt currently executing code
Ctrl-A	Move cursor to beginning of line
Ctrl-E	Move cursor to end of line
Ctrl-K	Delete text from cursor until end of line
Ctrl-U	Discard all text on current line
Ctrl-F	Move cursor forward one character
Ctrl-B	Move cursor back one character
Ctrl-L	Clear screen

## Python'da Bazı Magic Commandlar ve İşlevleri

Command	Description
%quickref	Display the IPython Quick Reference Card
%magic	Display detailed documentation for all of the available magic commands
%debug	Enter the interactive debugger at the bottom of the last exception traceback
%hist	Print command input (and optionally output) history
%pdb	Automatically enter debugger after any exception
%paste	Execute preformatted Python code from clipboard
%cpaste	Open a special prompt for manually pasting Python code to be executed
%reset	Delete all variables/names defined in interactive namespace
%page <i>OBJECT</i>	Pretty-print the object and display it through a pager
%run <i>script.py</i>	Run a Python script inside IPython
%prun <i>statement</i>	Execute <i>statement</i> with cProfile and report the profiler output
%time <i>statement</i>	Report the execution time of a single statement
%timeit <i>statement</i>	Run a statement multiple times to compute an ensemble average execution time; useful for timing code with very short execution time
%who, %who_ls, %whos	Display variables defined in interactive namespace, with varying levels of information/verbosity
%xdel <i>variable</i>	Delete a variable and attempt to clear any references to the object in the IPython internals

## Matplotlib Entegrasyonu



- Matplotlib inline ile görselleştirme örneği.
- # ile yorumlama yapılır.

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

```

File Edit View Insert Cell Kernel Widgets Help
[ ] + % < > ↑ ↓ ► Run ─ C ► Code

```

**In [16]:** a = 4.5

**In [20]:** b = 2

**In [21]:**

```

print('a is {0}, b is{1}'.format(type(a), type(b)))
a is <class 'float'>, b is <class 'int'>

```

cell In[21], line 2  
a is <class 'float'>, b is <class 'int'>  
^  
**SyntaxError:** invalid syntax

**In [22]:** a / b

**Out[22]:** 2.25

**In [15]:** type(a)

**Out[15]:** int

**In [16]:** a = 'foo'

**In [17]:** type(a)

**Out[17]:** str

**In [18]:** '5' + 5

-----  
**TypeError** Traceback (most recent call last)  
Cell In[18], line 1  
----> 1 '5' + 5  
  
**TypeError:** can only concatenate str (not "int") to str

- Değişkenler değer fonksiyonları ve dinamik referans tipleri.
- Python'da bazı özellikler ve metodlar.

```

In [2]: a.<Press Tab>
a.capitalize  a.format      a.isupper    a.rindex     a.strip
a.center       a.index       a.join       a.rjust      a.swapcase
a.count        a.isalnum   a.ljust      a.rpartition a.title
a.decode       a.isalpha    a.lower      a.rsplit     a.translate
a.encode       a.isdigit    a.lstrip     a.rstrip     a.upper
a.endswith    a.islower    a.partition  a.split      a.zfill
a.expandtabs  a.isspace   a.replace    a.splitlines a.startswith
a.find         a.istitle   a.rfind

```

```
File Edit View Insert Cell Kernel Widgets Help
[File, New, Open, Save, Run, Kernel, Stop, Help, Code, Terminal]

In [49]: 5 - 7
Out[49]: -2

In [50]: 5 <= 2
Out[50]: False

In [ ]: a = [1, 2, 3]
In [52]: b = a
In [53]: c = list(a)
In [54]: a is b
Out[54]: True
In [55]: a is not c
Out[55]: True
In [56]: a == c
Out[56]: False
In [57]: a = None
In [58]: a is None
Out[58]: True
```

- İkili operatör ve karşılaştırma örnekleri.
  - Operatör gösterimi ve sayısal değerler.

## Binary Operatörler

Operation	Description
a + b	Add a and b
a - b	Subtract b from a
a * b	Multiply a by b
a / b	Divide a by b
a // b	Floor-divide a by b, dropping any fractional remainder
a ** b	Raise a to the b power
a & b	True if both a and b are True; for integers, take the bitwise AND
a   b	True if either a or b is True; for integers, take the bitwise OR
a ^ b	For booleans, True if a or b is True, but not both; for integers, take the bitwise EXCLUSIVE-OR
a == b	True if a equals b
a != b	True if a is not equal to b
a <= b, a < b	True if a is less than (less than or equal) to b
a > b, a >= b	True if a is greater than (greater than or equal) to b
a is b	True if a and b reference the same Python object
a is not b	True if a and b reference different Python objects

## Standart Python Skaler Tipleri

Type	Description
<code>None</code>	The Python “null” value (only one instance of the <code>None</code> object exists)
<code>str</code>	String type; holds Unicode (UTF-8 encoded) strings
<code>bytes</code>	Raw ASCII bytes (or Unicode encoded as bytes)
<code>float</code>	Double-precision (64-bit) floating-point number (note there is no separate <code>double</code> type)
<code>bool</code>	A <code>True</code> or <code>False</code> value
<code>int</code>	Arbitrary precision signed integer

- Sayısal değerler ve string değerler ile örnek yapımı

The screenshot shows a Jupyter Notebook interface with the following code cells and their outputs:

```

File Edit View Insert Cell Kernel Widgets Help
[ ] + × ↻ ↺ ⌂ Run ⌂ C ⌂ ⌂ Code ⌂

In [60]: ival = 17239871
In [61]: ival ** 6
Out[61]: 26254519291092456596965462913230729701102721

In [64]: fval = 7.243
In [67]: fval2 = 6.78e-5
In [66]: 3 / 2
Out[66]: 1.5

In [68]: 3 // 2
Out[68]: 1

In [69]: a = 'one way of writting a string'
In [70]: b = 'another way'
In [74]: print(a, b)
one way of writting a string another way

In [73]: print(a, b, c)
one way of writting a string another way
  
```

The screenshot shows a Jupyter Notebook interface with the following code cells and their outputs, highlighting a common mistake in string manipulation:

```

File Edit View Insert Cell Kernel Widgets Help
[ ] + × ↻ ↺ ⌂ Run ⌂ C ⌂ ⌂ Code ⌂

In [79]: a = 'this is a string'
In [80]: a[10] = 'f'

-----
TypeError                                 Traceback (most recent call last)
Cell In[80], line 1
----> 1 a[10] = 'f'

TypeError: 'str' object does not support item assignment

In [81]: b = a.replace('string', 'longer string')
In [82]: b
Out[82]: 'this is a longer string'

In [83]: a
Out[83]: 'this is a string'
  
```

The screenshot shows a Jupyter Notebook interface with the following code execution history:

```
In [84]: a = 5.6
In [85]: s = str(a)
In [86]: print(s)
5.6
In [87]: s = 'python'
In [88]: list(s)
Out[88]: ['p', 'y', 't', 'h', 'o', 'n']
In [89]: ['p', 'y', 't', 'h', 'o', 'n']
Out[89]: ['p', 'y', 't', 'h', 'o', 'n']
In [90]: s[:3]
Out[90]: 'pyt'
In [91]: 'pyt'
Out[91]: 'pyt'
```

The screenshot shows a Jupyter Notebook interface with the following code execution history:

```
In [92]: s = r'this\has\no\special\characters'
In [93]: s
Out[93]: 'this\\has\\\\no\\\\special\\\\characters'
```

- Bayt ve evrensel kod gösterimi ile ilgili örnekler

The screenshot shows a Jupyter Notebook interface with the following code execution history:

```

In [94]: a = 'this is the first half'
In [95]: b = 'and this is the second half'
In [96]: a + b
Out[96]: 'this is the first halfand this is the second half'

In [97]: template = '{0:.2f} {1:s} are worth US${2:d}'
In [98]: template.format(4.5560, 'Argentine Pesos', 1)
Out[98]: '4.56 Argentine Pesos are worth US$1'

```

The screenshot shows a Jupyter Notebook interface with the following code execution history, demonstrating encoding and decoding operations:

```

In [2]: val = "espanol"
In [3]: val
Out[3]: 'espanol'

In [4]: val_utf8 = val.encode('utf-8')
In [5]: val_utf8
Out[5]: b'espanol'

In [6]: type(val_utf8)
Out[6]: bytes

In [8]: val_utf8.decode('utf-8')
Out[8]: 'espanol'

In [9]: val.encode('latin1')
Out[9]: b'espanol'

In [10]: val.encode('utf-16')
Out[10]: b'\xff\xfe\x00\x00p\x00a\x00n\x00o\x001\x00'

In [11]: val.encode('utf-16le')
Out[11]: b'e\x00s\x00p\x00a\x00n\x00o\x001\x00'

```

- o Boolean ifadelere örnek

```
In [16]: True and True
Out[16]: True

In [17]: False or True
Out[17]: True

In [18]: True or False
Out[18]: True
```

- o Tip dönüşümleri ile ilgili örnekler

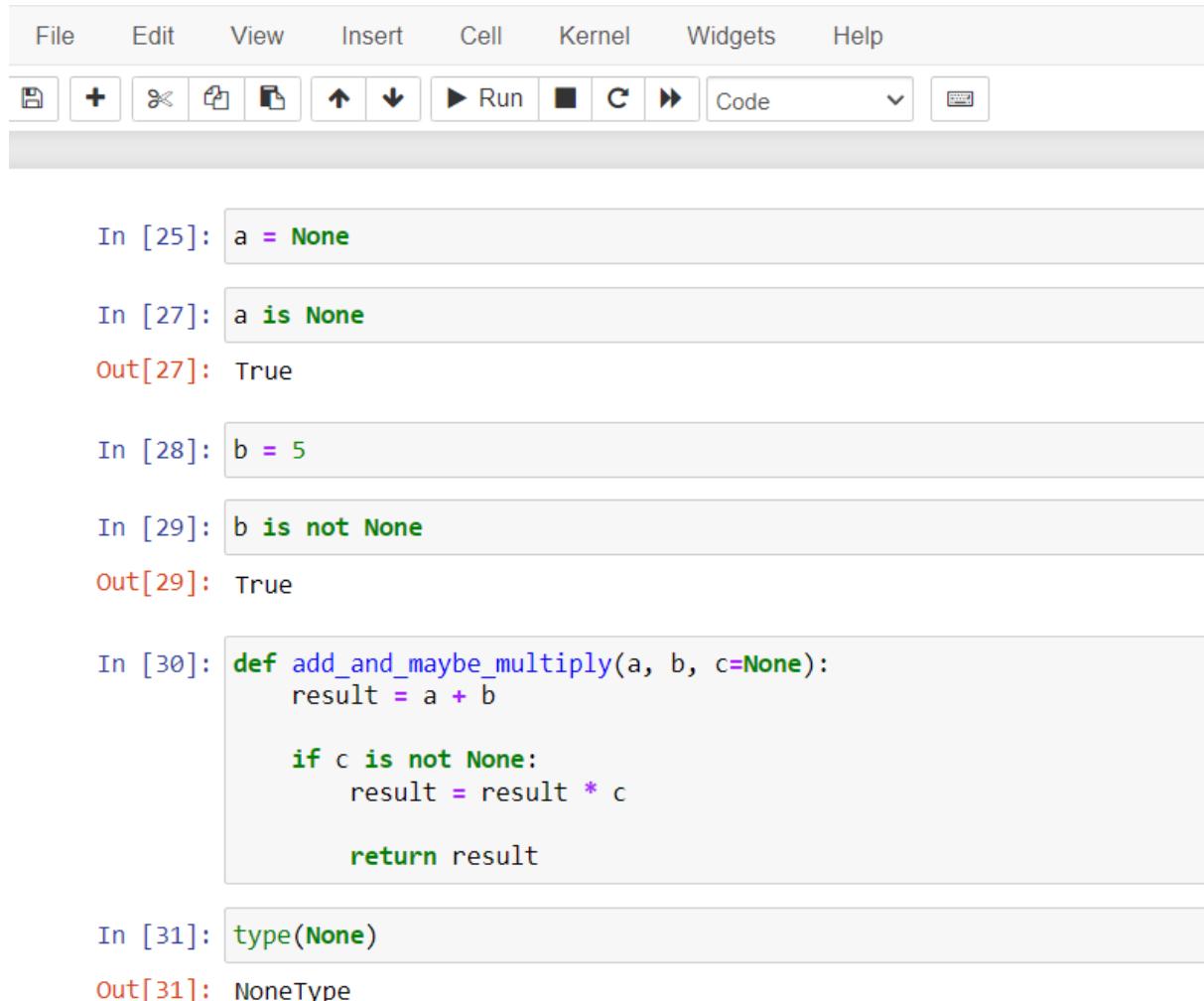
```
In [19]: s = '3.14159'
In [20]: fval = float(s)
In [21]: type(fval)
Out[21]: float

In [22]: int(fval)
Out[22]: 3

In [23]: bool(fval)
Out[23]: True

In [24]: bool(0)
Out[24]: False
```

- None tipi Python'da null değerler için kullanılır. Aşağıda bunlar ile ilgili örnekler bulunur.



The screenshot shows a Jupyter Notebook interface with the following session history:

```

File Edit View Insert Cell Kernel Widgets Help
[File, New, Open, Save, Run, Stop, Kernel, Help, etc.] [Code] [Cell Type]

```

In [25]: `a = None`

In [27]: `a is None`

Out[27]: True

In [28]: `b = 5`

In [29]: `b is not None`

Out[29]: True

In [30]: `def add_and_maybe_multiply(a, b, c=None):
 result = a + b

 if c is not None:
 result = result * c

 return result`

In [31]: `type(None)`

Out[31]: `NoneType`

## Datetime Biçimi Tanımlama

Type	Description
%Y	Four-digit year
%y	Two-digit year
%m	Two-digit month [01, 12]
%d	Two-digit day [01, 31]
%H	Hour (24-hour clock) [00, 23]
%I	Hour (12-hour clock) [01, 12]
%M	Two-digit minute [00, 59]
%S	Second [00, 61] (seconds 60, 61 account for leap seconds)
%w	Weekday as integer [0 (Sunday), 6]
%U	Week number of the year [00, 53]; Sunday is considered the first day of the week, and days before the first Sunday of the year are “week 0”
%W	Week number of the year [00, 53]; Monday is considered the first day of the week, and days before the first Monday of the year are “week 0”
%z	UTC time zone offset as +HHMM or -HHMM; empty if time zone naive
%F	Shortcut for %Y-%m-%d (e.g., 2012-4-18)
%D	Shortcut for %m/%d/%y (e.g., 04/18/12)

- Tarih ve zaman durumlarında kullanılan kısaltmalar işlevleri ve örnekleri

The screenshot shows a Jupyter Notebook interface with a toolbar at the top. Below the toolbar, there are several code cells. The first cell (In [32]) imports the datetime module. The second cell (In [42]) creates a datetime object for October 29, 2011, at 20:30:21. The third cell (In [34]) prints the day of the month (29). The fourth cell (Out[34]) shows the result as 29. The fifth cell (In [35]) prints the minute of the hour (30). The sixth cell (Out[35]) shows the result as 30. The seventh cell (In [36]) prints the date part of the datetime object. The eighth cell (Out[36]) shows the result as datetime.date(2011, 10, 29). The ninth cell (In [37]) prints the time part of the datetime object. The tenth cell (Out[37]) shows the result as datetime.time(20, 30, 21). The eleventh cell (In [44]) prints the string representation of the datetime object using strftime('%m/%d/%Y %H:%M'). The twelfth cell (Out[44]) shows the result as '10/29/2011 20:30'. The thirteenth cell (In [45]) replaces the minute and second fields with 0. The fourteenth cell (Out[45]) shows the result as datetime.datetime(2011, 10, 29, 20, 0).

```

File Edit View Insert Cell Kernel Widgets Help
[ ] + X ↻ ↺ Run ▶ Code ▾

In [32]: from datetime import datetime, date, time
In [42]: dt = datetime(2011, 10, 29, 20, 30, 21)
In [34]: dt.day
Out[34]: 29
In [35]: dt.minute
Out[35]: 30
In [36]: dt.date()
Out[36]: datetime.date(2011, 10, 29)
In [37]: dt.time()
Out[37]: datetime.time(20, 30, 21)
In [44]: dt.strftime('%m/%d/%Y %H:%M')
Out[44]: '10/29/2011 20:30'
In [45]: dt.replace(minute=0, second=0)
Out[45]: datetime.datetime(2011, 10, 29, 20, 0)

```

The screenshot shows a Jupyter Notebook interface with a toolbar at the top. Below the toolbar, there are several code cells. The first cell (In [46]) creates a datetime object for November 15, 2011, at 22:30. The second cell (In [47]) creates a timedelta object by subtracting the original datetime object from this new one. The third cell (In [48]) prints the timedelta object. The fourth cell (Out[48]) shows the result as datetime.timedelta(days=17, seconds=7179). The fifth cell (In [49]) prints the type of the timedelta object. The sixth cell (Out[49]) shows the result as datetime.timedelta. The seventh cell (In [50]) prints the original datetime object. The eighth cell (Out[50]) shows the result as datetime.datetime(2011, 10, 29, 20, 30, 21). The ninth cell (In [51]) adds the timedelta object to the original datetime object. The tenth cell (Out[51]) shows the result as datetime.datetime(2011, 11, 15, 22, 30).

```

File Edit View Insert Cell Kernel Widgets Help
[ ] + X ↻ ↺ Run ▶ Code ▾

In [46]: dt2 = datetime(2011, 11, 15, 22, 30)
In [47]: delta = dt2 - dt
In [48]: delta
Out[48]: datetime.timedelta(days=17, seconds=7179)
In [49]: type(delta)
Out[49]: datetime.timedelta
In [50]: dt
Out[50]: datetime.datetime(2011, 10, 29, 20, 30, 21)
In [51]: dt + delta
Out[51]: datetime.datetime(2011, 11, 15, 22, 30)

```

## CONTROL AKIŞLARI

### if, Elif ve Else

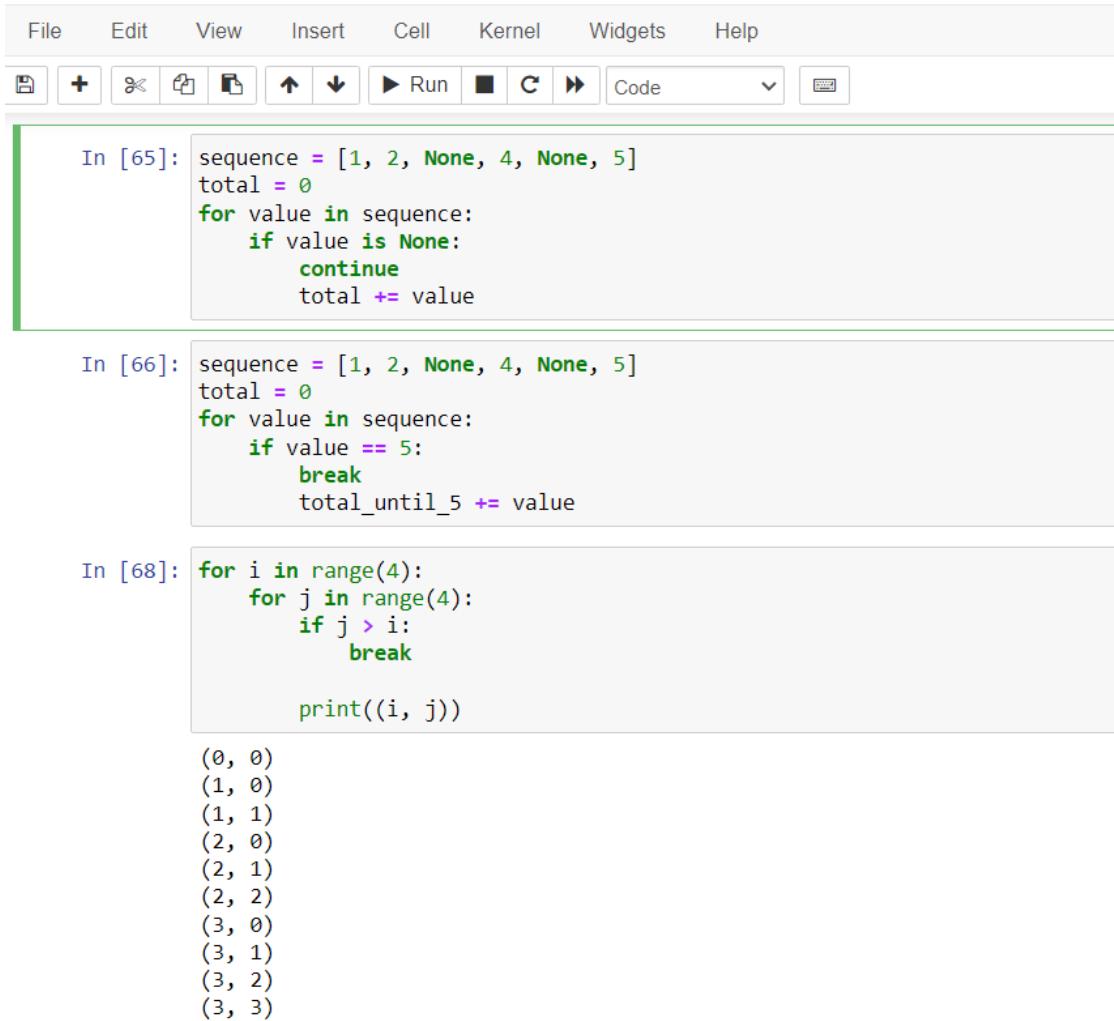
- o if, elif ve else kontrol akışlarına örnek

The screenshot shows a Jupyter Notebook interface with the following code examples:

```
In [ ]: if x < 0:  
         print('Its negative')  
  
In [ ]: elif x==0:  
         print('Equal to zero')  
  
In [ ]: elif 0 > x > 5:  
         print('Positive but smaller than 5')  
  
In [ ]: else:  
         print('Positive and larger than or equal to 5')  
  
In [59]: a = 5; b=7  
  
In [60]: c = 8; d = 4  
  
In [63]: if a < b or c > d:  
         print('Made it')  
     Made it  
  
In [64]: 4 > 3 > 2 > 1  
Out[64]: True
```

## For Döngüleri

- For döngülerine örnek



```
File Edit View Insert Cell Kernel Widgets Help  
In [65]: sequence = [1, 2, None, 4, None, 5]
total = 0
for value in sequence:
    if value is None:
        continue
    total += value

In [66]: sequence = [1, 2, None, 4, None, 5]
total = 0
for value in sequence:
    if value == 5:
        break
    total_until_5 += value

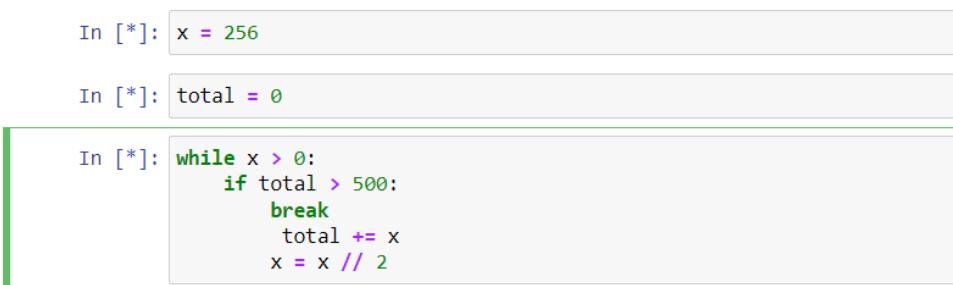
In [68]: for i in range(4):
            for j in range(4):
                if j > i:
                    break

                print((i, j))

(0, 0)
(1, 0)
(1, 1)
(2, 0)
(2, 1)
(2, 2)
(3, 0)
(3, 1)
(3, 2)
(3, 3)
```

## While Döngüsü

- While döngüsüne örnek

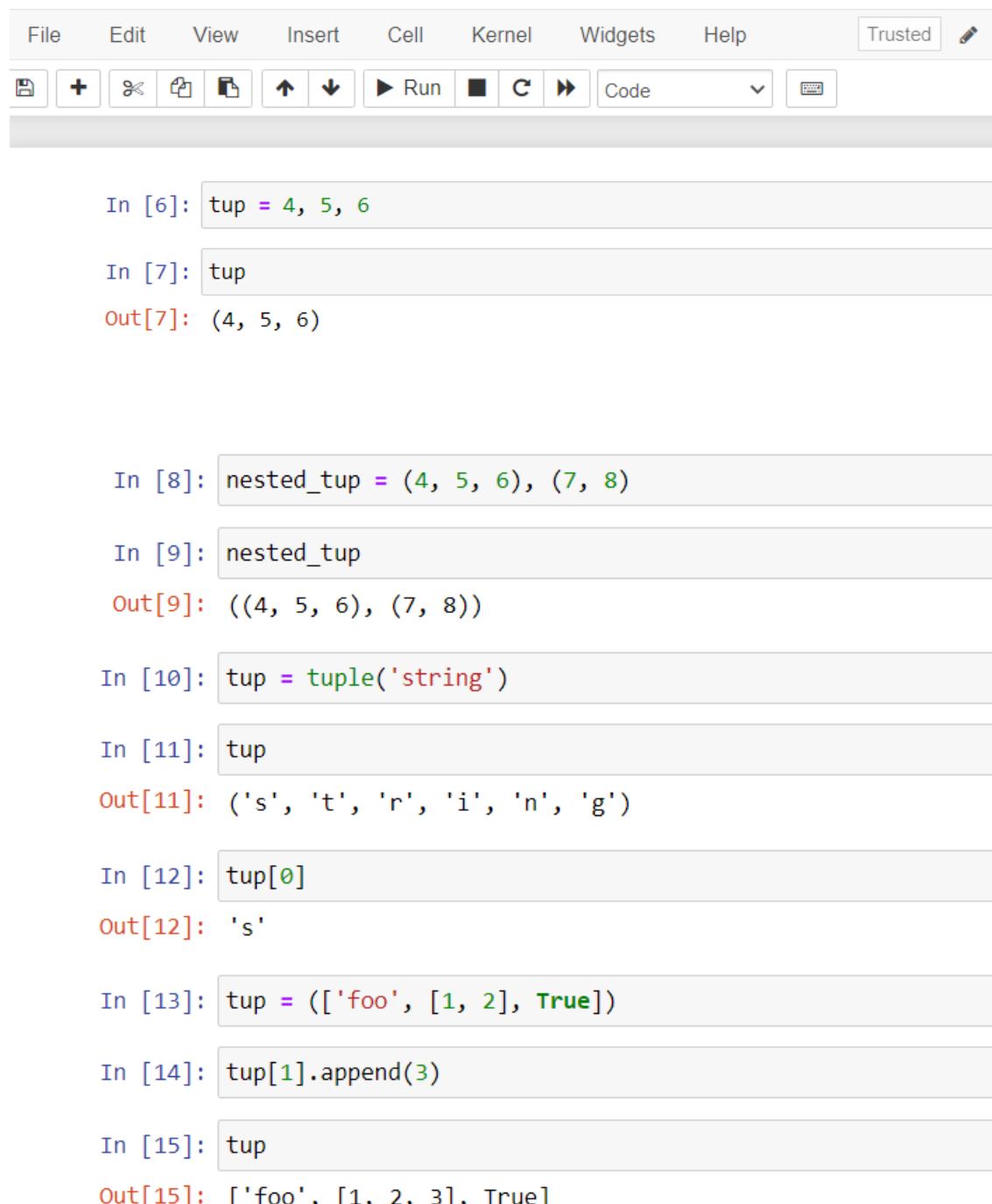


```
In [*]: x = 256
In [*]: total = 0
In [*]: while x > 0:
            if total > 500:
                break
            total += x
            x = x // 2
```

## Yerleşik Veri Yapıları, Fonksiyonlar ve Dosyalar

### Tuple

- Tuple, Python nesnelerinin sabit uzunlukta değişmez bir dizisidir.
- Tupleler ile ilgili birkaç örnek. İki Tuple arasında toplama çıkarma ve çarpma gibi işlemler de yapılabilir.



The screenshot shows a Jupyter Notebook interface with the following code execution history:

```
In [6]: tup = 4, 5, 6
In [7]: tup
Out[7]: (4, 5, 6)

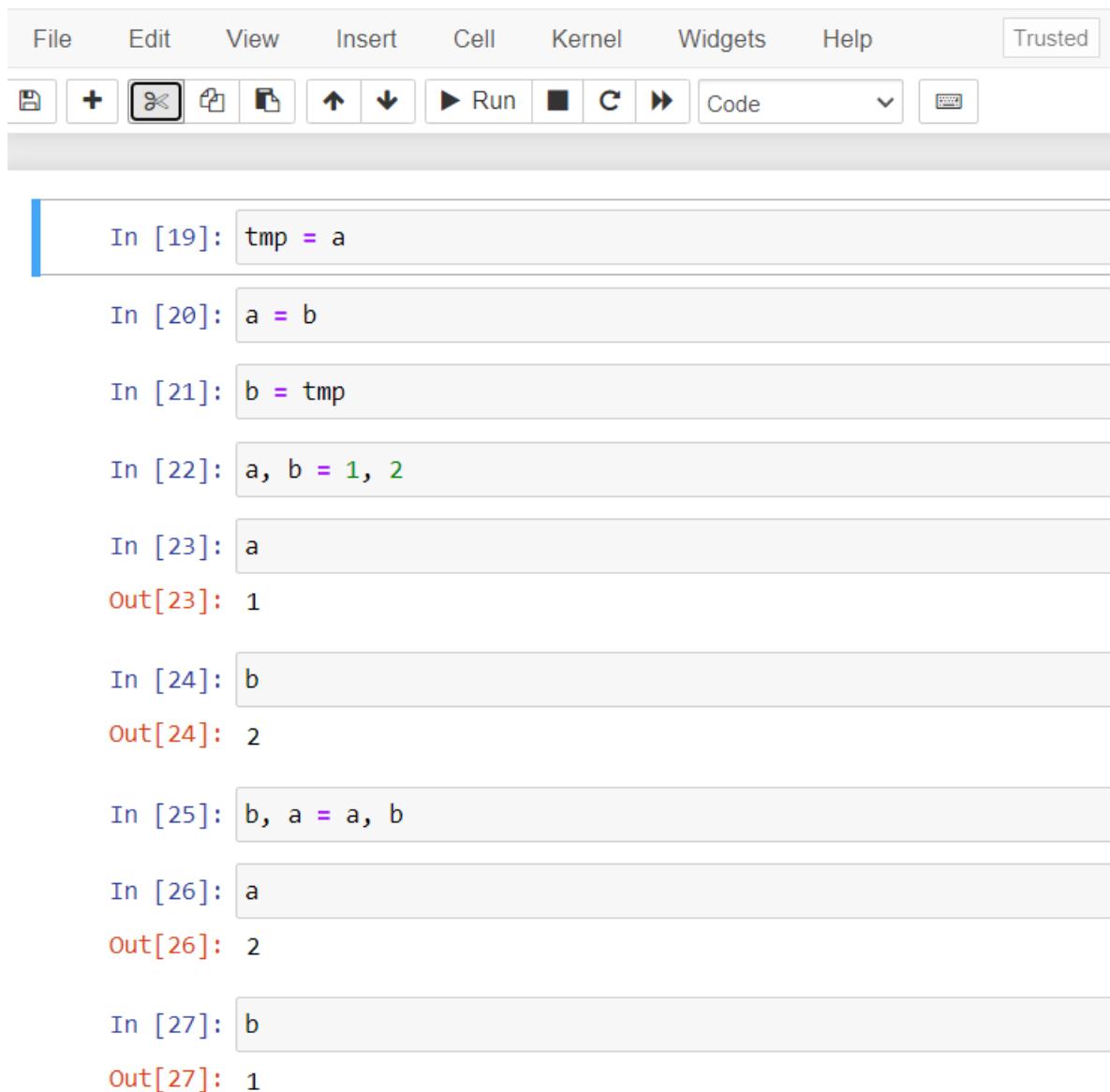
In [8]: nested_tup = (4, 5, 6), (7, 8)
In [9]: nested_tup
Out[9]: ((4, 5, 6), (7, 8))

In [10]: tup = tuple('string')
In [11]: tup
Out[11]: ('s', 't', 'r', 'i', 'n', 'g')

In [12]: tup[0]
Out[12]: 's'

In [13]: tup = ([ 'foo', [ 1, 2 ], True ])
In [14]: tup[1].append(3)
In [15]: tup
Out[15]: ['foo', [ 1, 2, 3 ], True ]
```

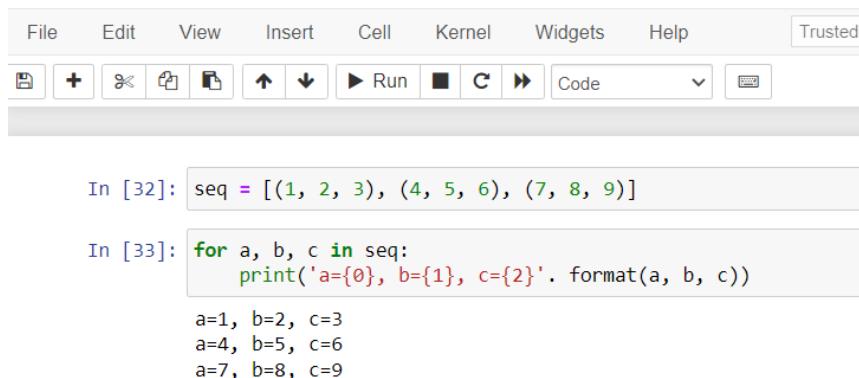
## Tuplelar'ı Açmak



The screenshot shows a Jupyter Notebook interface with the following code execution history:

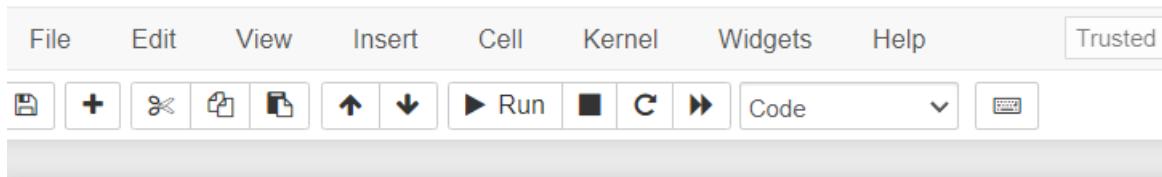
- In [19]: `tmp = a`
- In [20]: `a = b`
- In [21]: `b = tmp`
- In [22]: `a, b = 1, 2`
- In [23]: `a`  
Out[23]: `1`
- In [24]: `b`  
Out[24]: `2`
- In [25]: `b, a = a, b`
- In [26]: `a`  
Out[26]: `2`
- In [27]: `b`  
Out[27]: `1`

- Birçok dilde değişim 'tmp' ile sağlanır fakat Python'da çok daha basit şekilde işlem halledilir.



The screenshot shows a Jupyter Notebook interface with the following code execution history:

- In [32]: `seq = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]`
- In [33]: `for a, b, c in seq:  
 print('a={0}, b={1}, c={2}'.format(a, b, c))`  
a=1, b=2, c=3  
a=4, b=5, c=6  
a=7, b=8, c=9



The screenshot shows the top navigation bar of a Jupyter Notebook with options like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A "Trusted" button is also present. Below the menu is a toolbar with various icons for file operations (Save, New, Open, etc.) and cell execution (Run, Kernel Stop, Kernel Restart). A dropdown menu labeled "Code" is open.

```
In [28]: values = 1, 2, 3, 4, 5
```

```
In [29]: a, b, *rest = values
```

```
In [30]: a, b
```

```
Out[30]: (1, 2)
```

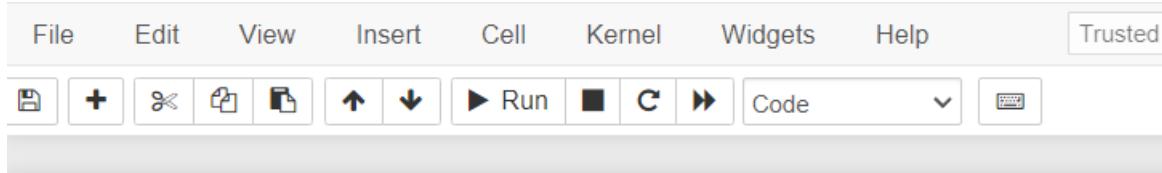
```
In [31]: rest
```

```
Out[31]: [3, 4, 5]
```

- \*rest ile diziyi istenilen yerden bölerek istenilen elde edilir.

## List

- Listeler Tuplelar'a göre daha değişken uzunluktadır ve içerikleri yerinde değiştirilebilir.



The screenshot shows the top navigation bar of a Jupyter Notebook with options like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A "Trusted" button is also present. Below the menu is a toolbar with various icons for file operations (Save, New, Open, etc.) and cell execution (Run, Kernel Stop, Kernel Restart). A dropdown menu labeled "Code" is open.

```
In [38]: a_list = [2, 3, 7, None]
```

```
In [39]: tup = ('foo', 'bar', 'baz')
```

```
In [40]: b_list = list(tup)
```

```
In [41]: b_list
```

```
Out[41]: ['foo', 'bar', 'baz']
```

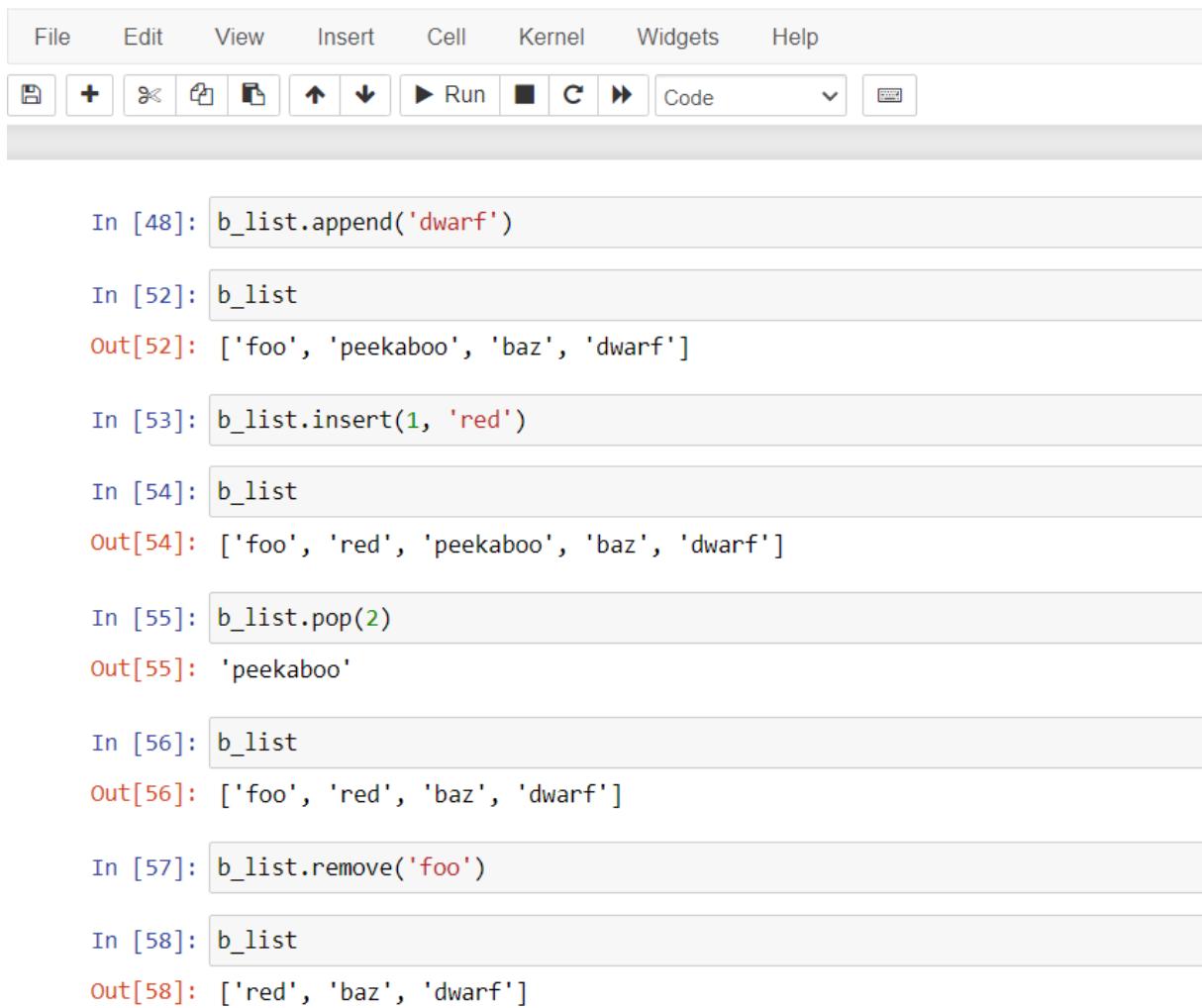
```
In [42]: b_list[1] = 'peekaboo'
```

```
In [43]: b_list
```

```
Out[43]: ['foo', 'peekaboo', 'baz']
```

## Eleman Ekleme ve Silme

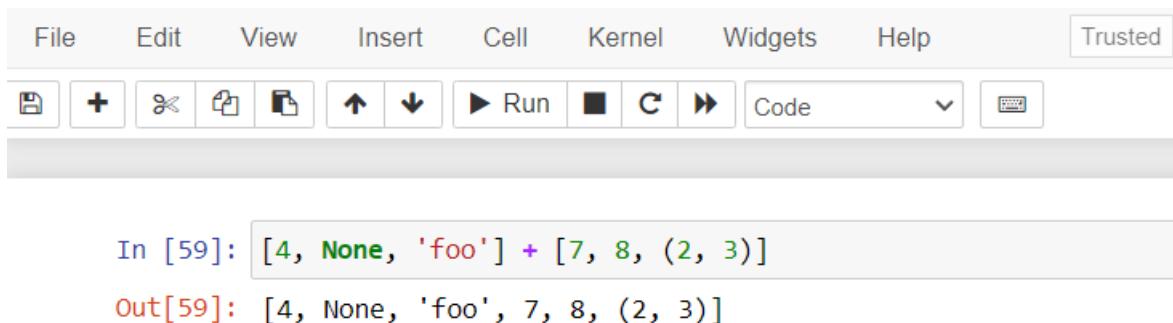
- Append ve Insert fonksiyonuyla eleman eklenir. Remove ile silinir. Pop ile eleman silinip döndürülür.



The screenshot shows a Jupyter Notebook interface with the following code execution history:

```
In [48]: b_list.append('dwarf')
In [52]: b_list
Out[52]: ['foo', 'peekaboo', 'baz', 'dwarf']
In [53]: b_list.insert(1, 'red')
In [54]: b_list
Out[54]: ['foo', 'red', 'peekaboo', 'baz', 'dwarf']
In [55]: b_list.pop(2)
Out[55]: 'peekaboo'
In [56]: b_list
Out[56]: ['foo', 'red', 'baz', 'dwarf']
In [57]: b_list.remove('foo')
In [58]: b_list
Out[58]: ['red', 'baz', 'dwarf']
```

## Listeleri Birleştirme



The screenshot shows a Jupyter Notebook interface with the following code execution history:

```
In [59]: [4, None, 'foo'] + [7, 8, (2, 3)]
Out[59]: [4, None, 'foo', 7, 8, (2, 3)]
```

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

```
In [61]: x = [4, None, 'foo']
In [62]: x.extend([7, 8, (2,3)])
In [63]: x
Out[63]: [4, None, 'foo', 7, 8, (2, 3)]
```

- Bir listeye birden fazla eleman ekleme işlemi için “extend” fonksiyonu kullanılır.

## Sorting

- Listedeki elemanları sıralama işlemidir.

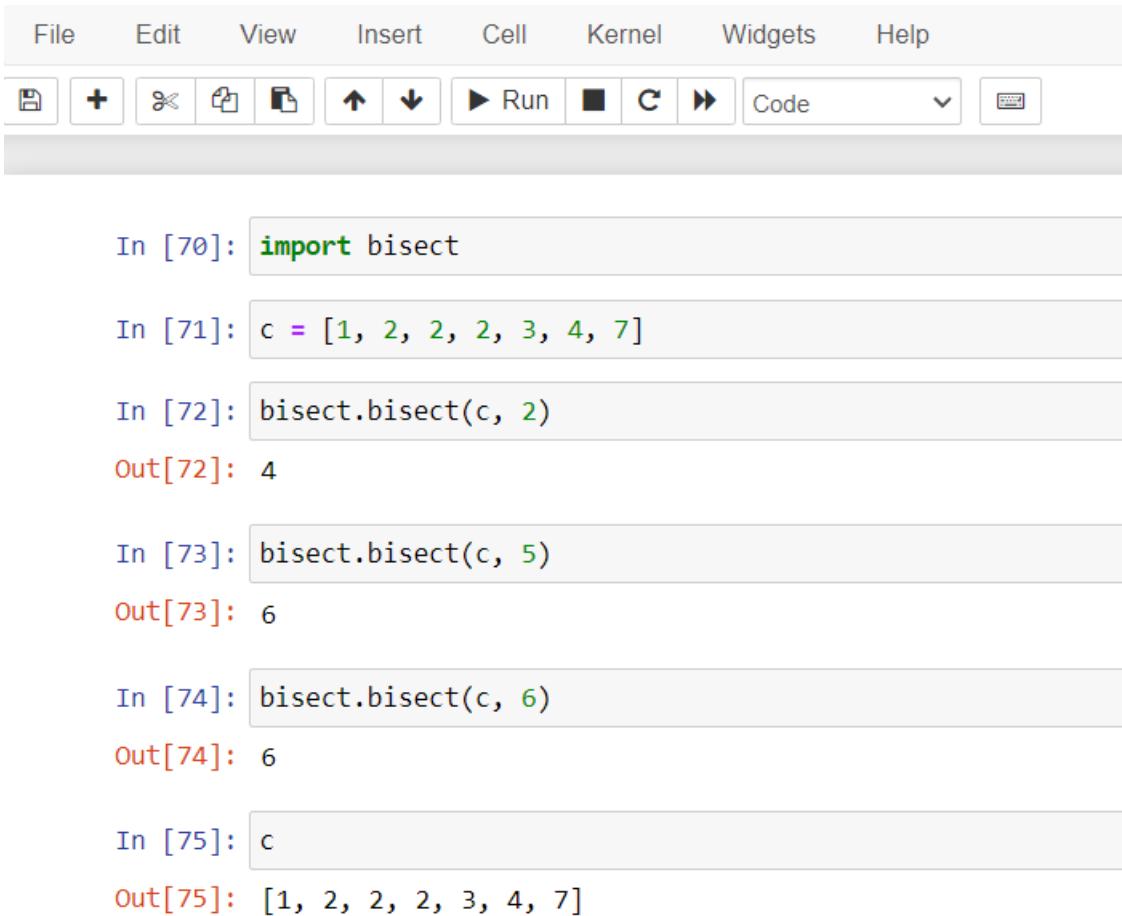
The screenshot shows a Jupyter Notebook interface with the following content:

```
In [64]: a = [7, 2, 5, 1, 3]
In [65]: a.sort()
In [66]: a
Out[66]: [1, 2, 3, 5, 7]

In [67]: b = ['saw', 'small', 'He', 'foxes', 'six']
In [68]: b.sort(key=len)
In [69]: b
Out[69]: ['He', 'saw', 'six', 'small', 'foxes']
```

## İkili Arama ve Sıralanmış Listenin Bakımı

- Yerleşik “bisect” modülü ikili aramayı uygular ve ikiye bölünürken konumunu buldurur.

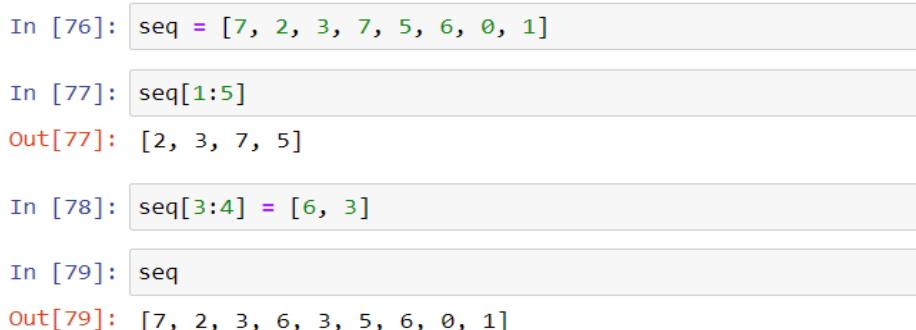


The screenshot shows a Jupyter Notebook interface with a toolbar at the top and several code cells below. The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help, along with various icons for file operations and cell execution. The code cells show the following interactions:

- In [70]: `import bisect`
- In [71]: `c = [1, 2, 2, 2, 3, 4, 7]`
- In [72]: `bisect.bisect(c, 2)`  
Out[72]: 4
- In [73]: `bisect.bisect(c, 5)`  
Out[73]: 6
- In [74]: `bisect.bisect(c, 6)`  
Out[74]: 6
- In [75]: `c`  
Out[75]: [1, 2, 2, 2, 3, 4, 7]

## Slicing

- [start:stop] biçiminde istenilen bolümden bölüme kadar dilimleme yapılır.



The screenshot shows a Jupyter Notebook interface with several code cells demonstrating slicing:

- In [76]: `seq = [7, 2, 3, 7, 5, 6, 0, 1]`
- In [77]: `seq[1:5]`  
Out[77]: [2, 3, 7, 5]
- In [78]: `seq[3:4] = [6, 3]`
- In [79]: `seq`  
Out[79]: [7, 2, 3, 6, 3, 5, 6, 0, 1]

```

File Edit View Insert Cell Kernel Widgets Help
[+] Run Cell Widgets Help
In [80]: seq[:5]
Out[80]: [7, 2, 3, 6, 3]

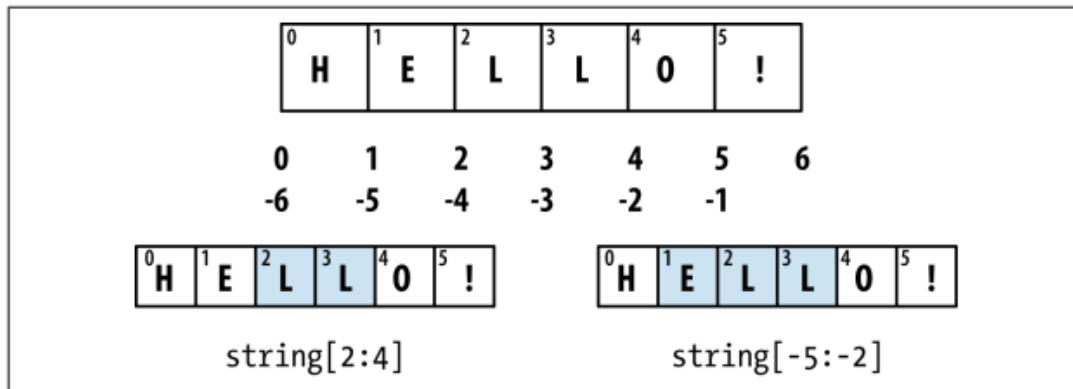
In [81]: seq[-6:-2]
Out[81]: [6, 3, 5, 6]

In [82]: seq[::-1]
Out[82]: [1, 0, 6, 5, 3, 6, 3, 2, 7]

In [83]: seq[::2]
Out[83]: [7, 3, 3, 6, 1]

```

- Dilimleme kurallarının gösterimi.



## Enumerate

```

File Edit View Insert Cell Kernel Widgets Help Trusted
[+] Run Cell Widgets Help
In [ ]: i = 0
         for value in collection:
                  i += 1

In [ ]: for i, value in enumerate(collection):

```

The screenshot shows a Jupyter Notebook interface with a toolbar at the top containing File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and a Trusted button. Below the toolbar are standard notebook icons for file operations, cell selection, and run. The main area displays the following code and its output:

```
In [84]: some_list = ['foo', 'bar', 'baz']
In [85]: mapping = {}
In [86]: for i, v in enumerate(some_list):
    mapping[v] = i
In [87]: mapping
Out[87]: {'foo': 0, 'bar': 1, 'baz': 2}
```

- Enum tipleri Python'da numaralandırmayı kullanan yararlı bir model, bir sıranın değerlerini (benzersiz olduğu varsayılan) sıradaki konumlarıyla eşleyen bir hesaplamadır.

## Sorted

- Listelerdeki sort yöntemiyle aynı işlevdedir.

The screenshot shows a Jupyter Notebook interface with a toolbar at the top containing File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and a Trusted button. Below the toolbar are standard notebook icons for file operations, cell selection, and run. The main area displays the following code and its output:

```
In [88]: sorted([7, 1, 2, 6, 0, 3, 2])
Out[88]: [0, 1, 2, 2, 3, 6, 7]

In [89]: sorted('horse race')
Out[89]: [' ', 'a', 'c', 'e', 'e', 'h', 'o', 'r', 'r', 's']
```

## Zip

- Dizilerde elemanları birleştirme olarak kullanılır. Satır ve sütun olarak birleştirir.

The screenshot shows a Jupyter Notebook interface with the following code execution history:

- In [97]: seq1 = ['foo', 'bar', 'baz']
- In [ ]: seq1 = ['one', 'two', 'three']
- In [ ]: zipped = zip(seq1, seq2)
- In [98]: list(zipped)  
Out[98]: [('one', 'one'), ('two', 'two'), ('three', 'three')]
- In [99]: pitchers = [('Nolan', 'Ryan'), ('Roger', 'Clemens'), ('Schilling', 'Curt')]
- In [100]: first\_names, last\_names = zip(\*pitchers)
- In [101]: first\_names  
Out[101]: ('Nolan', 'Roger', 'Schilling')
- In [102]: last\_names  
Out[102]: ('Ryan', 'Clemens', 'Curt')

## Dict

The screenshot shows a Jupyter Notebook interface with the following code execution history:

- In [103]: empty\_dict = {}
- In [104]: d1 = {'a': 'some value', 'b': [1, 2, 3, 4]}
- In [105]: d1  
Out[105]: {'a': 'some value', 'b': [1, 2, 3, 4]}
- In [106]: d1[7] = 'an integer'
- In [107]: d1  
Out[107]: {'a': 'some value', 'b': [1, 2, 3, 4], 7: 'an integer'}
- In [108]: d1['b']  
Out[108]: [1, 2, 3, 4]
- In [109]: 'b' in d1  
Out[109]: True

```
In [110]: d1['dummy'] = 'another value'

In [111]: d1

Out[111]: {'a': 'some value',
            'b': [1, 2, 3, 4],
            7: 'an integer',
            'dummy': 'another value'}
```

```
In [112]: ret = d1.pop('dummy')

In [113]: ret

Out[113]: 'another value'

In [114]: d1

Out[114]: {'a': 'some value', 'b': [1, 2, 3, 4], 7: 'an integer'}
```

```
In [115]: list(d1.keys())

Out[115]: ['a', 'b', 7]

In [116]: list(d1.values())

Out[116]: ['some value', [1, 2, 3, 4], 'an integer']

In [117]: d1.update({'b' : 'foo', 'c' : 12})
```

```
In [118]: d1

Out[118]: {'a': 'some value', 'b': 'foo', 7: 'an integer', 'c': 12}
```

- Dict Python'da en önemli yerleşik veri yapısıdır. Değerleri ayırabilir, birlestirebilir, erişebilir, ekleyebilir ve silebilir.

## Dizilerden Dictler Oluşturma

The screenshot shows a Jupyter Notebook interface with a toolbar at the top. The code cell contains the following Python code:

```
In [ ]: mapping = {}
for key, value in zip(key_list, value_list):
    mapping[key] = value

In [120]: mapping = dict(zip(range(5), reversed(range(5)))) 

In [121]: mapping
Out[121]: {0: 4, 1: 3, 2: 2, 3: 1, 4: 0}
```

## Default Değerler

The screenshot shows a Jupyter Notebook interface with a toolbar at the top. The code cell contains the following Python code:

```
In [129]: words = ['apple', 'bat', 'bar', 'atom', 'book']

In [ ]: by_letter() = {}

In [ ]: for word in words:
    letter = word[0]
    if letter not in by_letter:
        by_letter[letter] = [word]
    else:
        by_letter[letter].append[word]

In [ ]: by_letter
```

## Geçerli Dict Anahtar Türleri

- Anahtarlar genellikle skaler türlerdir (int, float, string) veya tuplelar gibi kullanılabilir nesnelere sahiptir. Buradaki teknik terim hashability'dir.
- Son yapılan örnek başarısızdır çünkü listeler değiştirilebilir.

The screenshot shows a Jupyter Notebook interface with the following session history:

```
In [132]: hash('string')
Out[132]: 1533089438643897774

In [133]: hash((1, 2,(2, 3)))
Out[133]: -9209053662355515447

In [134]: hash((1, 2, [2, 3]))
-----
TypeError: Traceback (most recent call last)
Cell In[134], line 1
      1 hash((1, 2, [2, 3]))
      2
      3     TypeError: unhashable type: 'list'

In [135]: d = {}
In [136]: d[tuple([1, 2, 3])] = 5
In [137]: d
Out[137]: {(1, 2, 3): 5}
```

## Set

The screenshot shows a Jupyter Notebook interface with the following session history:

```
In [138]: set([2, 2, 2, 1, 3, 3])
Out[138]: {1, 2, 3}
```

## Python'da Set İşlemleri

Function	Alternative syntax	Description
a.add(x)	N/A	Add element x to the set a
a.clear()	N/A	Reset the set a to an empty state, discarding all of its elements
a.remove(x)	N/A	Remove element x from the set a
a.pop()	N/A	Remove an arbitrary element from the set a, raising KeyError if the set is empty
a.union(b)	a   b	All of the unique elements in a and b
a.update(b)	a  = b	Set the contents of a to be the union of the elements in a and b
a.intersection(b)	a & b	All of the elements in <i>both</i> a and b
a.intersection_update(b)	a &= b	Set the contents of a to be the intersection of the elements in a and b
a.difference(b)	a - b	The elements in a that are not in b
a.difference_update(b)	a -= b	Set a to the elements in a that are not in b
a.symmetric_difference(b)	a ^ b	All of the elements in either a or b but <i>not both</i>
a.symmetric_difference_update(b)	a ^= b	Set a to contain the elements in either a or b but <i>not both</i>
a.issubset(b)	N/A	True if the elements of a are all contained in b
a.issuperset(b)	N/A	True if the elements of b are all contained in a
a.isdisjoint(b)	N/A	True if a and b have no elements in common

```
In [144]: a = {1, 2, 3, 4, 5}
```

```
In [145]: b = {3, 4, 5, 6, 7, 8}
```

```
In [146]: c = a.copy()
```

```
In [147]: c |= b
```

```
In [148]: c
```

```
Out[148]: {1, 2, 3, 4, 5, 6, 7, 8}
```

```
In [149]: d = a.copy()
```

```
In [150]: d &= b
```

```
In [151]: d
```

```
Out[151]: {3, 4, 5}
```

- Sözcükler gibi küme öğelerinin de genellikle değişmez olması gereklidir. Liste benzeri öğelere sahip olmak için onu bir tuple'a dönüştürmeliyiz.
- Küme setleri ancak elemanları eşitse eşittir.

```
In [152]: my_data = [1, 2, 3, 4]
```

```
In [153]: my_set = {tuple(my_data)}
```

```
In [154]: my_set
```

```
Out[154]: {(1, 2, 3, 4)}
```

```
In [155]: {1, 2, 3} == {3, 2, 1}
```

```
Out[155]: True
```

## List, Set ve Dict Kavramları

---

```
In [156]: strings = ['a', 'as', 'bat', 'car', 'dove', 'python']
```

```
In [157]: [x.upper() for x in strings if len(x) > 2]
```

```
Out[157]: ['BAT', 'CAR', 'DOVE', 'PYTHON']
```

- Harf sayısı 2'den büyük olan kelime listesini upper fonksiyonu ile hepsini büyük harf yazdırın örnek. Listenin uzunluğunu ve konumunu veren örnek de aşağıdadır.

```
In [158]: unique_lengths = {len(x) for x in strings}
```

```
In [159]: unique_lengths
```

```
Out[159]: {1, 2, 3, 4, 6}
```

```
In [160]: loc_mapping = {val : index for index, val in enumerate(strings)}
```

```
In [161]: loc_mapping
```

```
Out[161]: {'a': 0, 'as': 1, 'bat': 2, 'car': 3, 'dove': 4, 'python': 5}
```

## İç İçe Listeler

The screenshot shows a Jupyter Notebook interface with a toolbar at the top and three code cells below. The toolbar includes buttons for File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and various cell operations like Run, Kernel restart, and Cell clear.

```
In [162]: all_data = [['John', 'Emily', 'Michael', 'Mary', 'Steven'],
                  ['Maria', 'Juan', 'Javier', 'Natalia', 'Pilar']]
```

```
In [163]: names_of_interest = []
for names in all_data:
    enough_es = [name for name in names if name.count('e') >= 2]
    names_of_interest.extend(enough_es)
```

```
In [164]: result = [name for names in all_data for name in names
                  if name.count('e') >= 2 ]
```

## Fonksiyonlar

The screenshot shows a Jupyter Notebook interface with a toolbar at the top and four code cells below. The toolbar includes buttons for File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and various cell operations like Run, Kernel restart, and Cell clear.

```
In [167]: def my_function(x, y, z = 1.5):
    if z > 1:
        return z * (x + y)
    else:
        return z / (x + y)
```

```
In [168]: my_function(3.114, 7, 3.5)
```

```
Out[168]: 35.399
```

```
In [169]: my_function(x=6, y=5, z=7)
```

```
Out[169]: 77
```

## Namespace, Scope ve Yerel Fonksiyonlar

The screenshot shows a Jupyter Notebook interface with the following code execution history:

- In [170]:

```
def func():
    a = []
    for i in range(5):
        a.append(i)
```
- In [171]:

```
a = []
def func():
    for i in range(5):
        a.append(i)
```
- In [172]:

```
a = None
```
- In [173]:

```
def bind_a_variable():
    global a
    a = []
```
- In [174]:

```
print(a)
```

None

## Birden Fazla Değer Döndürme

The screenshot shows a Jupyter Notebook interface with the following code execution history:

- In [178]:

```
def f():
    a = 5
    b = 6
    c = 7
    return {'a' : a, 'b' : b, 'c' : c}
```

```
a, b, c = f()
```
- In [176]:

```
return_value = f()
```

## Fonksiyonlar ve Nesneler

The screenshot shows a Jupyter Notebook interface with the following code cells:

```
In [179]: states = ['Alabama', 'Georgia!', 'Georgia', 'georgia', 'FlOrIda',
'south carolina##', 'West virginia?']

In [180]: import re

In [181]: def clean_strings(strings):
    result = []
    for value in strings:
        value = value.strip()
        value = re.sub('[!#?]', '', value)
        value = value.title()
        result.append(value)
    return result

In [182]: clean_strings(states)

Out[182]: ['Alabama',
'Georgia',
'Georgia',
'Georgia',
'Florida',
'South Carolina',
'West Virginia']
```

- String ifadeleri temizleyerek listeyi yeniden döndüren fonksiyon tanımı. Alternatif olarak remove fonksiyonu kullanılabilir.

## Lambda Fonksiyonları

The screenshot shows a Jupyter Notebook interface with the following code cells:

```
In [183]: def apply_to_list(some_list, f):
    return [f(x) for x in some_list]

In [184]: ints = [4, 0, 1, 5, 6]
apply_to_list(ints, lambda x: x * 2)

Out[184]: [8, 0, 2, 10, 12]
```

## Currying : Kısmi Argüman Uygulaması

- Currying uygulamasıyla mevcut işlevlerden yeni işlevler türetmek anlamına gelir. Örneğin, iki sayıyı toplayan basit bir fonksiyon yerine tek değişkenli bir argüman türetilabilir.

The screenshot shows a Jupyter Notebook interface with the following code examples:

```
In [ ]: def add_numbers(x, y):
           return x + y

In [ ]: add_five = lambda y: add_numbers(5, y)

In [ ]: from functools import partial
        add_five = partial(add_numbers, 5)
```

## Generatorler

The screenshot shows a Jupyter Notebook interface with the following code examples:

```
In [185]: some_dict = {'a' : 1, 'b' : 2, 'c' : 3}

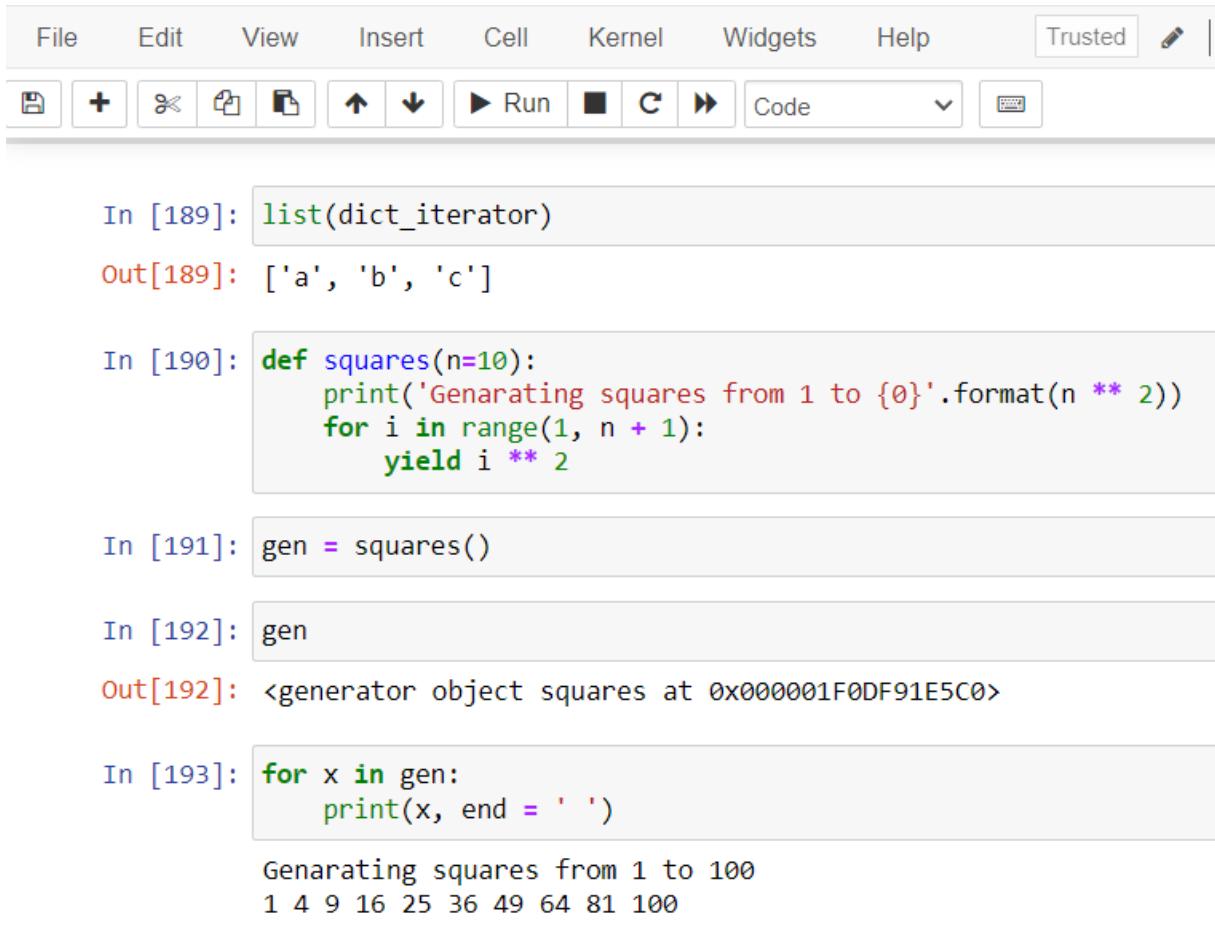
In [186]: for key in some_dict:
           print(key)

a
b
c

In [187]: dict_iterator = iter(some_dict)

In [188]: dict_iterator

Out[188]: <dict_keyiterator at 0x1f0e06d8db0>
```



```

File Edit View Insert Cell Kernel Widgets Help Trusted
[+] Run C Code

In [189]: list(dict_iterator)
Out[189]: ['a', 'b', 'c']

In [190]: def squares(n=10):
    print('Generating squares from 1 to {}'.format(n ** 2))
    for i in range(1, n + 1):
        yield i ** 2

In [191]: gen = squares()

In [192]: gen
Out[192]: <generator object squares at 0x000001F0DF91E5C0>

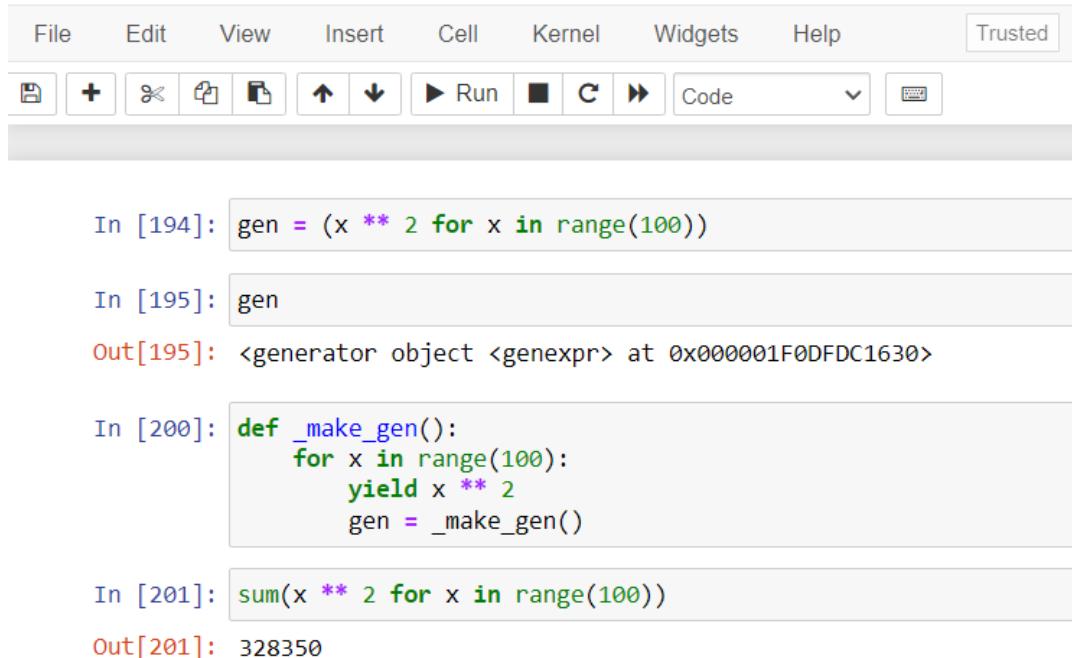
In [193]: for x in gen:
    print(x, end = ' ')

```

Generating squares from 1 to 100  
1 4 9 16 25 36 49 64 81 100

- Bir listedeki nesneler veya bir dosyadaki satırlar gibi dizeler üzerinde yineleme yapmak için önemli bir Python özelliğidir.

## Generator İfadeleri



```

File Edit View Insert Cell Kernel Widgets Help Trusted
[+] Run C Code

In [194]: gen = (x ** 2 for x in range(100))

In [195]: gen
Out[195]: <generator object <genexpr> at 0x000001F0DFDC1630>

In [200]: def _make_gen():
    for x in range(100):
        yield x ** 2
    gen = _make_gen()

In [201]: sum(x ** 2 for x in range(100))
Out[201]: 328350

```

## Itertools Modülü

- Groupby fonksiyonu ile listenin elemanlarını grupper.

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [203]: import itertools
In [204]: first_letter = lambda x: x[0]
In [205]: names = ['Alan', 'Adam', 'Wes', 'Will', 'Albert', 'Steven']
In [207]: for letter, names in itertools.groupby(names, first_letter):
              print(letter, list(names))
A ['Alan', 'Adam']
W ['Wes', 'Will']
A ['Albert']
S ['Steven']
```

## Bazı Itertools Kullanım Fonksiyonları

Function	Description
combinations(iterable, k)	Generates a sequence of all possible k-tuples of elements in the iterable, ignoring order and without replacement (see also the companion function <code>combinations_with_replacement</code> )
permutations(iterable, k)	Generates a sequence of all possible k-tuples of elements in the iterable, respecting order
groupby(iterable[, keyfunc])	Generates (key, sub-iterator) for each unique key
product(*iterables, repeat=1)	Generates the Cartesian product of the input iterables as tuples, similar to a nested for loop

## Hatalar ve Sıra Dışı Durumlar

The screenshot shows a Jupyter Notebook interface with the following session history:

- In [208]: `float('1.2345')`  
Out[208]: 1.2345
- In [209]: `float('something')`  
ValueError  
Cell In[209], line 1  
----> 1 float('something')  
ValueError: could not convert string to float: 'something'
- In [210]: `def attempt_float(x):  
 try:  
 return float(x)  
 except:  
 return x`
- In [212]: `attempt_float('1.2345')`  
Out[212]: 1.2345
- In [213]: `attempt_float('something')`  
Out[213]: 'something'
- In [214]: `float((1, 2))`  
TypeError  
Cell In[214], line 1  
----> 1 float((1, 2))  
TypeError: float() argument must be a string or a real number, not 'tuple'
- In [215]: `def attempt_float(x):  
 try:  
 return float(x)  
 except ValueError:  
 return x`
- In [216]: `attempt_float((1, 2))`  
TypeError  
Cell In[216], line 1  
----> 1 attempt\_float((1, 2))  
Cell In[215], line 3, in attempt\_float(x)  
1 def attempt\_float(x):  
2 try:  
----> 3 return float(x)  
4 except ValueError:  
5 return x  
TypeError: float() argument must be a string or a real number, not 'tuple'

The screenshot shows a Jupyter Notebook interface with the following code cells:

```
In [217]: def attempt_float(x):
    try:
        return float(x)
    except (TypeError, ValueError):
        return x
```

```
In [218]: f = open(path, 'w')

try:
    write_to_file(f)

Cell In[218], line 4
    write_to_file(f)
    ^
SyntaxError: incomplete input
```

---

```
In [219]: finally:
    f.close()

Cell In[219], line 1
    finally:
    ^
SyntaxError: invalid syntax
```

### AssertionError:

Tek başına ek içeriğe sahip olmak, standart Python yorumlayıcısıyla göre büyük bir avantajdır. %xmode magic komutunu kullanarak gösterilen bağlam miktarını, düzden ayrıntılıya kadar kontrol edebilirsiniz.

## Dosyalar ve İşletim Sistemleri

```
In [220]: path = 'examples/segismundo.txt'
```

```
In [221]: f = open(path)
```

## Python Dosya Modları

Mode	Description
r	Read-only mode
w	Write-only mode; creates a new file (erasing the data for any file with the same name)
x	Write-only mode; creates a new file, but fails if the file path already exists
a	Append to existing file (create the file if it does not already exist)
r+	Read and write
b	Add to mode for binary files (i.e., 'rb' or 'wb')
t	Text mode for files (automatically decoding bytes to Unicode). This is the default if not specified. Add t to other modes to use this (i.e., 'rt' or 'xt')

## Önemli Python Dosya Metotları ve Özellikleri

Method	Description
read([size])	Return data from file as a string, with optional size argument indicating the number of bytes to read
readlines([size])	Return list of lines in the file, with optional size argument
write(str)	Write passed string to file
writelines(strings)	Write passed sequence of strings to the file
close()	Close the handle
flush()	Flush the internal I/O buffer to disk
seek(pos)	Move to indicated file position (integer)
tell()	Return current file position as integer
closed	True if the file is closed

The screenshot shows a Jupyter Notebook interface with several code cells and their corresponding outputs.

- Cell 1: 

```
with open('tmp.txt', 'w') as handle:  
    handle.writelines(x for x in open(path) if len(x) > 1)
```
- Cell 2: 

```
with open('tmp.txt') as f:  
    lines = f.readlines()
```
- Cell 3: 

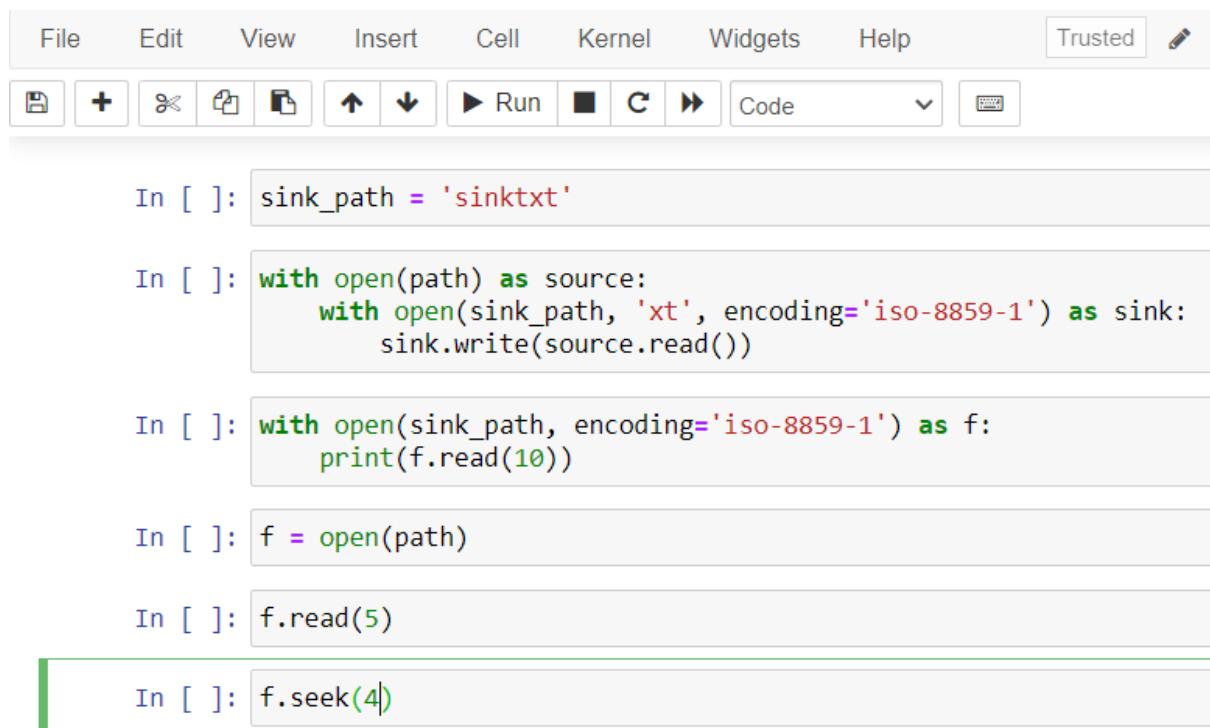
```
lines
```

Out[227]:

```
'Sueña el rico en su riqueza,\n',  
'que más cuidados le ofrece;\n',  
'sueña el pobre que padece\n',  
'su miseria y su pobreza;\n',  
'sueña el que a medrar empieza,\n',  
'sueña el que afana y pretende,\n',  
'sueña el que agravia y ofende,\n',  
'y en el mundo, en conclusión,\n',  
'todos sueñan lo que son,\n',  
'aunque ninguno lo entiende.\n'
```

## Dosyalarda Bayt ve Unicode

- Python dosyalarının (okunabilir veya yazılabilir) varsayılan davranışı metnidir; bu Python dizeleriyle (yani Unicode) çalışmayı planladığımız anlamına gelir.



The screenshot shows a Jupyter Notebook interface with the following code cells:

```
In [ ]: sink_path = 'sinktxt'

In [ ]: with open(path) as source:
         with open(sink_path, 'xt', encoding='iso-8859-1') as sink:
             sink.write(source.read())

In [ ]: with open(sink_path, encoding='iso-8859-1') as f:
         print(f.read(10))

In [ ]: f = open(path)

In [ ]: f.read(5)

In [ ]: f.seek(4)
```

## NumPy Temelleri: Diziler ve Vektörel Hesaplamalar

Python'un kısaltması olan NumPy, Python'daki sayısal hesaplama için en önemli temel paketlerden biridir. Bilimsel işlevsellik sağlayan çoğu hesaplama paketi, veri alışverişi için ortak dil olarak NumPy'ın dizi nesnelerini kullanır.

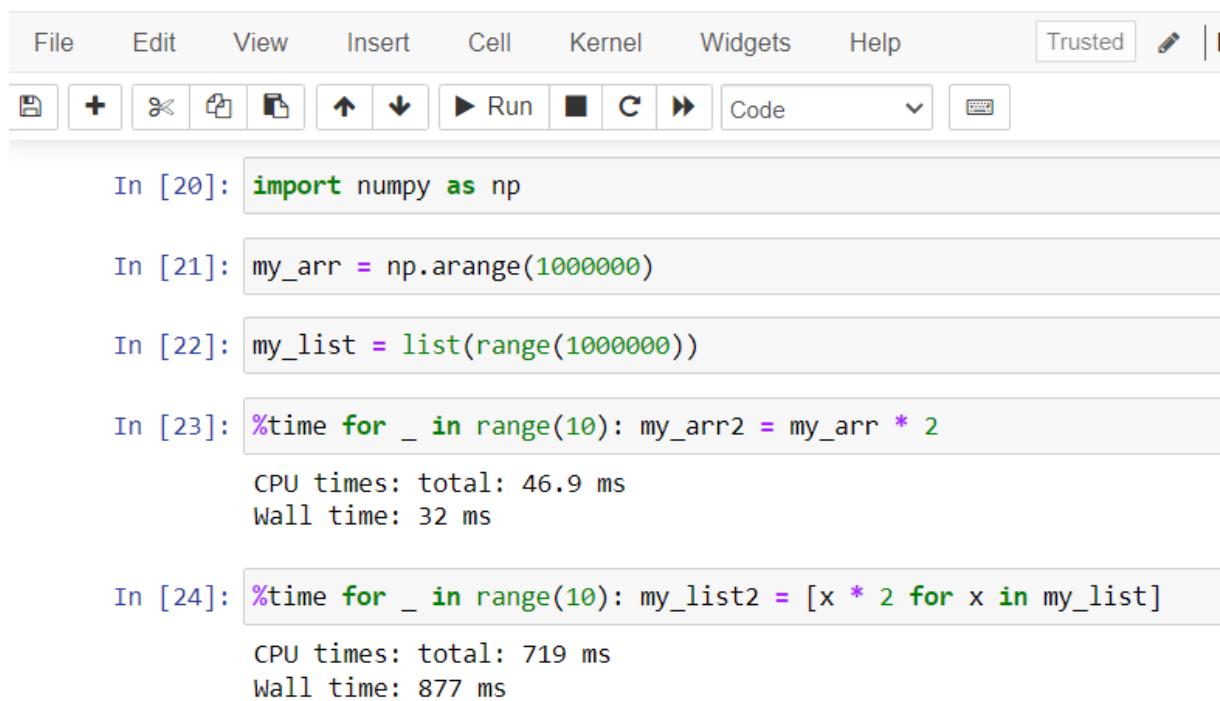
NumPy'ın Bazı İşlevleri:

- 'ndarray', hızlı dizi odaklı aritmetik işlemler ve esnek yayın yetenekleri sağlayan verimli, çok boyutlu bir dizi.
- Döngü yazmaya gerek kalmadan tüm veri dizileri üzerinde hızlı işlemler için matematiksel işlevler.
- Dizi verilerini diske okumak yazmak ve bellek eşlemeli dosyalarla çalışmak için araçlar.
- Doğrusal cebir, rastgele sayı üretme ve Fourier dönüşümü yetenekleri.

Çoğu veri analizi uygulaması için ana işlevsellik alanları:

- Veri birleştirme ve temizleme, filtrelemeyi alt kümeleme, dönüştürme ve diğer her türlü hesaplama hızlı vektörleştirilmiş dizi işlemleri.
- Sıralama, benzersiz ve ayarlama işlemleri gibi ortak dizi algoritmaları.
- Etkin tanımlayıcı istatistikler ve verilerin toplanması/özetlenmesi.
- Veri kümelerini birleştirmek ve birleştirmek için veri hizalama ve ilişkisel manipülasyonları.
- Grup bazında veri manipülasyonları (toplama, dönüştürme, fonksiyon uygulaması).

NumPy genel sayısal veri işleme için hesaplamalı bir temel sağlarken, birçok okuyucu özellikle tablo halindeki verilerle ilgili çoğu istatistik veya analitik türü için temel olarak pandaları kullanmak isteyecektir. Pandalar ayrıca NumPy'da bulunmayan zaman serisi manipülasyonu gibi alana özgü bazı işlevler de sağlar.



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

- File Edit View Insert Cell Kernel Widgets Help Trusted
- Toolbar icons: file, new, cell, cell type, run, cell, cell type, code, keyboard.
- In [20]: `import numpy as np`
- In [21]: `my_arr = np.arange(1000000)`
- In [22]: `my_list = list(range(1000000))`
- In [23]: `%time for _ in range(10): my_arr2 = my_arr * 2`  
CPU times: total: 46.9 ms  
Wall time: 32 ms
- In [24]: `%time for _ in range(10): my_list2 = [x * 2 for x in my_list]`  
CPU times: total: 719 ms  
Wall time: 877 ms

- NumPy tanımlı dizi normal dizi tanımına göre çok daha hızlı ve bellekte daha az yer kaplar.

## NumPy ndarray : Çok Boyutlu Dizi Nesnesi

- (2, 3) boyutlu rastgele bir dizi oluşturup boyut ve tip özelliği gösterildi.

The screenshot shows a Jupyter Notebook interface with the following code execution history:

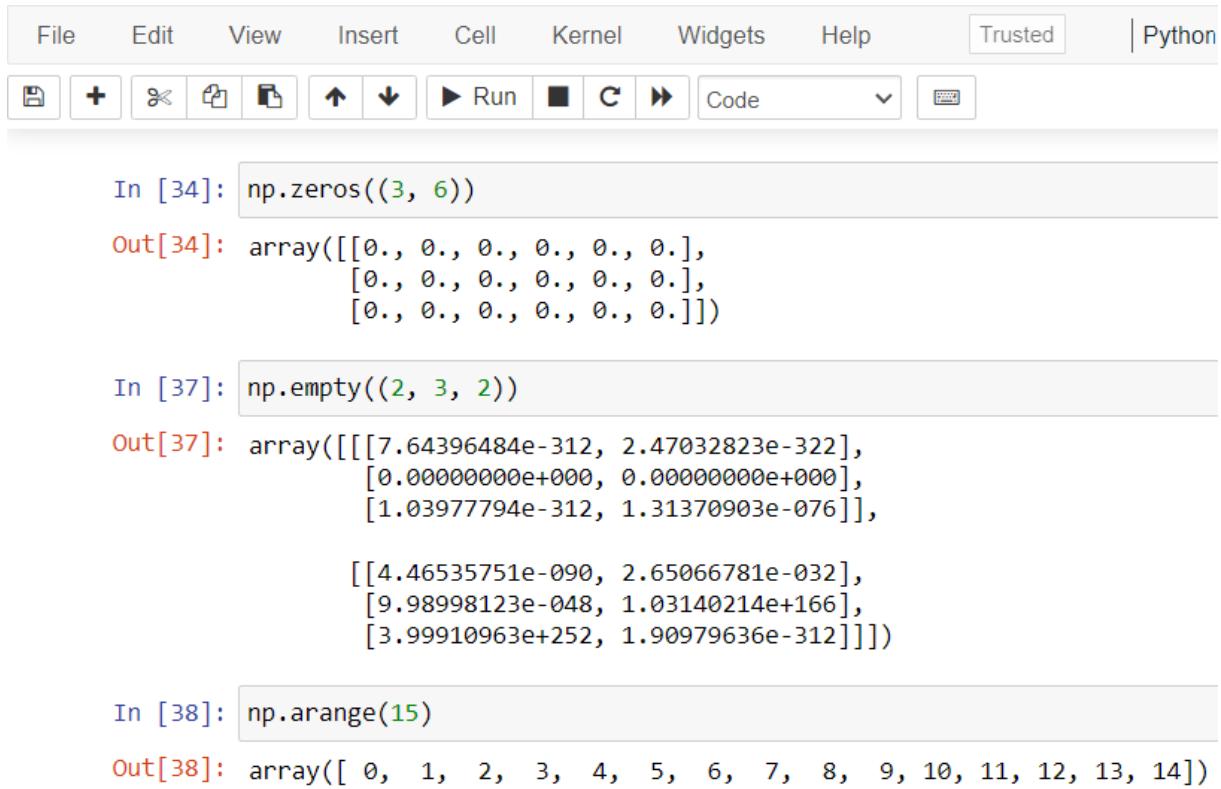
```
In [25]: import numpy as np
In [26]: data = np.random.randn(2, 3)
In [27]: data
Out[27]: array([[ 2.31699833,  0.42704633, -0.0544035 ],
   [-0.27061287, -1.52777822,  0.66047769]])
In [28]: data * 10
Out[28]: array([[ 23.16998328,   4.27046326,  -0.54403503],
   [-2.70612866, -15.2777822 ,   6.60477694]])
In [29]: data.shape
Out[29]: (2, 3)
In [30]: data.dtype
Out[30]: dtype('float64')
```

## ndarray Oluşturma

The screenshot shows a Jupyter Notebook interface with the following code execution history:

```
In [31]: data1 = [6, 7.5, 8, 0, 1]
In [32]: arr1 = np.array(data1)
In [33]: arr1
Out[33]: array([6. , 7.5, 8. , 0. , 1. ])
```

- NumPy ile dizi oluşturma işlemleri.



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

```
In [34]: np.zeros((3, 6))
Out[34]: array([[0., 0., 0., 0., 0., 0.],
   [0., 0., 0., 0., 0., 0.],
   [0., 0., 0., 0., 0., 0.]])
```

```
In [37]: np.empty((2, 3, 2))
Out[37]: array([[[7.64396484e-312, 2.47032823e-322],
   [0.00000000e+000, 0.00000000e+000],
   [1.03977794e-312, 1.31370903e-076]],

   [[4.46535751e-090, 2.65066781e-032],
   [9.98998123e-048, 1.03140214e+166],
   [3.99910963e+252, 1.90979636e-312]]])
```

```
In [38]: np.arange(15)
Out[38]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

## Dizi Oluşturma Fonksiyonları

Function	Description
array	Convert input data (list, tuple, array, or other sequence type) to an ndarray either by inferring a dtype or explicitly specifying a dtype; copies the input data by default
asarray	Convert input to ndarray, but do not copy if the input is already an ndarray
arange	Like the built-in range but returns an ndarray instead of a list
ones, ones_like	Produce an array of all 1s with the given shape and dtype; ones_like takes another array and produces a ones array of the same shape and dtype
zeros, zeros_like	Like ones and ones_like but producing arrays of 0s instead
empty, empty_like	Create new arrays by allocating new memory, but do not populate with any values like ones and zeros
full, full_like	Produce an array of the given shape and dtype with all values set to the indicated "fill value" full_like takes another array and produces a filled array of the same shape and dtype
eye, identity	Create a square N × N identity matrix (1s on the diagonal and 0s elsewhere)

## ndarray Veri Tipleri

The screenshot shows a Jupyter Notebook interface with the following code execution history:

```
In [39]: arr1 = np.array([1, 2, 3], dtype=np.float64)
In [40]: arr2 = np.array([1, 2, 3], dtype=np.int32)
In [41]: arr1.dtype
Out[41]: dtype('float64')
In [42]: arr2.dtype
Out[42]: dtype('int32')
```

## NumPy Veri Tipleri

Type	Type code	Description
int8, uint8	i1, u1	Signed and unsigned 8-bit (1 byte) integer types
int16, uint16	i2, u2	Signed and unsigned 16-bit integer types
int32, uint32	i4, u4	Signed and unsigned 32-bit integer types
int64, uint64	i8, u8	Signed and unsigned 64-bit integer types
float16	f2	Half-precision floating point
float32	f4 or f	Standard single-precision floating point; compatible with C float
float64	f8 or d	Standard double-precision floating point; compatible with C double and Python float object
float128	f16 or g	Extended-precision floating point
complex64, complex128, complex256	c8, c16, c32	Complex numbers represented by two 32, 64, or 128 floats, respectively
bool	?	Boolean type storing True and False values
object	o	Python object type; a value can be any Python object
string_	S	Fixed-length ASCII string type (1 byte per character); for example, to create a string dtype with length 10, use 'S10'
unicode_	U	Fixed-length Unicode type (number of bytes platform specific); same specification semantics as string_ (e.g., 'U10')

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

```

File Edit View Insert Cell Kernel Widgets Help Trusted
[+]
In [44]: arr = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
In [45]: arr
Out[45]: array([ 3.7, -1.2, -2.6,  0.5, 12.9, 10.1])
In [47]: arr.astype(np.int32)
Out[47]: array([ 3, -1, -2,  0, 12, 10])

```

- Tipi int olan değerler ile yeni bir dizi oluşturma örneği.

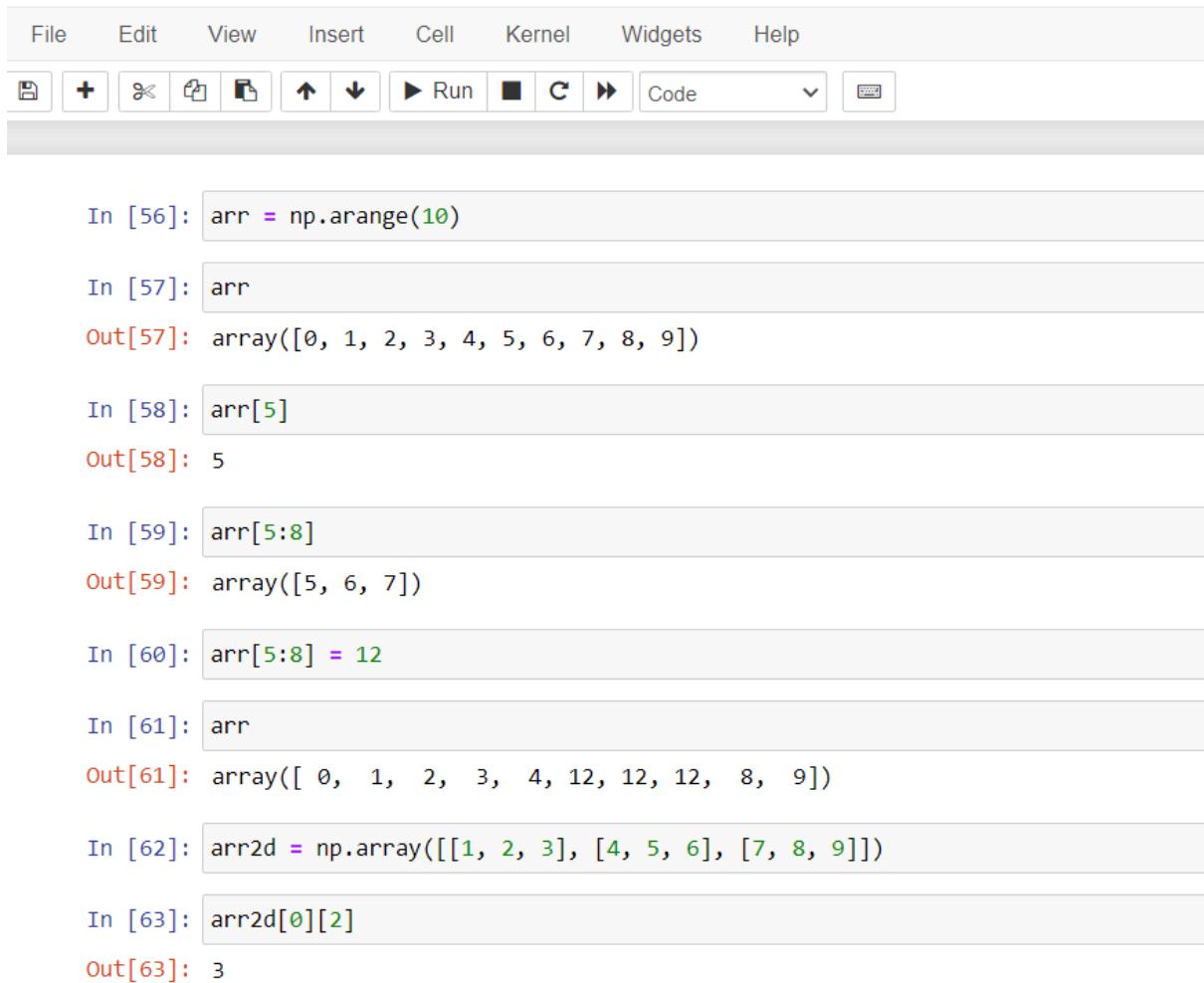
## NumPy Dizileriyle Aritmetik

```

In [48]: arr = np.array([[1., 2., 3.], [4., 5., 6.]])
In [49]: arr
Out[49]: array([[1., 2., 3.],
   [4., 5., 6.]])
In [50]: arr * arr
Out[50]: array([[ 1.,  4.,  9.],
   [16., 25., 36.]])
In [51]: 1 / arr
Out[51]: array([[1.          , 0.5        , 0.33333333],
   [0.25       , 0.2        , 0.16666667]])
In [52]: arr ** 0.5
Out[52]: array([[1.          , 1.41421356, 1.73205081],
   [2.          , 2.23606798, 2.44948974]])
In [53]: arr2 = np.array([[0., 4., 1.], [7., 2., 12.]])
In [54]: arr2
Out[54]: array([[ 0.,  4.,  1.],
   [ 7.,  2., 12.]])
In [55]: arr2 > arr
Out[55]: array([[False,  True, False],
   [ True, False,  True]])

```

## Temel İndexleme ve Dilimleme

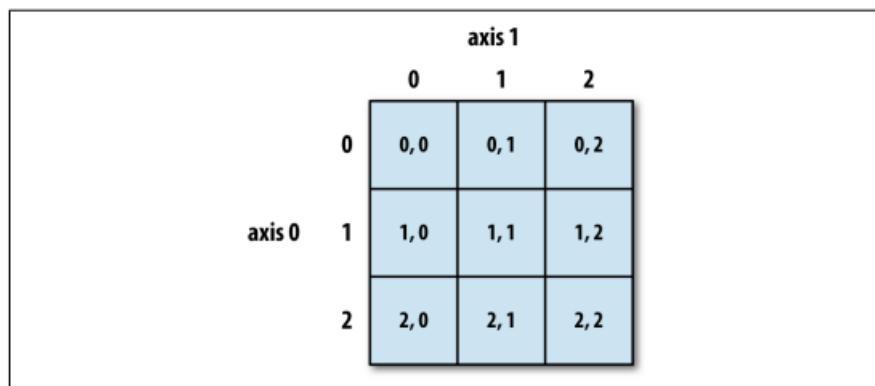


The screenshot shows a Jupyter Notebook interface with the following code execution history:

- In [56]: arr = np.arange(10)
- In [57]: arr
- Out[57]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
- In [58]: arr[5]
- Out[58]: 5
- In [59]: arr[5:8]
- Out[59]: array([5, 6, 7])
- In [60]: arr[5:8] = 12
- In [61]: arr
- Out[61]: array([ 0, 1, 2, 3, 4, 12, 12, 12, 8, 9])
- In [62]: arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
- In [63]: arr2d[0][2]
- Out[63]: 3

- Dizi oluşturulup indeks bulunması ve dilimleyerek istenileni buldurur.

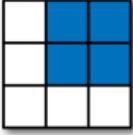
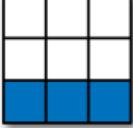
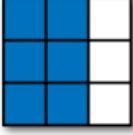
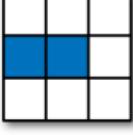
## NumPy Dizilerinde İndeksleme



A diagram of a 3x3 2D NumPy array. The array is labeled "axis 1" vertically on the right and "axis 0" horizontally at the bottom. The elements are indexed as follows:

	0	1	2
0	0, 0	0, 1	0, 2
1	1, 0	1, 1	1, 2
2	2, 0	2, 1	2, 2

## İki Boyutlu Dizi Dilimleme

	Expression	Shape
	arr[::2, ::1]	(2, 2)
	arr[2] arr[2, ::] arr[2:, ::]	(3,) (3,) (1, 3)
	arr[:, ::2]	(3, 2)
	arr[1, ::2] arr[1:2, ::2]	(2,) (1, 2)

## Boolean İndeksleme

```
In [64]: names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])

In [65]: data = np.random.rand(7, 4)

In [66]: names
Out[66]: array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')

In [67]: data
Out[67]: array([[ 0.6462285 ,  0.41795451,  0.15446267,  0.60340428],
       [ 0.39078234,  0.1870174 ,  0.67944349,  0.46394442],
       [ 0.82084118,  0.9601389 ,  0.8314062 ,  0.75482453],
       [ 0.31185891,  0.96927627,  0.34999063,  0.74812754],
       [ 0.70205258,  0.44594573,  0.19098329,  0.61958724],
       [ 0.53870869,  0.80438603,  0.41120082,  0.22381079],
       [ 0.82168924,  0.03175546,  0.68215965,  0.70988664]])]

In [68]: names == 'Bob'
Out[68]: array([ True, False, False,  True, False, False, False])

In [69]: data[names == 'Bob', 2:]
Out[69]: array([[ 0.15446267,  0.60340428],
       [ 0.34999063,  0.74812754]])
```

```
In [70]: names != 'Bob'
Out[70]: array([False, True, True, False, True, True, True])

In [71]: cond = names == 'Bob'

In [72]: data[~cond]

Out[72]: array([[0.39078234, 0.1870174 , 0.67944349, 0.46394442],
   [0.82084118, 0.9601389 , 0.8314062 , 0.75482453],
   [0.70205258, 0.44594573, 0.19098329, 0.61958724],
   [0.53870869, 0.80438603, 0.41120082, 0.22381079],
   [0.82168924, 0.03175546, 0.68215965, 0.70988664]])

In [73]: data[data < 0] = 0

In [74]: data

Out[74]: array([[0.6462285 , 0.41795451, 0.15446267, 0.60340428],
   [0.39078234, 0.1870174 , 0.67944349, 0.46394442],
   [0.82084118, 0.9601389 , 0.8314062 , 0.75482453],
   [0.31185891, 0.96927627, 0.34999063, 0.74812754],
   [0.70205258, 0.44594573, 0.19098329, 0.61958724],
   [0.53870869, 0.80438603, 0.41120082, 0.22381079],
   [0.82168924, 0.03175546, 0.68215965, 0.70988664]])
```

- Rand fonksiyonuyla rastgele bir dizi oluşturup üzerinde işlemler yapıtıldı. “=” operatörüyle karşılaştırma yapıldı. “!=” operatörüyle true, false kontrolü yapıldı. “~” operatörü ile deşilinin kontrolü yapıldı. Farklı operatörlerle de çok farklı kontroller yapılabilir.

## Fancy indeksleme, tamsayı

```
In [3]: import numpy as np
In [5]: arr = np.empty((8, 4))
In [7]: for i in range(8):
         arr[i] = i
In [8]: arr

Out[8]: array([[0., 0., 0., 0.],
   [1., 1., 1., 1.],
   [2., 2., 2., 2.],
   [3., 3., 3., 3.],
   [4., 4., 4., 4.],
   [5., 5., 5., 5.],
   [6., 6., 6., 6.],
   [7., 7., 7., 7.]])
```

File Edit View Insert Cell Kernel Widgets Help

+ X ↗ ↘ ↙ ↘ Run □ C ► Code ▾

In [9]: arr[[4, 3, 0, 6]]

Out[9]: array([[4., 4., 4., 4.],  
[3., 3., 3., 3.],  
[0., 0., 0., 0.],  
[6., 6., 6., 6.]])

In [10]: arr[[-3, -5, -7]]

Out[10]: array([[5., 5., 5., 5.],  
[3., 3., 3., 3.],  
[1., 1., 1., 1.]])

In [15]: arr = np.arange(32).reshape((8, 4))

In [16]: arr

Out[16]: array([[ 0, 1, 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]])

In [17]: arr[[1, 5, 7, 2], [0, 3, 1, 2]]]

Out[17]: array([ 4, 23, 29, 10])

- Fancy indeksleme, tamsayı dizileri kullanılarak indekslemeyi tanımlamak için benimsenen bir terimdir.
  - Fancy indekslemenin davranışları biraz farklıdır. Bu, matrisin satır ve sütunlarının bir alt kümesinin seçilmesiyle oluşturulan dikdörtgen bölgedir. Bunu elde etmenin yolu dilimlemeden farklı olarak fancy indeksleme verileri her zaman yeni bir diziye kopyalandığını unutmayalım.

## Dizilerin Transpoze Edilmesi ve Eksenlerin Değiştirilmesi

- Aktarma, herhangi bir şeyi kopyalamadan, benzer şekilde altta yatan verilere ilişkin bir görünüm döndüren özel bir yeniden şekillendirme biçimidir. Diziler transpoze yöntemine ve ayrıca özel T özelliğine sahiptir.

```
In [18]: arr = np.arange(15).reshape((3, 5))
```

```
In [19]: arr
```

```
Out[19]: array([[ 0,  1,  2,  3,  4],
   [ 5,  6,  7,  8,  9],
   [10, 11, 12, 13, 14]])
```

```
In [20]: arr.T
```

```
Out[20]: array([[ 0,  5, 10],
   [ 1,  6, 11],
   [ 2,  7, 12],
   [ 3,  8, 13],
   [ 4,  9, 14]])
```

```
In [23]: arr
```

```
Out[23]: array([[[ 0,  1,  2,  3],
   [ 4,  5,  6,  7]],
   [[ 8,  9, 10, 11],
   [12, 13, 14, 15]]])
```

```
In [24]: arr.transpose((1, 0, 2))
```

```
Out[24]: array([[[ 0,  1,  2,  3],
   [ 8,  9, 10, 11]],
   [[ 4,  5,  6,  7],
   [12, 13, 14, 15]]])
```

```
In [25]: arr
```

```
Out[25]: array([[[ 0,  1,  2,  3],
   [ 4,  5,  6,  7]],
   [[ 8,  9, 10, 11],
   [12, 13, 14, 15]]])
```

```
In [26]: arr.swapaxes(1, 2)
```

```
Out[26]: array([[[ 0,  4],
   [ 1,  5],
   [ 2,  6],
   [ 3,  7]],
   [[ 8, 12],
   [ 9, 13],
   [10, 14],
   [11, 15]]])
```

## Evransel Fonksiyonlar : Hızlı Elemanlar – Bilge Dizi Fonksiyonları

```
File Edit View Insert Cell Kernel Widgets Help
+ Run Code

In [28]: arr = np.arange(10)

In [29]: arr
Out[29]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [30]: np.sqrt(arr)
Out[30]: array([0.          , 1.          , 1.41421356, 1.73205081, 2.
   2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])

In [31]: np.exp(arr)
Out[31]: array([1.0000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
   5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
   2.98095799e+03, 8.10308393e+03])

In [32]: arr = np.random.randn(7) * 5

In [33]: arr
Out[33]: array([ 2.05704869, -8.23669003,  5.03880542,  9.67570443, -1.71439174,
   4.2639458 , -5.34887473])

In [34]: remainder, whole_part = np.modf(arr)
In [35]: remainder
Out[35]: array([ 0.05704869, -0.23669003,  0.03880542,  0.67570443, -0.71439174,
   0.2639458 , -0.34887473])

In [36]: whole_part
Out[36]: array([ 2., -8.,  5.,  9., -1.,  4., -5.])

In [37]: arr
Out[37]: array([ 2.05704869, -8.23669003,  5.03880542,  9.67570443, -1.71439174,
   4.2639458 , -5.34887473])

In [38]: np.sqrt(arr)
C:\Users\Eda\AppData\Local\Temp\ipykernel_8352\2296558006.py:1: RuntimeWarning: invalid value encountered in sqrt
np.sqrt(arr)
Out[38]: array([1.4342415 ,      nan, 2.24472836, 3.11057944,      nan,
   2.0649324 ,      nan])

In [39]: np.sqrt(arr, arr)
C:\Users\Eda\AppData\Local\Temp\ipykernel_8352\269955669.py:1: RuntimeWarning: invalid value encountered in sqrt
np.sqrt(arr, arr)
Out[39]: array([1.4342415 ,      nan, 2.24472836, 3.11057944,      nan,
   2.0649324 ,      nan])

In [40]: arr
Out[40]: array([1.4342415 ,      nan, 2.24472836, 3.11057944,      nan,
   2.0649324 ,      nan])
```

- Birçok ufunc, sqrt veya exp gibi basit öğe bazında dönüşümlerdir.
- Bunlara tekli işlevler denir. Add veya maximum gibi değerleri iki dizi alır (böylece ikili ufunc'lar) ve sonuç olarak tek bir dizi döndürür.
- Whole\_part fonksiyonuyla tam kısmı remainder ile ondalıklı kısmı gösterilir.

## Tekli Fonksiyon İşlevleri

Function	Description
abs, fabs	Compute the absolute value element-wise for integer, floating-point, or complex values
sqrt	Compute the square root of each element (equivalent to <code>arr ** 0.5</code> )
square	Compute the square of each element (equivalent to <code>arr ** 2</code> )
exp	Compute the exponent $e^x$ of each element
log, log10, log2, log1p	Natural logarithm (base $e$ ), log base 10, log base 2, and $\log(1 + x)$ , respectively
sign	Compute the sign of each element: 1 (positive), 0 (zero), or -1 (negative)
ceil	Compute the ceiling of each element (i.e., the smallest integer greater than or equal to that number)
floor	Compute the floor of each element (i.e., the largest integer less than or equal to each element)
rint	Round elements to the nearest integer, preserving the <code>dtype</code>
modf	Return fractional and integral parts of array as a separate array
isnan	Return boolean array indicating whether each value is NaN (Not a Number)
isfinite, isinf	Return boolean array indicating whether each element is finite (non- <code>NaN</code> , non- <code>Inf</code> ) or infinite, respectively
cos, cosh, sin, sinh, tan, tanh	Regular and hyperbolic trigonometric functions
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	Inverse trigonometric functions
logical_not	Compute truth value of <code>not x</code> element-wise (equivalent to <code>~arr</code> ).

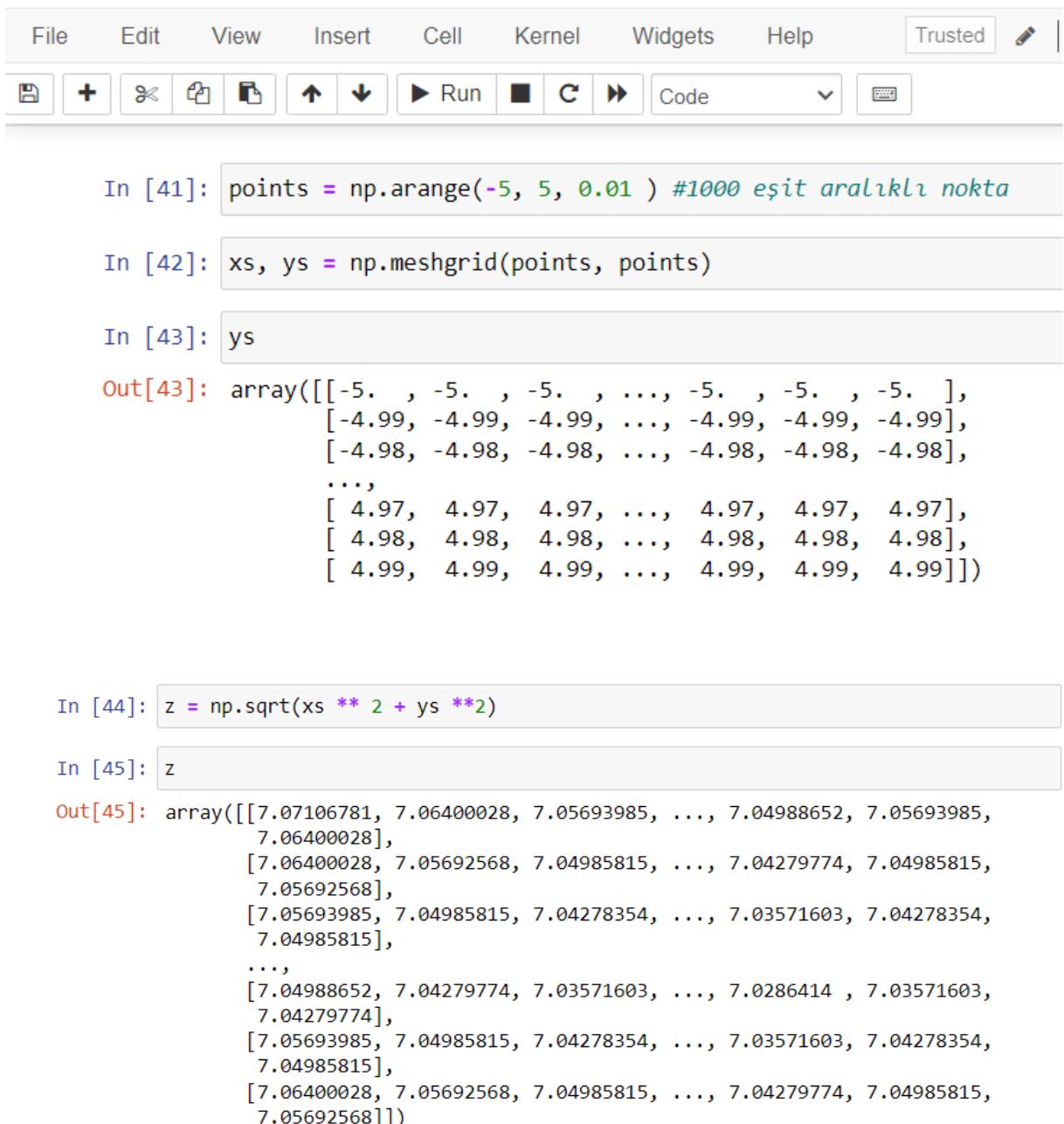
## İkili Evrensel Fonksiyonlar

Function	Description
add	Add corresponding elements in arrays
subtract	Subtract elements in second array from first array
multiply	Multiply array elements
divide, floor_divide	Divide or floor divide (truncating the remainder)
power	Raise elements in first array to powers indicated in second array
maximum, fmax	Element-wise maximum; <code>fmax</code> ignores NaN
minimum, fmin	Element-wise minimum; <code>fmin</code> ignores NaN
mod	Element-wise modulus (remainder of division)
copysign	Copy sign of values in second argument to values in first argument
greater, greater_equal, less, less_equal, equal, not_equal	Perform element-wise comparison, yielding boolean array (equivalent to infix operators <code>&gt;</code> , <code>&gt;=</code> , <code>&lt;</code> , <code>&lt;=</code> , <code>==</code> , <code>!=</code> )
logical_and, logical_or, logical_xor	Compute element-wise truth value of logical operation (equivalent to infix operators <code>&amp;</code> , <code> </code> , <code>^</code> )

## Dizilerle Dizi Odaklı Programlama

- NumPy dizilerini kullanmak, birçok türde veri işleme görevini şu şekilde ifade etmenize olanak sağlar:  
Aksi takdirde döngü yazma gerektirebilecek kısa dizi ifadeleri. Bu uygulamanın Açık döngülerin dizi ifadeleriyle değiştirilmesine genellikle vektörleştirme adı verilir.

Basit bir örnek olarak,  $\sqrt{x^2 + y^2}$  fonksiyonunu değerlendirmek istediğimizi varsayıalım. düzenli bir değerler tablosu boyunca. np.meshgrid işlevi iki adet 1 boyutlu dizi alır ve iki dizideki tüm  $(x, y)$  çiftlerine karşılık gelen iki 2 boyutlu matris üretir:



The screenshot shows a Jupyter Notebook interface with the following code execution history:

```
In [41]: points = np.arange(-5, 5, 0.01) #1000 eşit aralıklı noktası
In [42]: xs, ys = np.meshgrid(points, points)
In [43]: ys
Out[43]: array([[ -5. , -5. , -5. , ..., -5. , -5. , -5. ],
   [-4.99, -4.99, -4.99, ..., -4.99, -4.99, -4.99],
   [-4.98, -4.98, -4.98, ..., -4.98, -4.98, -4.98],
   ...,
   [ 4.97,  4.97,  4.97, ...,  4.97,  4.97,  4.97],
   [ 4.98,  4.98,  4.98, ...,  4.98,  4.98,  4.98],
   [ 4.99,  4.99,  4.99, ...,  4.99,  4.99,  4.99]])
```

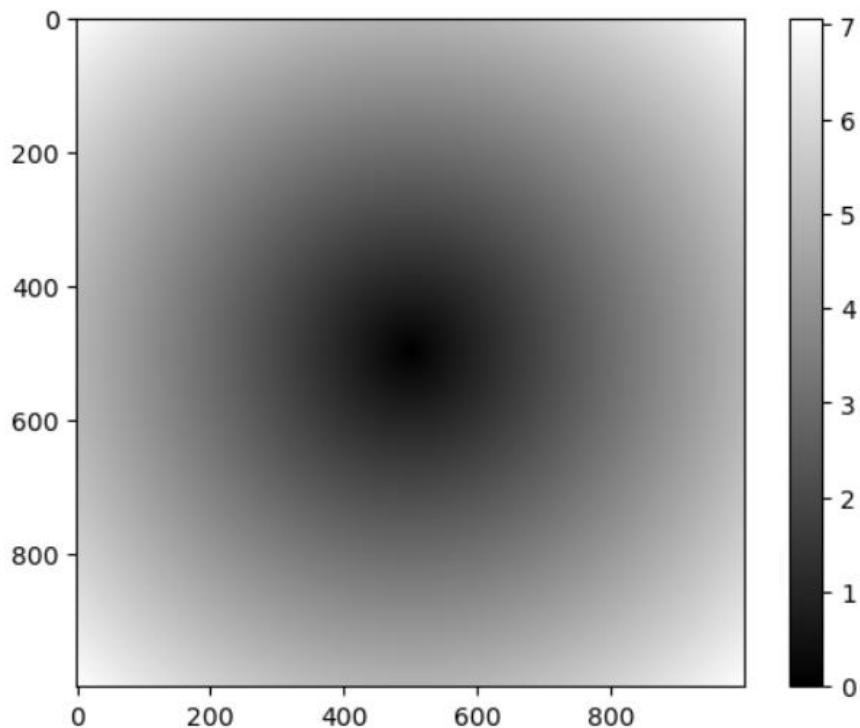
```
In [44]: z = np.sqrt(xs ** 2 + ys **2)
In [45]: z
Out[45]: array([[ 7.07106781,  7.06400028,  7.05693985, ...,  7.04988652,  7.05693985,
   7.06400028],
   [ 7.06400028,  7.05692568,  7.04985815, ...,  7.04279774,  7.04985815,
   7.05692568],
   [ 7.05693985,  7.04985815,  7.04278354, ...,  7.03571603,  7.04278354,
   7.04985815],
   ...,
   [ 7.04988652,  7.04279774,  7.03571603, ...,  7.0286414 ,  7.03571603,
   7.04279774],
   [ 7.05693985,  7.04985815,  7.04278354, ...,  7.03571603,  7.04278354,
   7.04985815],
   [ 7.06400028,  7.05692568,  7.04985815, ...,  7.04279774,  7.04985815,
   7.05692568]])
```

- Burada iki boyutlu fonksiyon değerleri dizisinden bir görüntü grafiği oluşturmak için matplotlib fonksiyonunu imshow kullanıdım.

```
In [47]: import matplotlib.pyplot as plt

In [48]: plt.imshow(z, cmap=plt.cm.gray); plt.colorbar()

Out[48]: <matplotlib.colorbar.Colorbar at 0x2310164f3d0>
```



## Koşullu Mantığı Dizi İşlemleri Olarak İfade Etme

```
In [54]: xarr = np.array([1.1, 1.2, 1.3, 1.4, 1.5])

In [55]: yarr = np.array([2.1, 2.2, 2.3, 2.4, 2.5])

In [56]: cond = np.array([True, False, True, True, False])

In [59]: result = [(x if c else y)]
         for x, y, c in zip(xarr, yarr, cond)

In [169]: result
Out[169]: [1.1, 2.2, 1.3, 1.4, 2.5]
```

- Diyelim ki karşılık gelen değer xarr'dan bir değer almak istediğimizi varsayıyalım. cond True'dur ve aksi halde değeri yarr'dan alır.

```
In [60]: result = np.where(cond, xarr, yarr)

In [61]: result
Out[61]: array([1.1, 2.2, 1.3, 1.4, 2.5])

In [67]: arr = np.random.randn(4, 4)

In [68]: arr
Out[68]: array([[ 0.1301468 ,  0.00461974, -0.41848998,  0.01684826],
   [-0.14113044, -0.26102031,  0.57563363,  0.20414482],
   [-0.98242526,  0.90169738,  0.41175398, -0.46385087],
   [ 0.15013668,  0.1405467 ,  0.56712363,  0.83325821]])

In [69]: arr > 0
Out[69]: array([[ True,  True, False,  True],
   [False, False,  True,  True],
   [False,  True,  True, False],
   [ True,  True,  True,  True]])

In [70]: np.where(arr > 0, 2, -2)
Out[70]: array([[ 2,  2, -2,  2],
   [-2, -2,  2,  2],
   [-2,  2,  2, -2],
   [ 2,  2,  2,  2]])
```

## Matematiksel ve İstatistiksel Yöntemler

- Toplam, ortalama ve eksen istatistiksel hesaplamalar örnekleri.

```
In [71]: arr = np.random.randn(5, 4)

In [72]: arr
Out[72]: array([[-1.84070442,  1.44542613, -0.07067449, -1.01359606],
   [ 1.06158604,  1.47675852, -1.21487565,  1.36752793],
   [ 0.903842 ,  0.19127604, -0.78561117, -0.91014913],
   [-1.02786588, -1.06299771, -1.33746097, -0.23611916],
   [-0.66917646,  1.27316119, -0.34269035, -1.12088419]])

In [73]: arr.mean()
Out[73]: -0.19566138922621965

In [74]: np.mean(arr)
Out[74]: -0.19566138922621965
```

```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)
In [75]: arr.sum()
Out[75]: -3.913227784524393

In [76]: arr.mean(axis=1)
Out[76]: array([-0.36988721,  0.67274921, -0.15016056, -0.91611093, -0.21489745])

In [77]: arr = np.array([0, 1, 2, 3, 4, 5, 6, 7])
In [78]: arr.cumsum()
Out[78]: array([ 0,  1,  3,  6, 10, 15, 21, 28])

In [80]: arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
In [81]: arr.cumprod(axis=1)
Out[81]: array([[ 0,  0,  0],
               [ 3, 12, 60],
               [ 6, 42, 336]])

```

## Temel İstatistiksel Dizi Yöntemleri

Method	Description
sum	Sum of all the elements in the array or along an axis; zero-length arrays have sum 0
mean	Arithmetic mean; zero-length arrays have NaN mean
std, var	Standard deviation and variance, respectively, with optional degrees of freedom adjustment (default denominator n)
min, max	Minimum and maximum
argmin, argmax	Indices of minimum and maximum elements, respectively
cumsum	Cumulative sum of elements starting from 0
cumprod	Cumulative product of elements starting from 1

## Boolean Dizi Yöntemleri

```

In [82]: arr = np.random.randn(100)
In [83]: (arr > 0).sum()
Out[83]: 45

```

- Pozitif değerlerin toplamı örneği.

## Sorting

```
In [84]: arr = np.random.randn(6)

In [85]: arr

Out[85]: array([ 1.04307227, -0.37358542,  1.06526478,  1.02511452, -0.62132398,
   0.47980014])

In [86]: arr.sort()

In [87]: arr

Out[87]: array([-0.62132398, -0.37358542,  0.47980014,  1.02511452,  1.04307227,
   1.06526478])

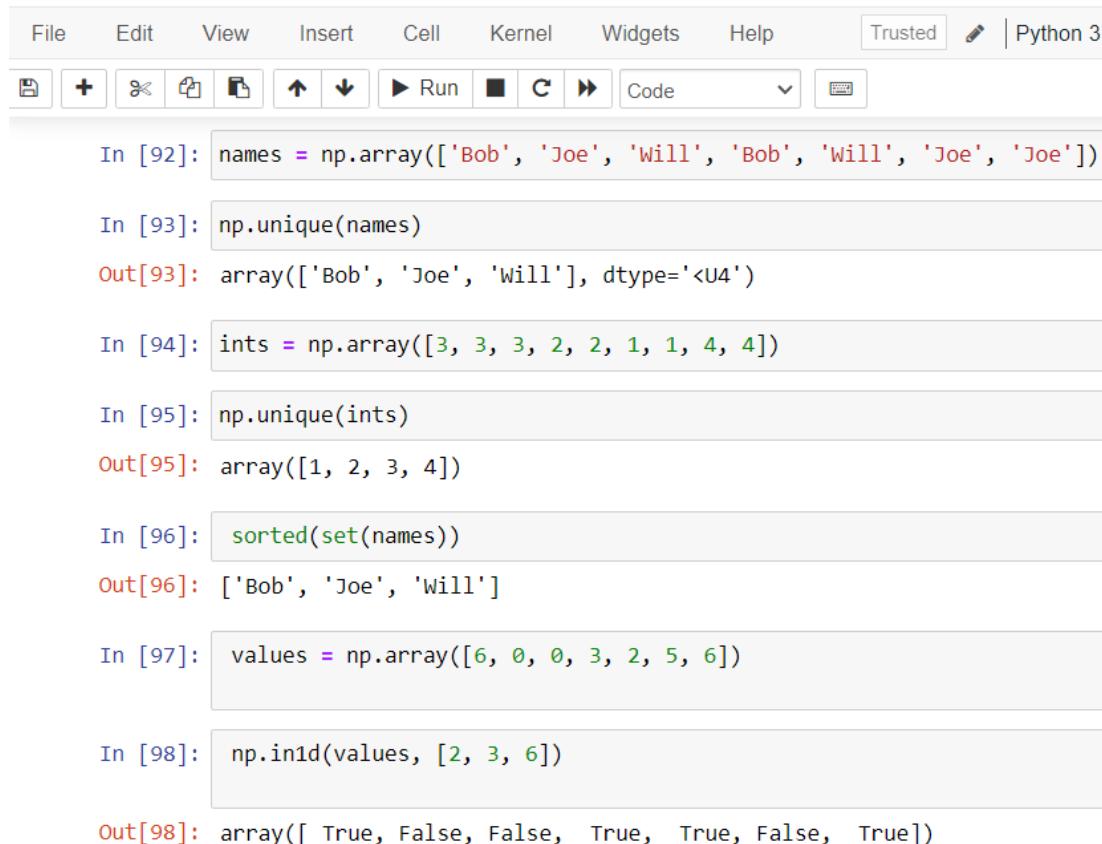
In [89]: large_arr = np.random.randn(1000)

In [90]: large_arr.sort()

In [91]: large_arr[int(0.05 * len(large_arr))] #%5'lik dilim

Out[91]: -1.6654529725454674
```

## Benzersiz ve Diğer Set Mantığı



The screenshot shows a Jupyter Notebook interface with the following code cells:

- In [92]: `names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])`
- In [93]: `np.unique(names)`
- Out[93]: `array(['Bob', 'Joe', 'Will'], dtype='<U4')`
- In [94]: `ints = np.array([3, 3, 3, 2, 2, 1, 1, 4, 4])`
- In [95]: `np.unique(ints)`
- Out[95]: `array([1, 2, 3, 4])`
- In [96]: `sorted(set(names))`
- Out[96]: `['Bob', 'Joe', 'Will']`
- In [97]: `values = np.array([6, 0, 0, 3, 2, 5, 6])`
- In [98]: `np.in1d(values, [2, 3, 6])`
- Out[98]: `array([ True, False, False, True, True, False, True])`

## Dizi Kümesi İşlemleri

Method	Description
unique(x)	Compute the sorted, unique elements in x
intersect1d(x, y)	Compute the sorted, common elements in x and y
union1d(x, y)	Compute the sorted union of elements
in1d(x, y)	Compute a boolean array indicating whether each element of x is contained in y
setdiff1d(x, y)	Set difference, elements in x that are not in y
setxor1d(x, y)	Set symmetric differences; elements that are in either of the arrays, but not both

## Dizilerle Dosya Giriş ve Çıkışı

- Dosya yükleme ve kaydetme fonksiyonlarının işlevleri örneği.

```
In [99]: arr = np.arange(10)
```

```
In [100]: np.save('some_array', arr)
```

```
In [101]: np.load('some_array.npy')
```

```
Out[101]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [102]: np.savez('array_archive.npz', a=arr, b=arr)
```

```
In [103]: arch = np.load('array_archive.npz')
```

```
In [104]: arch['b']
```

```
Out[104]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

## Lineer Cebir

- Matris çarpımı, ayırtırmalar, determinantlar ve diğerleri gibi doğrusal cebir kare matris matematiği herhangi bir dizi kütüphanesinin önemli bir parçasıdır.
- Bazı dillerden farklı olarak MATLAB gibi, iki boyutlu iki diziyi \* ile çarpmak eleman bazında bir işlemidir. Böylece hem bir dizi hem de bir nokta işlevi vardır.

The screenshot shows a Jupyter Notebook interface with the following code execution history:

- In [105]: `x = np.array([[1., 2., 3.], [4., 5., 6.]])`
- In [106]: `y = np.array([[6., 23.], [-1., 7.], [8., 9.]])`
- In [107]: `x`  
Out[107]: `array([[1., 2., 3.],  
[4., 5., 6.]])`
- In [108]: `y`  
Out[108]: `array([[ 6., 23.],  
[-1., 7.],  
[ 8., 9.]])`
- In [109]: `x.dot(y)`  
Out[109]: `array([[ 28., 64.],  
[ 67., 181.]])`
- In [110]: `np.dot(x, y)`  
Out[110]: `array([[ 28., 64.],  
[ 67., 181.]])`
- In [111]: `np.dot(x, np.ones(3))`  
Out[111]: `array([ 6., 15.])`
- In [112]: `x @ np.ones(3)`  
Out[112]: `array([ 6., 15.])`

- “dot” fonksiyonu ile eşitleme yapılır.
- Boyutsal çarpımda sonuç küçük boyutlu ile aynı boyuttadır.
- @ simbolü (Python 3.5'ten itibaren) aynı zamanda matris işlemlerini gerçekleştiren bir infix operatörü olarak da çalışır.
- linalg'in standart bir matris ayrıştırma seti ve tersi gibi belirleyici şeyleri vardır. Bunlar, MATLAB ve R gibi diğer dillerde kullanılan aynı endüstri standarı doğrusal cebir kitaplıkları aracılığıyla, başlık altında uygulanır. BLAS, LAPACK veya muhtemelen (NumPy yapınızı bağlı olarak) tescilli Intel MKL (Matematik Çekirdek Kütüphanesi):

```
In [118]: from numpy.linalg import inv, qr

In [119]: X = np.random.randn(5, 5)

In [121]: mat = X.T.dot(X)

In [122]: inv(mat)

Out[122]: array([[ 0.50487655,  0.54522878, -0.27406031, -0.25337175,  0.64806327],
   [ 0.54522878,  0.89979599, -0.39224516, -0.26765706,  1.00330137],
   [-0.27406031, -0.39224516,  0.45018252,  0.28593458, -0.52205424],
   [-0.25337175, -0.26765706,  0.28593458,  0.35942687, -0.41915439],
   [ 0.64806327,  1.00330137, -0.52205424, -0.41915439,  1.36046313]])
```

```
In [123]: mat.dot(inv(mat))

Out[123]: array([[ 1.00000000e+00, -1.11022302e-16,  1.45716772e-16,
   1.52655666e-16,  2.91433544e-16],
   [-8.88178420e-16,  1.00000000e+00,  0.00000000e+00,
   4.44089210e-16, -2.66453526e-15],
   [-5.55111512e-17,  5.55111512e-17,  1.00000000e+00,
   -2.22044605e-16, -2.77555756e-16],
   [ 4.44089210e-16,  0.00000000e+00, -2.22044605e-16,
   1.00000000e+00,  8.88178420e-16],
   [ 0.00000000e+00, -8.88178420e-16,  4.44089210e-16,
   0.00000000e+00,  1.00000000e+00]])
```

```
In [126]: q, r = qr(mat)

In [127]: r

Out[127]: array([[-8.39666951,  8.02788378,  2.95109976, -6.39258282, -2.75108265],
   [ 0.          , -7.98647403, -1.32545455,  3.38159651,  6.89848584],
   [ 0.          ,  0.          , -6.19798826,  4.87142754, -1.0184509 ],
   [ 0.          ,  0.          ,  0.          , -3.86822161, -1.35275907],
   [ 0.          ,  0.          ,  0.          ,  0.          ,  0.51807974]])
```

## Yaygın Olarak Kullanılan “numpy.linalg” İşlevleri

Function	Description
diag	Return the diagonal (or off-diagonal) elements of a square matrix as a 1D array, or convert a 1D array into a square matrix with zeros on the off-diagonal
dot	Matrix multiplication
trace	Compute the sum of the diagonal elements
det	Compute the matrix determinant
eig	Compute the eigenvalues and eigenvectors of a square matrix
inv	Compute the inverse of a square matrix
pinv	Compute the Moore-Penrose pseudo-inverse of a matrix
qr	Compute the QR decomposition
svd	Compute the singular value decomposition (SVD)
solve	Solve the linear system $Ax = b$ for $x$ , where $A$ is a square matrix
lstsq	Compute the least-squares solution to $Ax = b$

## Sözde Rastgele Sayı Üretimi

Python'un yerleşik rastgele modülü, aksine, bir seferde yalnızca bir değeri örnekler.

Bu kıyaslamada görebileceğiniz gibi, “numpy.random” büyülüüğün oldukça üzerindedir çok büyük numuneler oluşturmak için daha hızlıdır.

- NumPy'İN rastgele sayı üretme tohumunu “np.random.seed” kullanarak değiştirebilirsiniz.

```
In [128]: samples = np.random.normal(size=(4, 4))

In [129]: samples
Out[129]: array([[ 1.58343977, -0.35473392,  0.56115444,  0.18104016],
   [-0.61665135,  0.6104343 , -0.60866021,  1.49143549],
   [ 0.69281447,  1.04406484,  0.76388091,  1.64750117],
   [-0.43931716,  1.55664572,  1.22583693,  0.73061682]])

In [ ]: from random import normalvariate

In [130]: N = 1000000

In [ ]: %timeit samples = [normalvariate(0, 1) for _ in range(N)]
1.77 s +- 126 ms per loop (mean +- std. dev. of 7 runs, 1 loop each)

In [ ]: %timeit np.random.normal(size=N)
61.7 ms +- 1.32 ms per loop (mean +- std. dev. of 7 runs, 10 loops each)

In [132]: np.random.seed(1234)
```

## Numpy.random İşlevlerinin Kısmı Listesi

Function	Description
seed	Seed the random number generator
permutation	Return a random permutation of a sequence, or return a permuted range
shuffle	Randomly permute a sequence in-place
rand	Draw samples from a uniform distribution
randint	Draw random integers from a given low-to-high range
randn	Draw samples from a normal distribution with mean 0 and standard deviation 1 (MATLAB-like interface)
binomial	Draw samples from a binomial distribution
normal	Draw samples from a normal (Gaussian) distribution
beta	Draw samples from a beta distribution
chisquare	Draw samples from a chi-square distribution
gamma	Draw samples from a gamma distribution
uniform	Draw samples from a uniform [0, 1) distribution

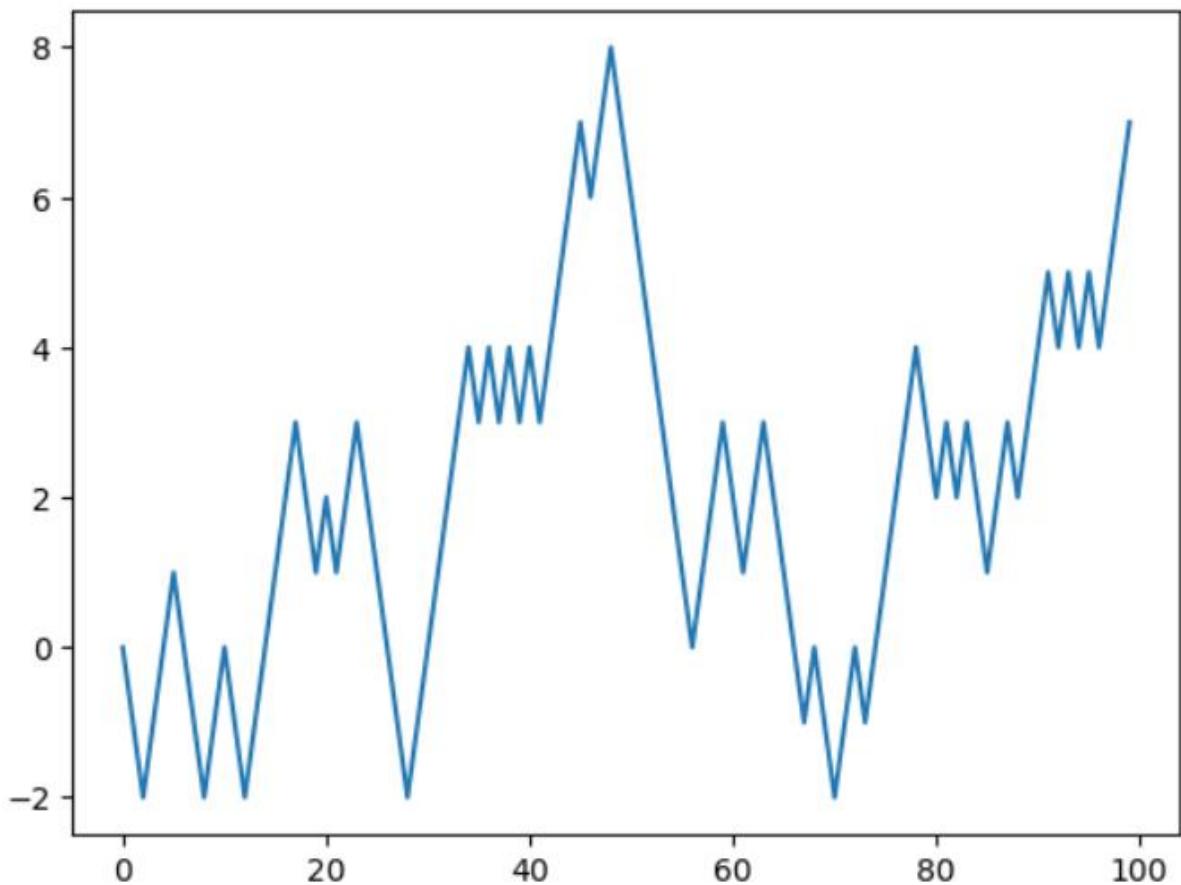
## Örnek : Random Walk

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

- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help.
- Toolbar icons:** Save, New, Cell, Run, Kernel, Cell Type, Cell Kernel, Help.
- In [133]:** A code cell containing Python code to generate a random walk. The code imports random, initializes position to 0, creates a walk list, sets steps to 1000, and then iterates through the range of steps, randomly choosing a step of 1 or -1 and appending the new position to the walk list.

```
import random
position = 0
walk = [position]
steps = 1000
for i in range(steps):
    step = 1 if random.randint(0, 1) else -1
    position += step
    walk.append(position)
```

- In [134]:** A code cell containing plt.plot(walk[:100]).
- Out[134]:** The output of the plot command, indicating a Line2D object at memory address 0x23101717790.



```
In [135]: nsteps = 1000
```

```
In [136]: draws = np.random.randint(0, 2, size=nsteps)
```

```
In [137]: steps = np.where(draws > 0, 1, -1)
```

```
In [138]: walk = steps.cumsum()
```

```
In [139]: walk.min()
```

```
Out[139]: -9
```

```
In [140]: walk.max()
```

```
Out[140]: 60
```

```
In [141]: (np.abs(walk) >= 10).argmax()
```

```
Out[141]: 297
```

## Aynı Anda Birçok Random Walks Simüle Edilmesi

The screenshot shows a Jupyter Notebook interface with the following code cells:

```
In [1]: nwalks = 5000
In [4]: import numpy as np
In [5]: nsteps = 1000
In [6]: draws = np.random.randint(0, 2, size=(nwalks, nsteps)) #0 veya
In [7]: steps = np.where(draws > 0, 1, -1)
In [8]: walks = steps.cumsum(1)
In [9]: walks
```

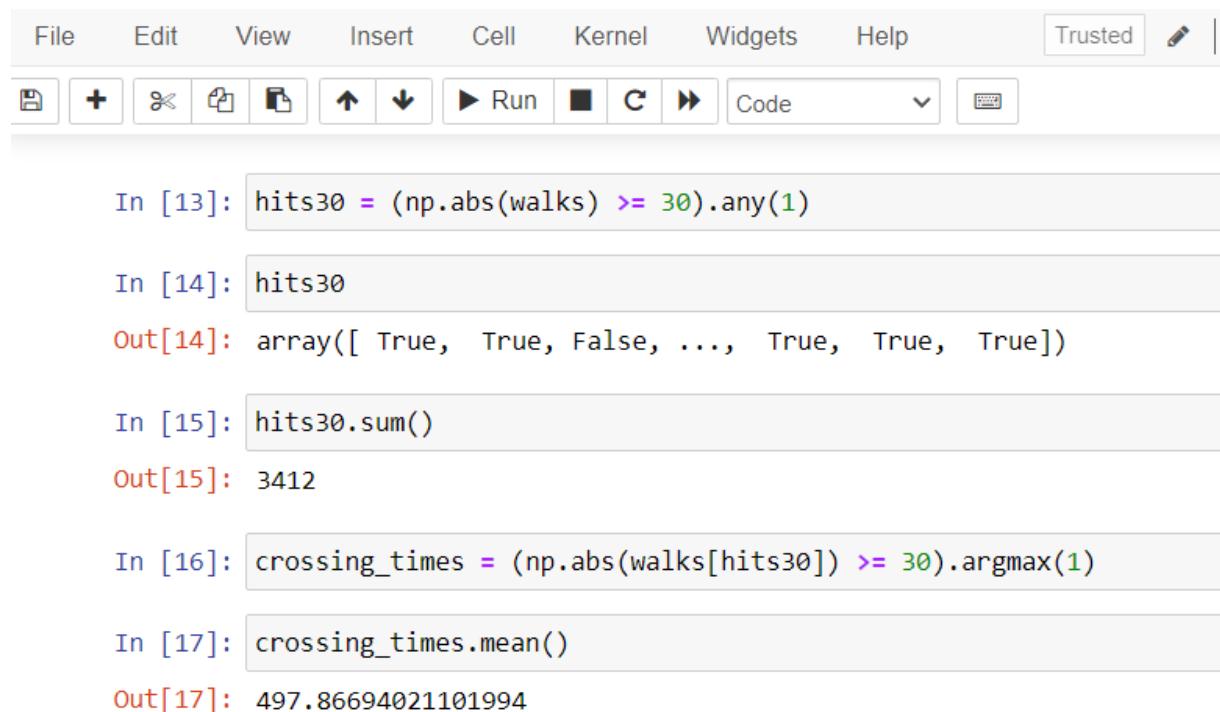
Out[9]:

```
array([[ -1,    0,    1, ..., -26, -25, -24],
       [  1,    2,    3, ..., -22, -21, -22],
       [ -1,   -2,   -3, ...,  -6,  -7,  -8],
       ...,
       [  1,    2,    1, ..., -32, -31, -30],
       [  1,    0,   -1, ..., -10,  -9,  -8],
       [ -1,    0,   -1, ..., -78, -79, -78]])
```

The screenshot shows a Jupyter Notebook interface with the following code cells:

```
In [10]: walks.max()
Out[10]: 144
In [11]: walks.min()
Out[11]: -123
```

- Bu yürüyüşlerden minimum geçiş süresini 30 veya -30 olarak hesaplayalım. Bu biraz zor çünkü 5.000 tanesinin hepsi 30'a ulaşmamıştır. Bunu herhangi birini kullanarak kontrol edebiliriz. Yöntem:



The screenshot shows a Jupyter Notebook interface with the following code execution history:

```

File Edit View Insert Cell Kernel Widgets Help Trusted | 
[File, New, Open, Save, Run, Kernel, Help, Trusted, Pencil]
In [13]: hits30 = (np.abs(walks) >= 30).any(1)
In [14]: hits30
Out[14]: array([ True,  True, False, ...,  True,  True,  True])
In [15]: hits30.sum()
Out[15]: 3412
In [16]: crossing_times = (np.abs(walks[hits30]) >= 30).argmax(1)
In [17]: crossing_times.mean()
Out[17]: 497.86694021101994

```

## | Pandas Başlangıç |

BT veri temizliği yapmak için tasarlanmış veri yapıları ve veri işleme araçlarını içerir. Python'da hızlı ve kolay analiz yapmaya yarar. Pandalar genellikle sayısal değerlerle birlikte kullanılır. NumPy ve SciPy gibi bilgi işlem araçları, statsmodels gibi analitik kütüphaneler ve scikit-learn ve matplotlib gibi veri görselleştirme kütüphaneleri vardır. NumPy'in dizi tabanlı hesaplamanın deyimsel stilinin parçaları, özellikle dizi tabanlı işlevler ve for döngüleri olmadan veri işleme tercihi vardır.

- Pandalar NumPy'dan birçok kodlama deyimini benimsese de en büyük fark şudur: pandalar tablo halinde veya heterojen verilerle çalışmak için tasarlanmıştır. NumPy, bağlı olarak trast, homojen sayısal dizi verileriyle çalışmak için en uygun yöntemdir.

- Pandas içe aktarma tanımı.

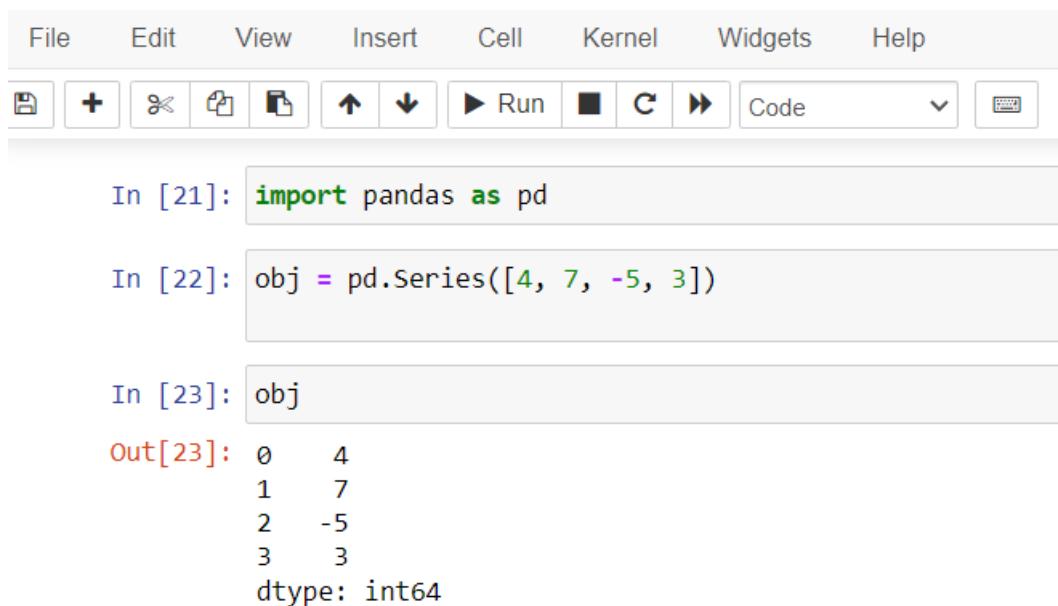
```
In [ ]: import pandas as pd
```

```
In [ ]: from pandas import Series, DataFrame
```

## Pandas Veri Yapılarına Giriş

### Seriler



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

- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and various file operations like Save, New, Open, etc.
- In [21]:** `import pandas as pd`
- In [22]:** `obj = pd.Series([4, 7, -5, 3])`
- In [23]:** `obj`
- Out[23]:**

0	4
1	7
2	-5
3	3
dtype: int64	

Veriler için bir indeks belirtmediğimiz için varsayılan olarak 0'dan N - 1'e kadar olan tamsayılardan oluşur (burada N, uzunluguđudur), veri oluşturulur.

```
In [24]: obj.values
```

```
Out[24]: array([ 4,  7, -5,  3], dtype=int64)
```

```
In [25]: obj.index
```

```
Out[25]: RangeIndex(start=0, stop=4, step=1)
```

File Edit View Insert Cell Kernel Widgets Help Trusted |

Run Cell Code

```
In [26]: obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
```

```
In [27]: obj2
```

```
Out[27]: d    4  
         b    7  
         a   -5  
         c    3  
        dtype: int64
```

```
In [28]: obj2.index
```

```
Out[28]: Index(['d', 'b', 'a', 'c'], dtype='object')
```

```
In [29]: obj2[['c', 'a', 'd']]
```

```
Out[29]: c    3  
         a   -5  
         d    4  
        dtype: int64
```

```
In [30]: obj2[obj2 > 0]
```

```
Out[30]: d    4  
         b    7  
         c    3  
        dtype: int64
```

```
In [31]: obj * 2
```

```
Out[31]: 0     8  
1    14  
2   -10  
3     6  
        dtype: int64
```

```
In [32]: np.exp(obj2)
```

```
Out[32]: d      54.598150  
b    1096.633158  
a      0.006738  
c    20.085537  
        dtype: float64
```

```
In [33]: 'b' in obj2
```

```
Out[33]: True
```

- Boolean ile filtreleme gibi NumPy işlevlerini veya NumPy benzeri işlemleri kullanma dizi, skaler çarpma veya matematik işlevlerinin uygulanması dizin değerini koruyacaktır.

```
In [34]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}

In [35]: obj3 = pd.Series(sdata)

In [36]: obj3
Out[36]:
Ohio      35000
Texas     71000
Oregon    16000
Utah      5000
dtype: int64

In [37]: states = ['California', 'Ohio', 'Oregon', 'Texas']

In [38]: obj4 = pd.Series(sdata, index=states)

In [39]: obj4
Out[39]:
California      NaN
Ohio            35000.0
Oregon          16000.0
Texas           71000.0
dtype: float64
```

- Burada sdata'da bulunan üç değer uygun yerlere yerleştirildi, ancak 'California' için bir değer bulunamadı, NaN (sayı değil) olarak görünüyor, bu da eksik veya NA değerlerini işaretlemek için pandalara yerleştirildi.

```
In [45]: obj3 + obj4
out[45]:
California      NaN
Ohio            70000.0
Oregon          32000.0
Texas           142000.0
Utah            NaN
dtype: float64

In [47]: obj4.name = 'population'
In [48]: obj4.index.name = 'state'
In [49]: obj4
out[49]:
state
California      NaN
Ohio            35000.0
Oregon          16000.0
Texas           71000.0
Name: population, dtype: float64
```

- 'Utah' dahil edilmediğinden ortaya çıkan nesnenin dışında tutulduğunu belirtir. Eksik verilere atıfta bulunmak için "eksik" veya "NA" terimlerini birbirinin yerine kullanılır. Eksik verileri tespit etmek için pandasdaki isnull ve notnull işlevleri kullanılmalıdır.

```
In [49]: obj4
```

```
Out[49]: state
California      NaN
Ohio            35000.0
Oregon           16000.0
Texas            71000.0
Name: population, dtype: float64
```

```
In [50]: obj
```

```
Out[50]: 0    4
1    7
2   -5
3    3
dtype: int64
```

```
In [53]: obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']
```

```
In [54]: obj
```

```
Out[54]: Bob    4
Steve   7
Jeff   -5
Ryan    3
dtype: int64
```

## DataFrame

DataFrame dikdörtgen bir veri tablosunu temsil eder ve sıralı bir koleksiyon içerir. Her biri farklı bir değer tipi (sayısal, dize, boolean vb.). DataFrame'in hem satır hem de sütun indeksi vardır; düşünülebilir Serilerin bir dict'i olarak hepsi aynı dizini paylaşıyor. Veriler, başlık altında tek bir dosya veya bir liste olarak depolanıyor.

```
In [55]: data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
              'year': [2000, 2001, 2002, 2001, 2002, 2003],
              'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
frame = pd.DataFrame(data)
```

```
In [56]: frame
```

```
Out[56]:
```

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

```
In [57]: frame.head()
```

- Head yöntemi, sadece beş satırı seçer.

```
Out[57]:
```

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9

```
In [58]: pd.DataFrame(data, columns=['year', 'state', 'pop'])
```

```
Out[58]:
```

	year	state	pop
0	2000	Ohio	1.5
1	2001	Ohio	1.7
2	2002	Ohio	3.6
3	2001	Nevada	2.4
4	2002	Nevada	2.9
5	2003	Nevada	3.2

```
In [61]: frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
                                index=['one', 'two', 'three', 'four',
                                       'five', 'six'])
```

```
In [62]: frame2.columns
```

```
Out[62]: Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

```
In [63]: frame2.year
```

```
Out[63]: one      2000
          two      2001
          three     2002
          four      2001
          five      2002
          six      2003
Name: year, dtype: int64
```

```
In [64]: frame2.loc['three']
```

```
Out[64]: year      2002
          state    Ohio
          pop       3.6
          debt      NaN
Name: three, dtype: object
```

```
In [65]: frame2['debt'] = 16.5
```

```
In [66]: frame2
```

```
Out[66]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	16.5
two	2001	Ohio	1.7	16.5
three	2002	Ohio	3.6	16.5
four	2001	Nevada	2.4	16.5
five	2002	Nevada	2.9	16.5
six	2003	Nevada	3.2	16.5

```
In [67]: frame2['debt'] = np.arange(6.)
```

```
In [68]: frame2
```

```
Out[68]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	0.0
two	2001	Ohio	1.7	1.0
three	2002	Ohio	3.6	2.0
four	2001	Nevada	2.4	3.0
five	2002	Nevada	2.9	4.0
six	2003	Nevada	3.2	5.0

- Sütunlar atamaya göre değiştirilebilir. Örneğin boş 'debt' sütunu skaler bir değere veya bir değerler dizisine atanabilir.

```
In [69]: frame2['eastern'] = frame2.state =='Ohio'
```

```
In [70]: frame2
```

out[70]:

	year	state	pop	debt	eastern
<b>one</b>	2000	Ohio	1.5	0.0	True
<b>two</b>	2001	Ohio	1.7	1.0	True
<b>three</b>	2002	Ohio	3.6	2.0	True
<b>four</b>	2001	Nevada	2.4	3.0	False
<b>five</b>	2002	Nevada	2.9	4.0	False
<b>six</b>	2003	Nevada	3.2	5.0	False

```
In [71]: del frame2['eastern']
```

```
In [72]: frame2.columns
```

out[72]: Index(['year', 'state', 'pop', 'debt'], dtype='object')

```
In [69]: frame2['eastern'] = frame2.state =='Ohio'
```

```
In [70]: frame2
```

Out[70]:

	year	state	pop	debt	eastern
<b>one</b>	2000	Ohio	1.5	0.0	True
<b>two</b>	2001	Ohio	1.7	1.0	True
<b>three</b>	2002	Ohio	3.6	2.0	True
<b>four</b>	2001	Nevada	2.4	3.0	False
<b>five</b>	2002	Nevada	2.9	4.0	False
<b>six</b>	2003	Nevada	3.2	5.0	False

```
In [71]: del frame2['eastern']
```

```
In [72]: frame2.columns
```

out[72]: Index(['year', 'state', 'pop', 'debt'], dtype='object')

- Oluşturulan sütunlar del yöntemiyle silinir.

```
In [73]: pop = {'Nevada': {2001: 2.4, 2002: 2.9},  
              'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
```

```
In [74]: frame3 = pd.DataFrame(pop)
```

```
In [75]: frame3
```

```
Out[75]:
```

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

```
In [76]: frame3.T
```

```
Out[76]:
```

	2001	2002	2000
Nevada	2.4	2.9	NaN
Ohio	1.7	3.6	1.5

- Dizilerin Diktleri hemen hemen aynı şekilde ele alınır:

```
In [77]: pdata = {'Ohio': frame3['Ohio'][:-1],  
                 'Nevada': frame3['Nevada'][:2]}
```

```
In [78]: pd.DataFrame(pdata)
```

```
Out[78]:
```

	Ohio	Nevada
2001	1.7	2.4
2002	3.6	2.9

## DataFrame Yapıcısına Olası Veri Girişleri

Type	Notes
2D ndarray	A matrix of data, passing optional row and column labels
dict of arrays, lists, or tuples	Each sequence becomes a column in the DataFrame; all sequences must be the same length
NumPy structured/record array	Treated as the "dict of arrays" case
dict of Series	Each value becomes a column; indexes from each Series are unioned together to form the result's row index if no explicit index is passed
dict of dicts	Each inner dict becomes a column; keys are unioned to form the row index as in the "dict of Series" case
List of dicts or Series	Each item becomes a row in the DataFrame; union of dict keys or Series indexes become the DataFrame's column labels
List of lists or tuples	Treated as the "2D ndarray" case
Another DataFrame	The DataFrame's indexes are used unless different ones are passed
NumPy MaskedArray	Like the "2D ndarray" case except masked values become NA/missing in the DataFrame result

## Index Nesneleri

```
In [79]: obj = pd.Series(range(3), index=['a', 'b', 'c'])
```

```
In [80]: index = obj.index
```

```
In [81]: index
```

```
Out[81]: Index(['a', 'b', 'c'], dtype='object')
```

```
In [82]: index[1:]
```

```
Out[82]: Index(['b', 'c'], dtype='object')
```

```
In [85]: index[1] = 'd'
```

```
-----  
TypeError  
Cell In[85], line 1  
----> 1 index[1] = 'd'
```

```
Traceback (most recent call last)
```

```
File ~\templateengine\Lib\site-packages\pandas\core\indexes\base.py:5157, in  
Index.__setitem__(self, key, value)  
    5155 @final  
    5156 def __setitem__(self, key, value):  
-> 5157     raise TypeError("Index does not support mutable operations")
```

```
TypeError: Index does not support mutable operations
```

```
In [86]: labels = pd.Index(np.arange(3))

In [87]: labels
Out[87]: Index([0, 1, 2], dtype='int32')

In [88]: obj2 = pd.Series([1.5, -2.5, 0], index=labels)

In [89]: obj2
Out[89]:
0    1.5
1   -2.5
2    0.0
dtype: float64

In [90]: obj2.index is labels
Out[90]: True

In [91]: dup_labels = pd.Index(['foo', 'foo', 'bar', 'bar'])

In [92]: dup_labels
Out[92]: Index(['foo', 'foo', 'bar', 'bar'], dtype='object')
```

## Bazı Index Yöntemleri ve Özellikleri

Method	Description
append	Concatenate with additional Index objects, producing a new Index
difference	Compute set difference as an Index
intersection	Compute set intersection
union	Compute set union
isin	Compute boolean array indicating whether each value is contained in the passed collection
delete	Compute new Index with element at index i deleted
drop	Compute new Index by deleting passed values
insert	Compute new Index by inserting element at index i
is_monotonic	Returns True if each element is greater than or equal to the previous element
is_unique	Returns True if the Index has no duplicate values
unique	Compute the array of unique values in the Index

## Temel Fonksiyonlar

### Yeniden Indexleme

- Pandas nesnelerinde önemli bir yöntem, yeni bir nesne oluşturmak anlamına gelen yeniden indekslemedir.

The screenshot shows a Jupyter Notebook interface with the following code execution history:

```
In [93]: obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])
```

```
In [94]: obj
Out[94]: d    4.5
          b    7.2
          a   -5.3
          c    3.6
          dtype: float64
```

```
In [95]: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
In [96]: obj2
Out[96]: a    -5.3
          b    7.2
          c    3.6
          d    4.5
          e    NaN
          dtype: float64
```

```
In [97]: obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])
In [98]: obj3
Out[98]: 0      blue
          2    purple
          4    yellow
          dtype: object
```

```
In [99]: obj3.reindex(range(6), method='ffill')
Out[99]: 0      blue
          1      blue
          2    purple
          3    purple
          4    yellow
          5    yellow
          dtype: object
```

```
In [100]: frame = pd.DataFrame(np.arange(9).reshape((3, 3)),  
                           index=['a', 'c', 'd'],  
                           columns=['Ohio', 'Texas', 'California'])
```

```
In [101]: frame
```

```
Out[101]:
```

	Ohio	Texas	California
a	0	1	2
c	3	4	5
d	6	7	8

```
In [102]: frame2 = frame.reindex(['a', 'b', 'c', 'd'])
```

```
In [103]: frame2
```

```
Out[103]:
```

	Ohio	Texas	California
a	0.0	1.0	2.0
b	NaN	NaN	NaN
c	3.0	4.0	5.0
d	6.0	7.0	8.0

## Yeniden Indexleme Fonksiyon Argümanları

Argument	Description
index	New sequence to use as index. Can be Index instance or any other sequence-like Python data structure. An Index will be used exactly as is without any copying.
method	Interpolation (fill) method; 'ffill' fills forward, while 'bfill' fills backward.
fill_value	Substitute value to use when introducing missing data by reindexing.
limit	When forward- or backfilling, maximum size gap (in number of elements) to fill.
tolerance	When forward- or backfilling, maximum size gap (in absolute numeric distance) to fill for inexact matches.
level	Match simple Index on level of MultiIndex; otherwise select subset of.
copy	If True, always copy underlying data even if new index is equivalent to old index; if False, do not copy the data when the indexes are equivalent.

## Bir Eksenden Girişleri Düşürme

```
File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3
[File, New, Open, Save, Run, Stop, Kernel, Help, Trusted, Python 3]
In [119]: obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
In [120]: obj
Out[120]:
a    0.0
b    1.0
c    2.0
d    3.0
e    4.0
dtype: float64

In [121]: new_obj = obj.drop('c')
In [122]: new_obj
Out[122]:
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64

In [114]: data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                           index=['Ohio', 'Colorado', 'Utah', 'New York'],
                           columns=['one', 'two', 'three', 'four'])
In [115]: data
Out[115]:
   one  two  three  four
Ohio     0     1     2     3
Colorado  4     5     6     7
Utah     8     9    10    11
New York 12    13    14    15

In [117]: data.drop(['Colorado', 'Ohio'])
Out[117]:
   one  two  three  four
Utah     8     9    10    11
New York 12    13    14    15
```

```
In [118]: data.drop(['two', 'four'], axis='columns')
```

```
Out[118]:
```

	one	three
Ohio	0	2
Colorado	4	6
Utah	8	10
New York	12	14

```
In [123]: data.drop(['two', 'four'], axis='columns')
```

```
Out[123]:
```

	one	three
Ohio	0	2
Colorado	4	6
Utah	8	10
New York	12	14

```
In [124]: obj.drop('c', inplace=True)
```

## Indexleme, Seçme ve Filtreleme

```
In [125]: obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
```

```
In [126]: obj
```

```
Out[126]: a    0.0
          b    1.0
          c    2.0
          d    3.0
          dtype: float64
```

```
In [127]: obj['b']
```

```
Out[127]: 1.0
```

```
In [7]: obj[2:4]
```

```
Out[7]: c    2.0
         d    3.0
        dtype: float64
```

```
In [8]: obj[['b', 'a', 'd']]
```

```
Out[8]: b    1.0
         a    0.0
         d    3.0
        dtype: float64
```

```
In [9]: obj['b':'c']
```

```
Out[9]: b    1.0
         c    2.0
        dtype: float64
```

```
In [12]: obj['b':'c'] = 5
```

```
In [13]: obj
```

```
Out[13]: a    0.0
         b    5.0
         c    5.0
         d    3.0
        dtype: float64
```

```
In [14]: data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                           index=['Ohio', 'Colorado', 'Utah', 'New York'],
                           columns=['one', 'two', 'three', 'four'])
```

```
In [15]: data
```

```
Out[15]:
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
In [16]: data['two']
```

```
Out[16]: Ohio      1
          Colorado   5
          Utah      9
          New York  13
          Name: two, dtype: int32
```

```
In [17]: data[['three', 'one']]
```

Out[17]:

	three	one
Ohio	2	0
Colorado	6	4
Utah	10	8
New York	14	12

```
In [19]: data[:2]
```

Out[19]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7

```
In [20]: data[data['three'] > 5]
```

Out[20]:

	one	two	three	four
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
In [21]: data < 5
```

Out[21]:

	one	two	three	four
Ohio	True	True	True	True
Colorado	True	False	False	False
Utah	False	False	False	False
New York	False	False	False	False

```
In [22]: data[data < 5] = 0
```

```
In [23]: data
```

Out[23]:

	one	two	three	four
Ohio	0	0	0	0
Colorado	0	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

- Dilimleme işlemleri bu şekilde olabilir. Dizinin bir kısmını elde etmek için ‘:’ operatörü kullanılır. Ve istenilen karşılaştırmalar ilgili operatörlerle yapılır.

## Loc ve Iloc ile Seçme

```
In [24]: data.loc['Colorado', ['two', 'three']]
```

```
Out[24]: two    5
          three   6
          Name: Colorado, dtype: int32
```

```
In [25]: data.iloc[2, [3, 0, 1]]
```

```
Out[25]: four    11
          one     8
          two     9
          Name: Utah, dtype: int32
```

```
In [26]: data.iloc[2, [3, 0, 1]]
```

```
Out[26]: four    11
          one     8
          two     9
          Name: Utah, dtype: int32
```

- Loc ve Iloc ile dizinin istenilen bölümleri seçilip işlem yaptırılabilir.

```
In [33]: data.loc[:'Utah', 'two']
```

```
Out[33]: Ohio      1
          Colorado  5
          Utah     9
          Name: two, dtype: int32
```

```
In [34]: data.iloc[:, :3][data.three > 5]
```

```
Out[34]:
```

	one	two	three
Colorado	4	5	6
Utah	8	9	10
New York	12	13	14

## DataFrame ile Indexleme Seçenekleri

Type	Notes
<code>df[val]</code>	Select single column or sequence of columns from the DataFrame; special case conveniences: boolean array (filter rows), slice (slice rows), or boolean DataFrame (set values based on some criterion)
<code>df.loc[val]</code>	Selects single row or subset of rows from the DataFrame by label
<code>df.loc[:, val]</code>	Selects single column or subset of columns by label
<code>df.loc[val1, val2]</code>	Select both rows and columns by label
<code>df.iloc[where]</code>	Selects single row or subset of rows from the DataFrame by integer position
<hr/>	
<code>df.iloc[:, where]</code>	Selects single column or subset of columns by integer position
<code>df.iloc[where_i, where_j]</code>	Select both rows and columns by integer position
<code>df.at[label_i, label_j]</code>	Select a single scalar value by row and column label
<code>df.iat[i, j]</code>	Select a single scalar value by row and column position (integers)
<code>reindex method</code>	Select either rows or columns by labels
<code>get_value, set_value methods</code>	Select single value by row and column label

## Integer Indexler

```
In [39]: ser2 = pd.Series(np.arange(3.), index=['a', 'b', 'c'])

In [40]: ser2[-1]
Out[40]: 2.0

In [41]: ser[:1]
Out[41]: 0    0.0
          dtype: float64

In [42]: ser.loc[:1]
Out[42]: 0    0.0
          1    1.0
          dtype: float64

In [43]: ser.iloc[:1]
Out[43]: 0    0.0
          dtype: float64
```

## Aritmetik ve Veri Hızalama

The screenshot shows a Jupyter Notebook interface with the following code execution history:

- In [44]: `s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])`
- In [45]: `s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1], index=['a', 'c', 'e', 'f', 'g'])`
- In [46]: `s1`  
Out[46]:  
a 7.3  
c -2.5  
d 3.4  
e 1.5  
dtype: float64
- In [47]: `s2`  
Out[47]:  
a -2.1  
c 3.6  
e -1.5  
f 4.0  
g 3.1  
dtype: float64
- In [49]: `s1 + s2`  
Out[49]:  
a 5.2  
c 1.1  
d NaN  
e 0.0  
f NaN  
g NaN  
dtype: float64

- Burada aynı satır ve sütunlar mevcutsa işlem gerçekleştirilir yoksa NaN değer döndürülür.
- Ortak sütun veya satır etiketleri olmayan DataFrame nesneleri eklersek sonuç tüm boş değerleri içerir.

## Doldurma Değerleri ile Aritmetik Yöntemler

```
In [55]: df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),  
columns=list('abcd'))
```

```
In [56]: df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),  
columns=list('abcde'))
```

```
In [57]: df2.loc[1, 'b'] = np.nan
```

```
In [58]: df1
```

```
Out[58]:
```

	a	b	c	d
0	0.0	1.0	2.0	3.0
1	4.0	5.0	6.0	7.0
2	8.0	9.0	10.0	11.0

```
In [59]: df2
```

```
Out[59]:
```

	a	b	c	d	e
0	0.0	1.0	2.0	3.0	4.0
1	5.0	NaN	7.0	8.0	9.0
2	10.0	11.0	12.0	13.0	14.0
3	15.0	16.0	17.0	18.0	19.0

```
In [60]: df1 + df2
```

```
Out[60]:
```

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	NaN
1	9.0	NaN	13.0	15.0	NaN
2	18.0	20.0	22.0	24.0	NaN
3	NaN	NaN	NaN	NaN	NaN

```
In [61]: df1.add(df2, fill_value=0)
```

```
Out[61]:
```

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	4.0
1	9.0	5.0	13.0	15.0	9.0
2	18.0	20.0	22.0	24.0	14.0
3	15.0	16.0	17.0	18.0	19.0

```
In [62]: 1 / df1
```

```
Out[62]:
```

	a	b	c	d
0	inf	1.000000	0.500000	0.333333
1	0.250	0.200000	0.166667	0.142857
2	0.125	0.111111	0.100000	0.090909

```
In [63]: df1.rdiv(1)
```

```
Out[63]:
```

	a	b	c	d
0	inf	1.000000	0.500000	0.333333
1	0.250	0.200000	0.166667	0.142857
2	0.125	0.111111	0.100000	0.090909

- DataFrame'ler tanımlanıp bazı aritmetik işlemler yapıldı. 'r' harfi ile ilgili DataFrame'in satır ve sütunlarının yer değiştirme işlemi yapıldı. 'r' harfi getirilen yerde ilgili argümanın ters çevrilme işlemi yapılır.

```
In [64]: df1.reindex(columns=df2.columns, fill_value=0)
```

```
Out[64]:
```

	a	b	c	d	e
0	0.0	1.0	2.0	3.0	0
1	4.0	5.0	6.0	7.0	0
2	8.0	9.0	10.0	11.0	0

## Esnek Aritmetik Yöntemler

Method	Description
add, radd	Methods for addition (+)
sub, rsub	Methods for subtraction (-)
div, rdiv	Methods for division (/)
floordiv, rfloordiv	Methods for floor division (//)
mul, rmul	Methods for multiplication (*)
pow, rpow	Methods for exponentiation (**)

## DataFrame ve Seriler Arasındaki İşlemler

```
In [65]: arr = np.arange(12.).reshape((3, 4))
```

```
In [66]: arr
```

```
Out[66]: array([[ 0.,  1.,  2.,  3.],
   [ 4.,  5.,  6.,  7.],
   [ 8.,  9., 10., 11.]])
```

```
In [67]: arr[0]
```

```
Out[67]: array([0., 1., 2., 3.])
```

```
In [68]: arr - arr[0]
```

```
Out[68]: array([[ 0.,  0.,  0.,  0.],
   [ 4.,  4.,  4.,  4.],
   [ 8.,  8.,  8.,  8.]])
```

```
In [69]: frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),
                           columns=list('bde'),
                           index=['Utah', 'Ohio', 'Texas', 'Oregon'])
```

```
In [70]: series = frame.iloc[0]
```

```
In [71]: frame
```

```
Out[71]:
```

	b	d	e
<b>Utah</b>	0.0	1.0	2.0
<b>Ohio</b>	3.0	4.0	5.0
<b>Texas</b>	6.0	7.0	8.0
<b>Oregon</b>	9.0	10.0	11.0

```
In [72]: series
```

```
Out[72]: b    0.0
d    1.0
e    2.0
Name: Utah, dtype: float64
```

```
In [73]: frame - series
```

```
Out[73]:
```

	b	d	e
<b>Utah</b>	0.0	0.0	0.0
<b>Ohio</b>	3.0	3.0	3.0
<b>Texas</b>	6.0	6.0	6.0
<b>Oregon</b>	9.0	9.0	9.0

- Varsayılan olarak DataFrame ile Seriler arasındaki aritmetik, Serinin indeksiyle eşleşir DataFrame'in sütunlarında, satırların aşağısına doğru işlemler yapılır.

```
In [74]: series3 = frame['d']

In [75]: series3
out[75]: Utah      1.0
          Ohio      4.0
          Texas     7.0
          Oregon   10.0
          Name: d, dtype: float64

In [76]: frame.sub(series3, axis='index')
out[76]:
```

	b	d	e
<b>Utah</b>	-1.0	0.0	1.0
<b>Ohio</b>	-1.0	0.0	1.0
<b>Texas</b>	-1.0	0.0	1.0
<b>Oregon</b>	-1.0	0.0	1.0

- İletilen eksen numarası, eşleşecek eksendir. Bu durumda eşleştirmeyi kastediyoruz DataFrame'in satır indeksinin (axis='index' veya axis=0) genelinde yayınlanır.

## Fonksiyon Uygulaması ve Haritalama

```
In [77]: frame = pd.DataFrame(np.random.randn(4, 3), columns=list('bde'),
                           index=['Utah', 'Ohio', 'Texas', 'Oregon'])
```

```
In [78]: frame
out[78]:
```

	b	d	e
<b>Utah</b>	-0.685112	0.240033	-0.639562
<b>Ohio</b>	1.492501	1.419799	-0.887344
<b>Texas</b>	0.972689	0.053020	-1.489385
<b>Oregon</b>	-0.340454	0.782531	-0.717631

```
In [79]: np.abs(frame)
```

```
Out[79]:
```

	b	d	e
Utah	0.685112	0.240033	0.639562
Ohio	1.492501	1.419799	0.887344
Texas	0.972689	0.053020	1.489385
Oregon	0.340454	0.782531	0.717631

```
In [80]: f = lambda x: x.max() - x.min()
```

```
In [81]: frame.apply(f)
```

```
Out[81]: b    2.177613  
         d    1.366779  
         e    0.849823  
        dtype: float64
```

- Buradaki fonksiyon her sütun için çağrılmış işlem yaptırılır.
- Satırlara uygulanmak için axis='columns' komutunu iletirseniz, işlev satır başına bir kez çağrılmacaktır.

```
In [82]: frame.apply(f, axis='columns')
```

```
Out[82]: Utah      0.925145  
          Ohio     2.379845  
          Texas    2.462074  
          Oregon   1.500162  
         dtype: float64
```

```
In [83]: def f(x):
```

```
    return pd.Series([x.min(), x.max()], index=['min', 'max'])
```

```
In [84]: frame.apply(f)
```

```
Out[84]:
```

	b	d	e
min	-0.685112	0.053020	-1.489385
max	1.492501	1.419799	-0.639562

- Öğe bazında Python işlevleri de kullanılabilir. Diyelim ki framedeki her float değerinden biçimlendirilmiş dize hesaplaması yapmak istediniz. Bunu ‘applymap’ ile uygulayabilirsiniz.

```
In [85]: format = lambda x: '%.2f' % x
```

```
In [86]: frame.applymap(format)
```

Out[86]:

	b	d	e
Utah	-0.69	0.24	-0.64
Ohio	1.49	1.42	-0.89
Texas	0.97	0.05	-1.49
Oregon	-0.34	0.78	-0.72

## Sort ve Rank

- Bir veri kümesini bazı ölçütlerde göre sıralamak başka bir önemli yerleşik işlemdir. Sözlük bilimsel olarak satır veya sütun indeksine göre sıralamak, sort\_index yöntemini kullanarak yapılabilir.

```
In [88]: obj = pd.Series(range(4), index=['d', 'a', 'b', 'c'])
```

```
In [89]: obj.sort_index()
```

```
Out[89]: a    1  
         b    2  
         c    3  
         d    0  
dtype: int64
```

```
In [93]: frame = pd.DataFrame(np.arange(8).reshape((2, 4)),  
                           index=['three', 'one'],  
                           columns=['d', 'a', 'b', 'c'])
```

```
In [94]: frame.sort_index()
```

Out[94]:

	d	a	b	c
one	4	5	6	7
three	0	1	2	3

- DataFrame ile her iki eksende de dizideye göre sıralama yapılabilir.

```
In [95]: frame.sort_index(axis=1)
```

Out[95]:

	a	b	c	d
three	1	2	3	0
one	5	6	7	4

```
In [96]: frame.sort_index(axis=1, ascending=False)
```

```
Out[96]:
```

	d	c	b	a
three	0	3	2	1
one	4	7	6	5

```
In [97]: obj = pd.Series([4, 7, -3, 2])
```

```
In [98]: obj.sort_values()
```

```
Out[98]: 2    -3  
3     2  
0     4  
1     7  
dtype: int64
```

- Bir Seriyi değerlerine göre sıralamak için ‘sort\_values’ yöntemi kullanılabilir.

```
In [99]: obj = pd.Series([4, np.nan, 7, np.nan, -3, 2])
```

```
In [100]: obj.sort_values()
```

```
Out[100]: 4    -3.0  
5     2.0  
0     4.0  
2     7.0  
1     NaN  
3     NaN  
dtype: float64
```

```
In [101]: frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})
```

```
In [102]: frame
```

```
Out[102]:
```

	b	a
0	4	0
1	7	1
2	-3	0
3	2	1

```
In [104]: obj.rank()
```

```
Out[104]: 0    6.5
1    1.0
2    6.5
3    4.5
4    3.0
5    2.0
6    4.5
dtype: float64
```

```
In [105]: obj.rank(method='first')
```

```
Out[105]: 0    6.0
1    1.0
2    7.0
3    4.0
4    3.0
5    2.0
6    5.0
dtype: float64
```

```
In [106]: obj.rank(ascending=False, method='max')
```

```
Out[106]: 0    2.0
1    7.0
2    2.0
3    4.0
4    5.0
5    6.0
6    4.0
dtype: float64
```

```
In [107]: frame = pd.DataFrame({'b': [4.3, 7, -3, 2], 'a': [0, 1, 0, 1],
                                'c': [-2, 5, 8, -2.5]})
```

```
In [108]: frame
```

```
Out[108]:
      b   a   c
0  4.3  0 -2.0
1  7.0  1  5.0
2 -3.0  0  8.0
3  2.0  1 -2.5
```

```
In [110]: frame.rank(axis='columns')
```

```
Out[110]:
      b   a   c
0  3.0  2.0  1.0
1  3.0  1.0  2.0
2  1.0  2.0  3.0
3  3.0  2.0  1.0
```

## Sıralamayla beraberliği bozma yöntemleri

Method	Description
'average'	Default: assign the average rank to each entry in the equal group
'min'	Use the minimum rank for the whole group
'max'	Use the maximum rank for the whole group
'first'	Assign ranks in the order the values appear in the data
'dense'	Like <code>method='min'</code> , but ranks always increase by 1 in between groups rather than the number of equal elements in a group

## Yinelenen Etiketlerle Eksen Indexleri

```
In [111]: obj = pd.Series(range(5), index=['a', 'a', 'b', 'b', 'c'])
```

```
In [112]: obj
```

```
Out[112]: a    0  
          a    1  
          b    2  
          b    3  
          c    4  
         dtype: int64
```

```
In [113]: obj.index.is_unique
```

```
Out[113]: False
```

- Dizinin `is_unique` özelliği, etiketlerinin benzersiz olup olmadığını size söyler.

```
In [114]: df = pd.DataFrame(np.random.randn(4, 3), index=['a', 'a', 'b', 'b'])
```

```
In [115]: df
```

```
Out[115]:
```

	0	1	2
a	2.186411	1.407012	-1.688555
a	-1.817483	0.138047	0.419834
b	-1.501669	0.350272	-0.120884
b	-1.313122	1.194529	0.603038

```
In [116]: df.loc['b']
```

```
Out[116]:
```

	0	1	2
b	-1.501669	0.350272	-0.120884
b	-1.313122	1.194529	0.603038

## Tanımlayıcı İstatistiklerin Özeti ve Hesaplanması

- Pandas nesneleri bir dizi ortak matematiksel ve istatistiksel yöntemle donatılmıştır. Bir Seriden veya bir değer Serisinden tek bir değer (toplam veya ortalama gibi) çıkarılan DataFrame'in satırlarından veya sütunlarından azaltmalar veya özet istatistikler, yöntemler kategorisine girmektedir. Benzer yöntemlerle karşılaşıldığında NumPy dizilerinde bulunan bu diziler, eksik veriler için yerleşik işleme sahiptir.

```
In [117]: df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5],
                           [np.nan, np.nan], [0.75, -1.3]],
                           index=['a', 'b', 'c', 'd'],
                           columns=['one', 'two'])
```

```
In [118]: df
```

```
Out[118]:
```

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3

```
In [119]: df.sum()
```

```
Out[119]: one    9.25
           two   -5.80
           dtype: float64
```

```
In [120]: df.sum(axis='columns')
```

```
Out[120]: a    1.40
           b    2.60
           c    0.00
           d   -0.55
           dtype: float64
```

## Her Azaltma Yöntemine İlişkin Ortak Seçeneklerin Listesi

Method	Description
axis	Axis to reduce over; 0 for DataFrame's rows and 1 for columns
skipna	Exclude missing values; True by default
level	Reduce grouped by level if the axis is hierarchically indexed (MultiIndex)

```
In [122]: df.idxmax()
```

```
Out[122]: one    b  
          two    d  
          dtype: object
```

```
In [123]: df.cumsum()
```

```
Out[123]:
```

	one	two
a	1.40	NaN
b	8.50	-4.5
c	NaN	NaN
d	9.25	-5.8

```
In [124]: df.describe()
```

```
Out[124]:
```

	one	two
count	3.000000	2.000000
mean	3.083333	-2.900000
std	3.493685	2.262742
min	0.750000	-4.500000
25%	1.075000	-3.700000
50%	1.400000	-2.900000
75%	4.250000	-2.100000
max	7.100000	-1.300000

- Bir diğer yöntem ise ne azaltma ne de biriktirme yöntemidir. Tarif bir tane örneğin, tek seferde birden fazla özet istatistik üretmek. ‘describe’ ile oluşturulur.
- Sayısal olmayan verilerde, açıklama olarak alternatif özet istatistikler üretir.

```
In [125]: obj = pd.Series(['a', 'a', 'b', 'c'] * 4)
```

```
In [126]: obj.describe()
```

```
Out[126]: count      16  
unique       3  
top         a  
freq        8  
dtype: object
```

## Tanımlayıcı ve Özeti İstatistikler

Method	Description
count	Number of non-NA values
describe	Compute set of summary statistics for Series or each DataFrame column
min, max	Compute minimum and maximum values
argmin, argmax	Compute index locations (integers) at which minimum or maximum value obtained, respectively
idxmin, idxmax	Compute index labels at which minimum or maximum value obtained, respectively
quantile	Compute sample quantile ranging from 0 to 1
sum	Sum of values
mean	Mean of values
median	Arithmetic median (50% quantile) of values
mad	Mean absolute deviation from mean value
prod	Product of all values
var	Sample variance of values
std	Sample standard deviation of values
skew	Sample skewness (third moment) of values
kurt	Sample kurtosis (fourth moment) of values
cumsum	Cumulative sum of values
cummin, cummax	Cumulative minimum or maximum of values, respectively
cumprod	Cumulative product of values
diff	Compute first arithmetic difference (useful for time series)
pct_change	Compute percent changes

## Korelasyon ve Kovaryans

```
In [ ]: import pandas_datareader.data as web  
all_data = {ticker: web.get_data_yahoo(ticker)  
           for ticker in ['AAPL', 'IBM', 'MSFT', 'GOOG']}
```

```
In [ ]: price = pd.DataFrame({ticker: data['Adj Close']  
                           for ticker, data in all_data.items()})  
volume = pd.DataFrame({ticker: data['Volume']  
                       for ticker, data in all_data.items()})
```

```
In [ ]: returns = price.pct_change()
```

```
In [12]: returns.tail()
```

Date	AAPL	GOOG	IBM	MSFT
2016-10-17	-0.000680	0.001837	0.002072	-0.003483
2016-10-18	-0.000681	0.019616	-0.026168	0.007690
2016-10-19	-0.002979	0.007846	0.003583	-0.002255
2016-10-20	-0.000512	-0.005652	0.001719	-0.004867
2016-10-21	-0.003930	0.003011	-0.012474	0.042096

```
In [244]: returns['MSFT'].corr(returns['IBM'])
Out[244]: 0.49976361144151144
```

```
In [245]: returns['MSFT'].cov(returns['IBM'])
Out[245]: 8.8706554797035462e-05
```

```
In [247]: returns.corr()
Out[247]:
          AAPL      GOOG       IBM      MSFT
AAPL  1.000000  0.407919  0.386817  0.389695
GOOG  0.407919  1.000000  0.405099  0.465919
IBM   0.386817  0.405099  1.000000  0.499764
MSFT  0.389695  0.465919  0.499764  1.000000
```

```
In [248]: returns.cov()
Out[248]:
          AAPL      GOOG       IBM      MSFT
AAPL  0.000277  0.000107  0.000078  0.000095
GOOG  0.000107  0.000251  0.000078  0.000108
IBM   0.000078  0.000078  0.000146  0.000089
MSFT  0.000095  0.000108  0.000089  0.000215
```

## Kovaryans

- İki değişkenin birlikte değişimlerinin ölçüsü kovaryans olarak bilinir.

## Korelasyon

- Korelasyon katsayıları iki değişken arasındaki doğrusal ilişkinin derecesini belirleyen ve karşılaştırmaya olanak veren bir katsayıdır.

Burada yahoo'dan çekilen veriler `pandas_datareader` ile okunarak kovaryans ve korelasyon bulma çalışmaları yapılmıştır. İstenilen index için ayrı çalışmalar da yapılabilir.

## Unique Değerler, Değer Sayıları ve Membership

```
File Edit View Insert Cell Kernel Widgets Help Trusted |  
[ ] + × ⌂ ⌄ ⌅ ⌆ ⌇ Run C ⌈ ⌉ Code ⌄ ⌅  
  
In [13]: obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])  
  
In [14]: uniques = obj.unique()  
  
In [15]: uniques  
  
Out[15]: array(['c', 'a', 'd', 'b'], dtype=object)
```

- Benzersiz değerlerin mutlaka sıralı olarak döndürülmesi gerekmektedir, ancak ihtiyaç duyulduğundan sonra “(uniques.sort())” ile sıralanabilir. “value\_counts” ise bir seriyi hesaplar.

```
In [16]: obj.value_counts()  
  
Out[16]: c    3  
         a    3  
         b    2  
         d    1  
Name: count, dtype: int64  
  
In [17]: pd.value_counts(obj.values, sort=False)  
  
Out[17]: c    3  
         a    3  
         d    1  
         b    2  
Name: count, dtype: int64
```

- Kolaylık sağlamak amacıyla seriler değere göre azalan düzende sıralanmıştır. value\_counts ayrıca herhangi bir dizile kullanılabilecek üst düzey bir panda yöntemi veya sekans olarak da mevcuttur.

- `isin`, vektörleştirilmiş bir küme üyesi kontrolü gerçekleştirir ve bir kümenin filtrelenmesinde yararlı olabilir

```
Out[19]: 0    c  
          1    a  
          2    d  
          3    a  
          4    a  
          5    b  
          6    b  
          7    c  
          8    c  
         dtype: object
```

```
In [20]: mask = obj.isin(['b', 'c'])
```

```
In [21]: mask
```

```
Out[21]: 0    True  
          1   False  
          2   False  
          3   False  
          4   False  
          5    True  
          6    True  
          7    True  
          8    True  
         dtype: bool
```

```
In [22]: obj[mask]
```

```
Out[22]: 0    c  
          5    b  
          6    b  
          7    c  
          8    c
```

- “`isin`” ile ilgili olarak size bir indeks dizisi veren `Index.get_indexer` yöntemi vardır.

```
In [23]: to_match = pd.Series(['c', 'a', 'b', 'b', 'c', 'a'])
```

```
In [24]: unique_vals = pd.Series(['c', 'b', 'a'])
```

```
In [25]: pd.Index(unique_vals).get_indexer(to_match)
```

```
Out[25]: array([0, 2, 1, 1, 0, 2], dtype=int64)
```

## Unique Değerler, Değer Sayıları ve Membership Yöntemleri

Method	Description
isin	Compute boolean array indicating whether each Series value is contained in the passed sequence of values
match	Compute integer indices for each value in an array into another array of distinct values; helpful for data alignment and join-type operations
unique	Compute array of unique values in a Series, returned in the order observed
value_counts	Return a Series containing unique values as its index and frequencies as its values, ordered count in descending order

- Burada sonuçtaki satır etiketleri, tüm sütunlarda oluşan farklı değerlerdir. Değerler, her sütundaki bu değerlerle ilgili sayılardır.

The screenshot shows a Jupyter Notebook interface with the following code execution history:

```
File Edit View Insert Cell Kernel Widgets Help  
In [26]: data = pd.DataFrame({'Qu1': [1, 3, 4, 3, 4],  
                           'Qu2': [2, 3, 1, 2, 3],  
                           'Qu3': [1, 5, 2, 4, 4]})  
  
In [27]: data  
Out[27]:  
   Qu1  Qu2  Qu3  
0    1    2    1  
1    3    3    5  
2    4    1    2  
3    3    2    4  
4    4    3    4  
  
In [28]: result = data.apply(pd.value_counts).fillna(0)  
  
In [29]: result  
Out[29]:  
   Qu1  Qu2  Qu3  
1    1.0  1.0  1.0  
2    0.0  2.0  1.0  
3    2.0  2.0  0.0  
4    2.0  0.0  2.0  
5    0.0  0.0  1.0
```

# Veri Yükleme, Depolama ve Dosya Formatları

- Verilere erişim, bu kitaptaki araçların çoğunu kullanmak için gerekli bir ilk adımdır. Pandalar kullanılarak veri girişi ve çıkışına odaklanılır, ancak bunun için çok sayıda seçenek var. Giriş ve çıkış genellikle birkaç ana kategoriye ayrılır: metin dosyalarının okunması ve diğer daha verimli disk formatları, veri tabanlarından veri yükleme ve ağ ile etkileşimde bulunma web API'leri gibi çalışma kaynakları.

## Verileri Metin Formatında Okumak ve Yazmak

### Pandalarda ayrıştırma işlevleri

Function	Description
<code>read_csv</code>	Load delimited data from a file, URL, or file-like object; use comma as default delimiter
<code>read_table</code>	Load delimited data from a file, URL, or file-like object; use tab (' \t ') as default delimiter
<code>read_fwf</code>	Read data in fixed-width column format (i.e., no delimiters)
<code>read_clipboard</code>	Version of <code>read_table</code> that reads data from the clipboard; useful for converting tables from web pages
<code>read_excel</code>	Read tabular data from an Excel XLS or XLSX file
<code>read_hdf</code>	Read HDF5 files written by pandas
<code>read_html</code>	Read all tables found in the given HTML document
<code>read_json</code>	Read data from a JSON (JavaScript Object Notation) string representation
<code>read_msgpack</code>	Read pandas data encoded using the MessagePack binary format
<code>read_pickle</code>	Read an arbitrary object stored in Python pickle format
<code>read_sas</code>	Read a SAS dataset stored in one of the SAS system's custom storage formats
<code>read_sql</code>	Read the results of a SQL query (using SQLAlchemy) as a pandas DataFrame
<code>read_stata</code>	Read a dataset from Stata file format
<code>read_feather</code>	Read the Feather binary file format

Dönüştürmeyi amaçlayan bu işlevlerin mekanigine genel bir bakış, verileri bir DataFrame'e yazıldıktan sonra bu işlevler için istege bağlı bağımsız değişkenler aşağıdakilere düşebilir.

Birkaç kategori:

- İndeksleme
- Tür çıkarımı ve veri dönüştürme
- Datetime Ayrıştırma
- İterasyon
- Temiz olmayan veri sorunları

## CSV Metin Dosyası

```
In [5]: import pandas as pd
```

```
In [8]: df = pd.read_csv('ex1.csv')
```

```
In [9]: df
```

```
Out[9]:
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

```
In [11]: pd.read_table('ex1.csv', sep=',')
```

```
Out[11]:
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

```
In [12]: pd.read_csv('ex2.csv', header=None)
```

```
Out[12]:
```

	0	1	2	3	4
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

```
In [14]: pd.read_csv('ex2.csv', names=['a', 'b', 'c', 'd', 'message'])
```

```
Out[14]:
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

- CSV dosyasını okuma işlemleri. Bir başlıkla oluşturmak zorunda değiliz istersek header'ına istediğimiz isimlendirmeyi yapabilir ya da kaldırabiliriz.

```
In [15]: names = ['a', 'b', 'c', 'd', 'message']
```

```
In [16]: pd.read_csv('ex2.csv', names=names, index_col='message')
```

Out[16]:

	a	b	c	d
message				
hello	1	2	3	4
world	5	6	7	8
foo	9	10	11	12

```
In [20]: parsed = pd.read_csv('csv_mindex.csv',
                           index_col=['key1', 'key2'])
```

```
In [21]: parsed
```

Out[21]:

		value1	value2
key1	key2		
one	a	1	2
	b	3	4
	c	5	6
	d	7	8
two	a	9	10
	b	11	12
	c	13	14
	d	15	16

In [17]: !cat examples/csv\_mindex.csv  
key1,key2,value1,value2  
one,a,1,2  
one,b,3,4  
one,c,5,6  
one,d,7,8  
two,a,9,10  
two,b,11,12  
two,c,13,14  
two,d,15,16

- Bazı durumlarda, bir tablonun boşluk veya başka bir karakter kullanılarak sabit bir sınırlayıcısı olmayabilir.

```
In [24]: list(open('ex3.txt'))
```

```
Out[24]: ['      A          B          C\n',
          'aaa -0.264438 -1.026059 -0.619500\n',
          'bbb  0.927272  0.302904 -0.032399\n',
          'ccc -0.264273 -0.386314 -0.217601\n',
          'ddd -0.871858 -0.348382  1.100491']
```

- Bu, \s+ normal ifadesi ile ifade edilebilir.

```
In [25]: result = pd.read_table('ex3.txt', sep='\s+')
```

```
In [26]: result
```

```
Out[26]:
```

	A	B	C
<b>aaa</b>	-0.264438	-1.026059	-0.619500
<b>bbb</b>	0.927272	0.302904	-0.032399
<b>ccc</b>	-0.264273	-0.386314	-0.217601
<b>ddd</b>	-0.871858	-0.348382	1.100491

- Skiprow ile atlama satırları özelliği kullanılır.

```
In [31]: pd.read_csv('ex4.csv', skiprows=[0, 2, 3])
```

```
Out[31]:
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

```
In [23]: !cat examples/ex4.csv
# hey!
a,b,c,d,message
# just wanted to make things more difficult for you
# who reads CSV files with computers, anyway?
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

```
In [33]: result = pd.read_csv('ex5.csv')
```

```
In [34]: result
```

```
Out[34]:
```

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

```
In [25]: !cat examples/ex5.csv
something,a,b,c,d,message
one,1,2,3,4,NA
two,5,6,,8,world
three,9,10,11,12,foo
```

```
In [35]: pd.isnull(result)
```

```
Out[35]:
```

	something	a	b	c	d	message
0		False	False	False	False	True
1		False	False	False	True	False
2		False	False	False	False	False

- Burada atana değerlerin NaN olanları ile ilgili uygulamalar yapılmıştır. Bazıları değiştirilmiş ve NaN olanların True gönderme işlemi yapılmıştır.

```
In [36]: result = pd.read_csv('ex5.csv', na_values=['NULL'])
```

```
In [37]: result
```

Out[37]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

```
In [38]: sentinel = {'message': ['foo', 'NA'], 'something': ['two']}
```

```
In [39]: pd.read_csv('ex5.csv', na_values=sentinel)
```

Out[39]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	NaN	5	6	NaN	8	world
2	three	9	10	11.0	12	NaN

## Bazı “read\_csv/read\_table” İşlev Argümanları

Argument	Description
path	String indicating filesystem location, URL, or file-like object
sep or delimiter	Character sequence or regular expression to use to split fields in each row
header	Row number to use as column names; defaults to 0 (first row), but should be None if there is no header row
index_col	Column numbers or names to use as the row index in the result; can be a single name/number or a list of them for a hierarchical index
names	List of column names for result, combine with header=None

<code>skiprows</code>	Number of rows at beginning of file to ignore or list of row numbers (starting from 0) to skip.
<code>na_values</code>	Sequence of values to replace with NA.
<code>comment</code>	Character(s) to split comments off the end of lines.
<code>parse_dates</code>	Attempt to parse data to <code>datetime</code> ; <code>False</code> by default. If <code>True</code> , will attempt to parse all columns. Otherwise can specify a list of column numbers or name to parse. If element of list is tuple or list, will combine multiple columns together and parse to date (e.g., if date/time split across two columns).
<code>keep_date_col</code>	If joining columns to parse date, keep the joined columns; <code>False</code> by default.
<code>converters</code>	Dict containing column number or name mapping to functions (e.g., <code>{'foo': f}</code> would apply the function <code>f</code> to all values in the ' <code>foo</code> ' column).
<code>dayfirst</code>	When parsing potentially ambiguous dates, treat as international format (e.g., <code>7/6/2012</code> -> June 7, 2012); <code>False</code> by default.
<code>date_parser</code>	Function to use to parse dates.
<code>nrows</code>	Number of rows to read from beginning of file.
<code>iterator</code>	Return a <code>TextParser</code> object for reading file piecemeal.
<code>chunksize</code>	For iteration, size of file chunks.
<code>skip_footer</code>	Number of lines to ignore at end of file.
<code>verbose</code>	Print various parser output information, like the number of missing values placed in non-numeric columns.
<code>encoding</code>	Text encoding for Unicode (e.g., <code>'utf-8'</code> for UTF-8 encoded text).
<code>squeeze</code>	If the parsed data only contains one column, return a Series.
<code>thousands</code>	Separator for thousands (e.g., <code>,</code> or <code>.</code> ).

## Metin Dosyalarını Parçalar Halinde Okumak

```
In [45]: pd.options.display.max_rows = 10
In [46]: result = pd.read_csv('ex6.csv')
In [47]: result
Out[47]:
   one two three four key
0  0.467976 -0.038649 -0.295344 -1.824726 L
1 -0.358893  1.404453  0.704965 -0.200638 B
2 -0.501840  0.659254 -0.421691 -0.057688 G
3  0.204886  1.074134  1.388361 -0.982404 R
4  0.354628 -0.133116  0.283763 -0.837063 Q
...
21 -0.358893  1.404453  0.704965 -0.200638 B
22 -0.501840  0.659254 -0.421691 -0.057688 G
23  0.204886  1.074134  1.388361 -0.982404 R
24  0.354628 -0.133116  0.283763 -0.837063 Q
25  0.467976 -0.038649 -0.295344 -1.824726 L
```

26 rows × 1 columns

- Önce max 10 satır gösterme özelliği yazıldı sonrasında dosya görüntülendi.

```
In [48]: pd.read_csv('ex6.csv', nrows=5)
```

Out[48]:

	one	two	three	four	key
0	0	0.467976	-0.038649	-0.295344	L
1	1	-0.358893	1.404453	0.704965	B
2	2	-0.501840	0.659254	-0.421691	G
3	3	0.204886	1.074134	1.388361	R
4	4	0.354628	-0.133116	0.283763	Q

- Bir dosyayı parçalar halinde okumak için parça boyutunu satır sayısı olarak belirtilir.

```
In [49]: chunker = pd.read_csv('ex6.csv', chunksize=6)
```

In [50]: chunker

Out[50]: <pandas.io.parsers.TextFileReader at 0x294e73d5e50>

- TextParser nesnesi, bazı bölümleri dosya yiğin boyutuna göre yinelemenize olanak tanır. Örneğin, ex6.csv üzerinde yinelemeler yapabiliriz, 'key' sütunundaki değer sayımlarının geçişine göre:

```
chunker = pd.read_csv('examples/ex6.csv', chunksize=1000)

tot = pd.Series([])
for piece in chunker:
    tot = tot.add(piece['key'].value_counts(), fill_value=0)

tot = tot.sort_values(ascending=False)
```

We have then:

In [40]: tot[:10]

Out[40]:

E	368.0
X	364.0
L	346.0
O	343.0
Q	340.0
M	338.0
J	337.0
F	335.0
K	334.0
H	330.0

dtype: float64

## Verileri Metin Formatına Yazma

- DataFrame'in `to_csv` yöntemini kullanarak verileri virgülle ayrılmış bir dosyaya yazabiliriz. Elbette başka sınırlayıcılar da kullanılabilir (`sys.stdout`'a metni basması için konsola sonuç yazılır.)

```
In [53]: data = pd.read_csv('ex5.csv')
```

```
In [54]: data
```

```
Out[54]:
```

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

```
In [ ]: data.to_csv('examples/out.csv')
```

```
In [60]: import sys
```

```
In [62]: data.to_csv(sys.stdout, sep='|')
```

```
|something|a|b|c|d|message  
0|one|1|2|3.0|4|  
1|two|5|6||8|world  
2|three|9|10|11.0|12|foo
```

```
In [63]: data.to_csv(sys.stdout, na_rep='NULL')
```

```
,something,a,b,c,d,message  
0,one,1,2,3.0,4,NULL  
1,two,5,6,NULL,8,world  
2,three,9,10,11.0,12,foo
```

```
In [64]: data.to_csv(sys.stdout, index=False, header=False)
```

```
one,1,2,3.0,4,  
two,5,6,,8,world  
three,9,10,11.0,12,foo
```

```
In [65]: data.to_csv(sys.stdout, index=False, columns=['a', 'b', 'c'])
```

```
a,b,c  
1,2,3.0  
5,6,  
9,10,11.0
```

```
In [66]: dates = pd.date_range('1/1/2000', periods=7)
```

```
2000-01-01,0
```

```
2000-01-02,1
```

```
2000-01-03,2
```

```
2000-01-04,3
```

```
2000-01-05,4
```

```
2000-01-06,5
```

```
2000-01-07,6
```

```
In [69]: ts = pd.Series(np.arange(7), index=dates)
```

```
In [70]: ts.to_csv('tseries.csv')
```

```
In [71]: !cat tseries.csv
```

## Sınırlandırılmış Formatlarla Çalışma

```
In [72]: import csv
In [73]: f = open('ex7.csv')
In [75]: reader = csv.reader(f)
In [76]: for line in reader:
    print(line)
['a', 'b', 'c']
['1', '2', '3']
['1', '2', '3']

In [77]: with open('ex7.csv') as f:
    lines = list(csv.reader(f))

In [82]: header, values = lines[0], lines[1:]

In [83]: data_dict = {h: v for h, v in zip(header, *values)})

In [84]: data_dict
Out[84]: {'a': ('1', '1'), 'b': ('2', '2'), 'c': ('3', '3')}
```

## CSV Diyalekt Seçenekleri

Argument	Description
delimiter	One-character string to separate fields; defaults to ', '.
lineterminator	Line terminator for writing; defaults to '\r\n'. Reader ignores this and recognizes cross-platform line terminators.
quotechar	Quote character for fields with special characters (like a delimiter); default is '''.
quoting	Quoting convention. Options include <code>csv.QUOTE_ALL</code> (quote all fields), <code>csv.QUOTE_MINIMAL</code> (only fields with special characters like the delimiter), <code>csv.QUOTE_NONNUMERIC</code> , and <code>csv.QUOTE_NONE</code> (no quoting). See Python's documentation for full details. Defaults to <code>QUOTE_MINIMAL</code> .
skipinitialspace	Ignore whitespace after each delimiter; default is <code>False</code> .
doublequote	How to handle quoting character inside a field; if <code>True</code> , it is doubled (see online documentation for full detail and behavior).
escapechar	String to escape the delimiter if <code>quoting</code> is set to <code>csv.QUOTE_NONE</code> ; disabled by default.

- Sınırlanmış dosyaları manuel olarak yazmak için csv.writer'ı kullanabilirsiniz. Açık ve yazılı bir metni kabul eder. ble dosya nesnesine ve csv.reader ile aynı diyalekti̇e ve format seçeneklerine sahiptir.

```
In [87]: with open('mydata.csv', 'w') as f:
    writer = csv.writer(f, dialect=my_dialect)
    writer.writerow(['one', 'two', 'three'])
    writer.writerow(['1', '2', '3'])
    writer.writerow(['4', '5', '6'])
    writer.writerow(['7', '8', '9'])
```

## JSON Verileri

- JSON (JavaScript Object Notation'ın kısaltması) web tarayıcıları ve diğer uygulamalar arasında HTTP isteği ile veri göndermek için standart formatlardan biri haline geldi. Bu CSV gibi tablolu bir metin biçiminden çok daha serbest biçimli bir veri biçimidir.
- JSON, null değeri ve null değeri haricinde neredeyse geçerli bir Python kodudur. Diğer bazı nüanslar (listelerin sondaki virgülere izin verilmemesi gibi) temel türler nesneler (diktler), diziler (listeler), dizeler, sayılar, boolean'lar ve boş değerlerdir. Tüm bir nesnedeki anahtarların her biri dize olmalıdır. JSON verilerini okumak ve yazmak için çeşitli Python kütüphaneleri vardır. JSON dizesini Python formuna dönüştürmek için json.loads kullanılır.

```
In [89]: import json
```

```
In [90]: obj = """
{"name": "Wes",
 "places_lived": ["United States", "Spain", "Germany"],
 "pet": null,
 "siblings": [{"name": "Scott", "age": 30, "pets": ["Zeus", "Zuko"]},
 {"name": "Katie", "age": 38,
 "pets": ["Sixes", "Stache", "Cisco"]}]
}"""
"""
```

```
In [91]: result = json.loads(obj)
```

```
In [92]: result
```

```
Out[92]: {'name': 'Wes',
 'places_lived': ['United States', 'Spain', 'Germany'],
 'pet': None,
 'siblings': [{'name': 'Scott', 'age': 30, 'pets': ['Zeus', 'Zuko']},
 {'name': 'Katie', 'age': 38, 'pets': ['Sixes', 'Stache', 'Cisco']}]}
```

```
In [93]: asjson = json.dumps(result)

In [94]: siblings = pd.DataFrame(result['siblings'], columns=['name', 'age'])

In [95]: siblings
Out[95]:
   name  age
0  Scott   30
1  Katie   38
```

- `json.dumps` Python nesnesini tekrar JSON'a dönüştürür.

```
In [96]: data = pd.read_json('example.json')

In [97]: data
Out[97]:
   a  b  c
0  1  2  3
1  4  5  6
2  7  8  9

In [98]: print(data.to_json())
{"a": {"0": 1, "1": 4, "2": 7}, "b": {"0": 2, "1": 5, "2": 8}, "c": {"0": 3, "1": 6, "2": 9}}

In [99]: print(data.to_json(orient='records'))
[{"a": 1, "b": 2, "c": 3}, {"a": 4, "b": 5, "c": 6}, {"a": 7, "b": 8, "c": 9}]
```

## XML ve HTML: Web Scraping

- Python'un her yerde veri okumak ve yazmak için birçok HTML ve XML kütüphanesi vardır. Örnekler arasında lxml, BeautifulSoup ve html5lib yer alır. lxml genel olarak nispeten daha hızlı, diğer kütüphaneler hatalı biçimlendirilmiş HTML veya XML dosyaları daha iyi işleyebilir.
- pandalar, lxml ve BeautifulSoup gibi kitaplıklarını kullanan yerleşik bir `read_html` işlevine sahiptir. Tabloları HTML dosyalarından DataFrame nesneleri olarak otomatik olarak ayırtırır.

```
In [23]: import pandas as pd
In [24]: tables = pd.read_html("https://www.fdic.gov/resources/resolutions/bank-failures/failed-bank-list/")
In [25]: len(tables)
Out[25]: 1
In [26]: failures = tables[0]
In [27]: failures.head()
Out[27]:
   Bank NameBank     CityCity  StateSt CertCert Acquiring InstitutionAI Closing DateClosing FundFund
0  Citizens Bank    Sac City    IA      8758  Iowa Trust & Savings Bank  November 3, 2023      10545
1 Heartland Tri-State Bank Elkhart    KS     25851        Dream First Bank, N.A.  July 28, 2023      10544
2 First Republic Bank  San Francisco CA     59017        JPMorgan Chase Bank, N.A.  May 1, 2023       10543
3 Signature Bank     New York    NY     57053        Flagstar Bank, N.A.  March 12, 2023      10540
4 Silicon Valley Bank Santa Clara CA     24735 First-Citizens Bank & Trust Company  March 10, 2023      10539
```

```
In [37]: from datetime import datetime
In [38]: close_timestamps = pd.to_datetime(failures['Closing DateClosing'])
In [36]: close_timestamps.dt.year.value_counts()
Out[36]: closing DateClosing
2010    157
2009    140
2011     92
2012     51
2008     25
2013     24
2014     18
2002     11
2017      8
2015      8
2023      5
2016      5
2020      4
2004      4
2019      4
2001      4
2007      3
2003      3
2000      2
Name: count, dtype: int64
```

- Buradaki örnekte öncelikle pandas ile html verilerini okundu. Sonra bazı özelliklerini denenip istenilen tablo elde edildi. Daha sonra yıllara göre banka iflaslarının sayısı hesaplandı.

## XML'i lxml.objectify ile Ayırıştırma

- XML (eXtensible Markup Language), diğer bir yaygın yapılandırılmış veri formatıdır. Hiyerarşik, iç içe geçmiş verileri meta verilerle taşıma.
- XML ve HTML yapısal olarak benzer, ancak XML daha geneldir.

```
In [88]: from lxml import objectify

In [89]: path = 'Performance_MNR.xml'
parsed = objectify.parse(open(path))
root = parsed.getroot()

In [91]: for elt in root.INDICATOR:
    el_data = {}
    for child in elt.getchildren():
        if child.tag in skip_fields:
            continue
        el_data[child.tag] = child.pyval
    data.append(el_data)

In [92]: data = []

skip_fields = ['PARENT_SEQ', 'INDICATOR_SEQ',
               'DESIRED_CHANGE', 'DECIMAL_PLACES']

In [93]: perf = pd.DataFrame(data)

In [94]: perf.head()

Out[94]:
```

- New York Metropolitan Ulaşım Otoritesi (MTA) otobüs ve tren hizmetlerine ilişkin veri serisini içeren bir dizi yayinallyamaktadır. Burada performans verilerine bakacağız, bir dizi XML dosyasında bulunur. Her tren veya otobüs servisinin farklı bir dosyası vardır ve (Performance\_MNR.xml gibi) aylık verileri içerir. lxml.objectify'yi kullanarak dosyayı ayırtırırız ve dosyanın root düğümüne bir referans alırız (Getroot'lu XML dosyası). root.INDICATOR, her <INDICATOR> XML ögesini sağlayan bir oluşturucu döndürür. Son olarak, bu sözlük listesi bir DataFrame'e dönüştürüldü.

```
In [102]: from io import StringIO

In [103]: tag = '<a href="http://www.google.com">Google</a>'

In [104]: root = objectify.parse(StringIO(tag)).getroot()

In [105]: root
Out[105]: <Element a at 0x23e5c78d380>

In [106]: root.get('href')
Out[106]: 'http://www.google.com'

In [107]: root.text
Out[107]: 'Google'
```

Burada 'a href' ile bağlantıya erişilir. Bağlantının root'u bağlantı text'i ve adresine erişilir.

## İkili (Binary) Veri Formatları

- Verileri (serileştirme olarak da bilinir) ikili dosyada verimli bir şekilde depolamanın en kolay yollarından biridir. Format Python'un yerleşik turşu serileştirmesini kullanıyor.

```
In [112]: frame = pd.read_csv('ex1.csv')
```

```
In [113]: frame
```

```
Out[113]:
```

	a	b	c	d	message
0	0	1	2	3	4 hello
1	1	5	6	7	8 world
2	2	9	10	11	12 foo

```
In [114]: frame.to_pickle('frame_pickle')
```

```
In [115]: pd.read_pickle('frame_pickle')
```

```
Out[115]:
```

	a	b	c	d	message
0	0	1	2	3	4 hello
1	1	5	6	7	8 world
2	2	9	10	11	12 foo

- Bir dosyada saklanan herhangi bir "pickled" nesneyi doğrudan yerleşik pickle kullanarak veya pandas.read\_pickle kullanarak daha da rahat bir şekilde okunabilir.

## bcolz

Blosc sıkıştırmasını temel alan sıkıştırılabilir, sütun odaklı ikili formatında kütüphanedir.

## Feather

R programlama toplulukları ile tasarlanan diller arası sütun odaklı bir dosya formatıdır. Feather Apache Arrow sütununu kullanıyor (hafıza formatı ).

## HDF5 Formatını Kullanma

- HDF5, büyük miktarlarda bilimsel verileri depolamak için tasarlanmış saygın bir dosya formatıdır. Bir C kütüphanesi olarak mevcuttur ve diğer birçok kütüphanede Java, Julia, MATLAB ve Python dahil mevcut olan arayzlere sahiptir.
- Her HDF5 dosyası birden fazla veri kümesini saklayabilir ve destekleyebilir.
- Daha basit formatlarla karşılaştırıldığında HDF5, anında sıkıştırmayı destekler.
- Tekrarlanan desenlere sahip verilerin işlenmesine olanak tanıyan çeşitli sıkıştırma modlarıyla daha verimli bir şekilde depolanır.
- HDF5, çok büyük verilerle çalışmak için iyi bir seçim olabilir.
- HDF5 dosyalarına PyTables veya h5py kullanarak doğrudan erişmek mümkün olsa da kütüphaneler, pandalar, Serileri ve DataFrame nesnesi depolamayı basitleştiren üst düzey bir arayüz sağlar.
- HDFStore sınıfı bir dict gibi çalışır ve düşük seviyeli işlemleri yönetir.

```
In [117]: import pandas as pd  
  
In [120]: frame = pd.DataFrame({'a': np.random.randn(100)})  
  
In [121]: store = pd.HDFStore('mydata.h5')  
  
In [122]: store['obj1'] = frame  
  
In [123]: store['obj1_col'] = frame['a']  
  
In [126]: store  
  
Out[126]: <class 'pandas.io.pytables.HDFStore'>  
File path: mydata.h5
```

```
In [127]: store['obj1']  
Out[127]:  
 a  
 0  0.809499  
 1  0.452145  
 2  1.296059  
 3  0.664105  
 4  -2.071584  
 ...  
 95  0.109636  
 96  -0.609370  
 97  -1.311129  
 98  0.500075  
 99  0.282426  
  
100 rows × 1 columns
```

```
In [128]: store.put('obj2', frame, format='table')
```

```
In [129]: store.select('obj2', where=['index >= 10 and index <= 15'])
```

```
Out[129]:
```

	a
10	-0.032801
11	-0.147226
12	-0.689883
13	-0.690790
14	-0.151628
15	-0.163094

```
In [131]: store.close()
```

```
In [132]: frame.to_hdf('mydata.h5', 'obj3', format='table')
```

```
In [133]: pd.read_hdf('mydata.h5', 'obj3', where=['index < 5'])
```

```
Out[133]:
```

	a
0	0.809499
1	0.452145
2	1.296059
3	0.664105
4	-2.071584

## Microsoft Excel Dosyalarını Okumak

```
In [176]: xlsx = pd.ExcelFile('ex1.xlsx')
```

```
In [ ]: pd.read_excel(xlsx, 'Sheet1')
```

```
In [ ]: frame = pd.read_excel('ex1.xlsx', 'Sheet1')
```

```
Out[180]:
```

	Unnamed: 0	a	b	c	d	message
0	0	NaN	2	3	4	hello
1	1	NaN	6	7	8	world
2	2	NaN	10	11	12	foo

- Bir dosyada birden fazla sayfa okuyorsanız ExcelFile'i oluşturmak daha hızlıdır ancak dosya adını pandas.read\_excel'e de iletebilirsiniz.

```
In [183]: writer = pd.ExcelWriter('ex2.xlsx')
```

```
In [184]: frame.to_excel(writer, 'Sheet1')
```

```
In [ ]: writer.save()
```

```
In [186]: frame.to_excel('ex2.xlsx')
```

- Pandaların verilerini Excel formatına yazmak için önce bir ExcelWriter oluşturup, ardından pandas nesnelerinin to\_excel yöntemini kullanarak ona veri yazdırabilirsiniz.

## Web API'leriyle Etkileşim Kurma

- Pek çok web sitesinde JSON veya başka bir format aracılığıyla veri akışı sağlayan genel API'ler bulunur. Bu API'lere Python'dan erişmenin birkaç yolu vardır; kullanımı kolay tavsiye edilen yöntemlerden biri request paketidir.

```
In [187]: import requests
```

```
In [190]: url = 'https://api.github.com/repos/pandas-dev/pandas/issues'
```

```
In [191]: resp = requests.get(url)
```

```
In [192]: resp
```

```
Out[192]: <Response [200]>
```

```
In [193]: data = resp.json()
```

```
In [194]: data[0]['title']
```

```
Out[194]: 'BUG: calendar day date_range AmbiguousTimeError with unambiguous DST boundaries'
```

- GitHub'da add-on isteklerini (requests) kullanarak pandalar için son 30 GitHub sorununu bulmak için bir GET HTTP yapabiliriz.

- Response nesnesinin json yöntemi, yerel Python nesnelerine ayrıstırılmış JSON içeren bir sözlük döndürecektir.

```
In [195]: issues = pd.DataFrame(data, columns=['number', 'title',
                                             'labels', 'state'])

In [196]: issues
```

number	title	labels	state
0	BUG: calendar day date_range AmbiguousTimeError	[{"id": 76811, "node_id": "MDU6TGFiZWw3NjgxMQ=...}	open
1	TST: dt64 units	[]	open
2	BUG: Read CSV on python engine fails with call...	[]	open
3	DEPR: Default of observed=False in DataFrame.p...	[{"id": 13098779, "node_id": "MDU6TGFiZWwxMzA5...}	open
4	DEPR: observed=False default for DataFrame.pivot...	[{"id": 13098779, "node_id": "MDU6TGFiZWwxMzA5...}	open
5	Table cell styling	[]	open
6	DOC: Fix typo in copy_on_write docs	[]	open
7	BUG: 'to_numpy' raise 'ValueError: cannot conv...	[{"id": 76811, "node_id": "MDU6TGFiZWw3NjgxMQ=...}	open
8	DOC: Update release instructions	[{"id": 134699, "node_id": "MDU6TGFiZWwxMzQ2OT...}	open
9	BUG: DataFrame([empty EA Series]) returns NAs ...	[{"id": 76811, "node_id": "MDU6TGFiZWw3NjgxMQ=...}	open
10	CoW warning mode: enable chained assignment wa...	[{"id": 2085877452, "node_id": "MDU6TGFiZWwyMD...}	open
11	TST/CLN: Remove makeFloat/Period/Int/NumericIndex	[{"id": 127685, "node_id": "MDU6TGFiZWwxMjc2OD...}	open
12	BUG: DataFrame.update not operating in-place f...	[{"id": 2085877452, "node_id": "MDU6TGFiZWwyMD...}	open
13	BUG: DataFrame.update not operating in-place f...	[{"id": 76811, "node_id": "MDU6TGFiZWw3NjgxMQ=...}	open
14	BUG: `DataFrame.groupby with axis=1 is depreca...	[{"id": 233160, "node_id": "MDU6TGFiZWwyMzMxNj...}	open
15	BUG: ``.loc[., 'a']= `` is inplace when assigni...	[{"id": 76811, "node_id": "MDU6TGFiZWw3NjgxMQ=...}	open
16	BUG: ``pd.nlargest`` with ``keep='all'' does not ...	[{"id": 76811, "node_id": "MDU6TGFiZWw3NjgxMQ=...}	open
17	Switch arrow type for string array to large st...	[]	open
18	DOC: DataFrame.update doctests	[{"id": 134699, "node_id": "MDU6TGFiZWwxMzQ2OT...}	open
19	BUG: `DataFrame.astype` does not suppress exce...	[{"id": 76811, "node_id": "MDU6TGFiZWw3NjgxMQ=...}	open
20	CoW: Avoid warning for ArrowDtypes when settin...	[{"id": 2085877452, "node_id": "MDU6TGFiZWwyMD...}	open
21	CoW: Add warning for slicing a Series with a M...	[{"id": 2085877452, "node_id": "MDU6TGFiZWwyMD...}	open
22	BUG: inconsistency in freq of DatetimeIndex wh...	[{"id": 76811, "node_id": "MDU6TGFiZWw3NjgxMQ=...}	open
23	CoW: Avoid warning in apply for mixed dtype frame	[{"id": 2085877452, "node_id": "MDU6TGFiZWwyMD...}	open
24	CoW: Avoid warning if temporary object is a copy	[{"id": 2085877452, "node_id": "MDU6TGFiZWwyMD...}	open
25	Update indexing unpacking logic for single blo...	[]	open
26	WEB: Better management of releases	[{"id": 1508144531, "node_id": "MDU6TGFiZWwxNT...}	open
27	BUG: Creating a string column on a mask result...	[{"id": 76811, "node_id": "MDU6TGFiZWw3NjgxMQ=...}	open
28	BUG: expanding dataframe row-wise with single-...	[{"id": 76811, "node_id": "MDU6TGFiZWw3NjgxMQ=...}	open
29	Adjust tests in json folder for new string option	[{"id": 127685, "node_id": "MDU6TGFiZWwxMjc2OD...}	open

## Veritabanlarıyla Etkileşim

```
In [198]: import sqlite3

In [199]: query = """
CREATE TABLE test
(a VARCHAR(20),   b VARCHAR(20),
c REAL,          d INTEGER
);"""

In [200]: con = sqlite3.connect('mydata.sqlite')

In [201]: con.execute(query)

Out[201]: <sqlite3.Cursor at 0x23e5eb030c0>

In [202]: con.commit()
```

```
In [203]: data = [('Atlanta', 'Georgia', 1.25, 6),
              ('Tallahassee', 'Florida', 2.6, 3),
              ('Sacramento', 'California', 1.7, 5)]
```

```
In [204]: stmt = "INSERT INTO test VALUES(?, ?, ?, ?)"
```

```
In [205]: con.executemany(stmt, data)
```

```
Out[205]: <sqlite3.Cursor at 0x23e5e987e40>
```

```
In [206]: con.commit()
```

```
In [207]: cursor = con.execute('select * from test')
```

```
In [208]: rows = cursor.fetchall()
```

```
In [209]: rows
```

```
Out[209]: [('Atlanta', 'Georgia', 1.25, 6),
            ('Tallahassee', 'Florida', 2.6, 3),
            ('Sacramento', 'California', 1.7, 5)]
```

```
In [210]: cursor.description
```

```
Out[210]: (('a', None, None, None, None, None),
            ('b', None, None, None, None, None),
            ('c', None, None, None, None, None),
            ('d', None, None, None, None, None))
```

```
In [211]: pd.DataFrame(rows, columns=[x[0] for x in cursor.description])
```

```
Out[211]:
```

	a	b	c	d
0	Atlanta	Georgia	1.25	6
1	Tallahassee	Florida	2.60	3
2	Sacramento	California	1.70	5

- SQL'den DataFrame'e veri yüklemek oldukça basittir ve pandas süreci basitleştirmek için bazı işlevler vardır. Örnek olarak sqlite3 sürücüsünü kullanarak bir SQLite veritabanı oluşturuldu. İçerisine yeni veriler eklendi.
- Çoğu Python SQL sürücüsü (PyODBC, psycopg2, MySQLdb, pymssql, vb.) bir tablodan veri seçerken tuple'ların listelerini döndürür.

```
In [215]: import sqlalchemy as sqa
In [216]: db = sqa.create_engine('sqlite:///mydata.sqlite')
In [217]: pd.read_sql('select * from test', db)
Out[217]:
   a      b      c   d
0  Atlanta  Georgia  1.25  6
1  Tallahassee  Florida  2.60  3
2  Sacramento  California  1.70  5
```

- SQLAlchemy projesi, soyutlama yapan popüler bir Python SQL araç takımıdır. SQL veritabanları arasındaki ortak farklılıkların çoğunu ortadan kaldırır. Genel bir SQLAlchemy bağlantı verileri kolayca okumanızı sağlayan read\_sql işlevine sahiptir. Burada aynı SQLite veritabanına SQLAlchemy ile bağlanılıp daha önce oluşturulan tablodaki verileri okundu.

## Veri Temizleme ve Hazırlama

- Veri analizi ve modelleme sırasında önemli miktarda zaman veri hazırlığına harcanır: yükleme, temizleme, dönüştürme ve yeniden düzenleme.

### Eksik Verileri İşleme

- Eksik veriler, birçok veri analizi uygulamasında yaygın olarak ortaya çıkar. Hedeflerden biri Pandaların amacı eksik verilerle çalışmayı mümkün olduğunda acısız hale getirmektir. Örneğin, panda nesnelerine ilişkin tüm tanımlayıcı istatistikler, varsayılan olarak eksik verileri hariç tutar.
- Eksik verilerin panda nesnelerinde temsil edilme şekli biraz kusurludur ancak birçok kullanıcı için işlevseldir. Sayısal veriler için pandalar kayan noktayı kullanır eksik verileri temsil etmek için NaN (Sayı Değil) değerini kullanır. Buna ‘sentinel’ değer denir kolayca tespit edilebilir.

```
In [218]: string_data = pd.Series(['aardvark', 'artichoke', np.nan, 'avocado'])
```

```
In [219]: string_data
```

```
Out[219]: 0    aardvark  
1    artichoke  
2        NaN  
3    avocado  
dtype: object
```

```
In [220]: string_data.isnull()
```

```
Out[220]: 0    False  
1    False  
2     True  
3    False  
dtype: bool
```

- Pandalarda, R programlama dilinde kullanılan bir kural referans alındı. Eksik verilere, mevcut değil anlamına gelen NA adını verin. İstatistik uygulamalarında, NA verileri, var olmayan veya var olan ancak gözlemlenmemiş veriler olabilir. (örneğin veri toplamıyla ilgili sorunlar nedeniyle). Verileri temizlerken analizde, verileri tanımlamak için eksik verilerin kendisi üzerinde analiz yapmak genellikle önemlidir.

```
In [221]: string_data[0] = None
```

```
In [222]: string_data.isnull()
```

```
Out[222]: 0     True  
1    False  
2     True  
3    False  
dtype: bool
```

## NA İşleme Yöntemleri

Argument	Description
dropna	Filter axis labels based on whether values for each label have missing data, with varying thresholds for how much missing data to tolerate.
fillna	Fill in missing data with some value or using an interpolation method such as 'ffill' or 'bfill'.
isnull	Return boolean values indicating which values are missing/NA.
notnull	Negation of isnull.

## Eksik Verileri Filtreleme

- Eksik verileri filtrelemenin birkaç yolu vardır. Bunu pandas. isnull ve boolean indekslemeyi kullanarak yapmak daha çok tercih edilir. Dropna da yardımcı olabilir. Bir Seride, Seriyi yalnızca boş olmayan veriler ve dizin değerleri ile döndürür.

```
In [223]: from numpy import nan as NA
```

```
In [224]: data = pd.Series([1, NA, 3.5, NA, 7])
```

```
In [225]: data.dropna()
```

```
Out[225]: 0    1.0  
2    3.5  
4    7.0  
dtype: float64
```

```
In [226]: data[data.notnull()]
```

```
Out[226]: 0    1.0  
2    3.5  
4    7.0  
dtype: float64
```

```
In [227]: data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],  
                           [NA, NA, NA], [NA, 6.5, 3.]])
```

```
In [228]: cleaned = data.dropna()
```

```
In [229]: data
```

```
Out[229]:  
      0    1    2  
-----  
 0  1.0  6.5  3.0  
 1  1.0  NaN  NaN  
 2  NaN  NaN  NaN  
 3  NaN  6.5  3.0
```

```
In [230]: cleaned
```

```
Out[230]:  
      0    1    2  
-----  
 0  1.0  6.5  3.0
```

- Dropna ile NaN olan tüm satırlar silinebilir.

```
In [231]: data.dropna(how = 'all')
```

```
Out[231]:
```

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
3	NaN	6.5	3.0

- Tüm değerleri NaN olan satır silinir.

```
In [232]: data[4] = NA
```

```
In [233]: data
```

```
Out[233]:
```

	0	1	2	4
0	1.0	6.5	3.0	NaN
1	1.0	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	6.5	3.0	NaN

```
In [234]: data.dropna(axis = 1, how = 'all')
```

```
Out[234]:
```

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

- DataFrame satırlarını filtrelemenin bir yolu da genellikle zaman serisi verileriyle ilgilidir. Yalnızca belirli sayıda gözlem içeren satırları tutmak istiyorsanız bunu thresh argümanıyla yapabilirsiniz.

```
In [235]: df = pd.DataFrame(np.random.randn(7, 3))
```

```
In [236]: df.iloc[:4, 1] = NA
```

```
In [237]: df.iloc[:, 2] = NA
```

```
In [238]: df
```

```
Out[238]:
```

	0	1	2
0	-0.751017	NaN	NaN
1	-0.612584	NaN	NaN
2	-0.294537	NaN	-0.026366
3	0.896208	NaN	-0.473022
4	0.739283	-0.277325	0.595485
5	-1.787355	-0.925672	-0.497830
6	0.742676	-0.421850	-0.823856

```
In [239]: df.dropna()
```

```
Out[239]:
```

	0	1	2
4	0.739283	-0.277325	0.595485
5	-1.787355	-0.925672	-0.497830
6	0.742676	-0.421850	-0.823856

```
In [240]: df.dropna(thresh=2)
```

```
Out[240]:
```

	0	1	2
2	-0.294537	NaN	-0.026366
3	0.896208	NaN	-0.473022
4	0.739283	-0.277325	0.595485
5	-1.787355	-0.925672	-0.497830
6	0.742676	-0.421850	-0.823856

## Eksik Verilerin Doldurulması

```
In [241]: df.fillna(0)
```

```
Out[241]:
```

	0	1	2
0	-0.751017	0.000000	0.000000
1	-0.612584	0.000000	0.000000
2	-0.294537	0.000000	-0.026366
3	0.896208	0.000000	-0.473022
4	0.739283	-0.277325	0.595485
5	-1.787355	-0.925672	-0.497830
6	0.742676	-0.421850	-0.823856

```
In [242]: df.fillna({1: 0.5, 2: 0})
```

```
Out[242]:
```

	0	1	2
0	-0.751017	0.500000	0.000000
1	-0.612584	0.500000	0.000000
2	-0.294537	0.500000	-0.026366
3	0.896208	0.500000	-0.473022
4	0.739283	-0.277325	0.595485
5	-1.787355	-0.925672	-0.497830
6	0.742676	-0.421850	-0.823856

- Fillna bir dict ile çağrııldığında, her sütun için farklı bir doldurma değeri kullanılabilir.

```
In [253]: df = pd.DataFrame(np.random.randn(6, 3))
```

```
In [254]: df.iloc[2:, 1] = NA
```

```
In [255]: df.iloc[4:, 2] = NA
```

```
In [256]: df
```

```
Out[256]:
```

	0	1	2
0	1.369017	-0.743139	-1.716792
1	1.350089	1.288066	1.572971
2	-0.346273	NaN	0.620443
3	-0.762129	NaN	0.022266
4	0.161975	NaN	NaN
5	-0.721289	NaN	NaN

- Yeniden indeksleme için mevcut olan enterpolasyon yöntemlerinin aynısı fillna ile de kullanılabilir.

```
In [257]: df.fillna(method='ffill')
```

```
Out[257]:
```

	0	1	2
0	1.369017	-0.743139	-1.716792
1	1.350089	1.288066	1.572971
2	-0.346273	1.288066	0.620443
3	-0.762129	1.288066	0.022266
4	0.161975	1.288066	0.022266
5	-0.721289	1.288066	0.022266

```
In [258]: df.fillna(method='ffill', limit=2)
```

```
Out[258]:
```

	0	1	2
0	1.369017	-0.743139	-1.716792
1	1.350089	1.288066	1.572971
2	-0.346273	1.288066	0.620443
3	-0.762129	1.288066	0.022266
4	0.161975	NaN	0.022266
5	-0.721289	NaN	0.022266

- Fillna ile biraz yaratıcılıkla başka birçok şey yapabilirsiniz. Örneğin, bir Serinin ortalama veya medyan değerini bulabilirsiniz.

```
In [259]: data = pd.Series([1., NA, 3.5, NA, 7])
```

```
In [260]: data.fillna(data.mean())
```

```
Out[260]: 0    1.000000  
1    3.833333  
2    3.500000  
3    3.833333  
4    7.000000  
dtype: float64
```

## fillna İşlevleri

### Argument Description

value	Scalar value or dict-like object to use to fill missing values
method	Interpolation; by default 'ffill' if function called with no other arguments
axis	Axis to fill on; default axis=0
inplace	Modify the calling object without producing a copy
limit	For forward and backward filling, maximum number of consecutive periods to fill

# Veri Dönüşümü

## Yinelenenleri Kaldırma

```
In [5]: data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],
                           'k2': [1, 1, 2, 3, 3, 4, 4]})
```

```
In [6]: data
```

```
Out[6]:
```

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4

```
In [8]: data.duplicated()
```

```
Out[8]: 0    False
        1    False
        2    False
        3    False
        4    False
        5    False
        6    True
       dtype: bool
```

- Bu örnekte bir önceki satıra göre yinelenme durumu olup olmadığı kontrol edilir.

```
In [9]: data.drop_duplicates()
```

```
Out[9]:
```

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4

- drop\_duplicates, kopyalanan dizinin olduğu yerde bir DataFrame döndürür

```
In [10]: data['v1'] = range(7)
```

```
In [11]: data.drop_duplicates(['k1'])
```

```
Out[11]:
```

	k1	k2	v1
0	one	1	0
1	two	1	1

```
In [12]: data.drop_duplicates(['k1', 'k2'], keep='last')
```

```
Out[12]:
```

	k1	k2	v1
0	one	1	0
1	two	1	1
2	one	2	2
3	two	3	3
4	one	3	4
6	two	4	6

- Bu yöntemlerin her ikisi de varsayılan olarak tüm sütunları dikkate alır; alternatif olarak kopyaları tespit etmek için bunların herhangi bir alt kümesini belirtin. Bu örnekte yeni bir sütun oluşturulup 'k1' sütununa göre filtrelandı.

## Bir Fonksiyon veya Haritalama (Mapping) Kullanarak Verileri Dönüşürme

```
In [13]: data = pd.DataFrame({'food': ['bacon', 'pulled pork', 'bacon',  
'Pastrami', 'corned beef', 'Bacon',  
'pastrami', 'honey ham', 'nova lox'],  
'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
```

```
In [14]: data
```

```
Out[14]:
```

	food	ounces
0	bacon	4.0
1	pulled pork	3.0
2	bacon	12.0
3	Pastrami	6.0
4	corned beef	7.5
5	Bacon	8.0
6	pastrami	3.0
7	honey ham	5.0
8	nova lox	6.0

```
In [15]: meat_to_animal = {  
    'bacon': 'pig',  
    'pulled pork': 'pig',  
    'pastrami': 'cow',  
    'corned beef': 'cow',  
    'honey ham': 'pig',  
    'nova lox': 'salmon'  
}
```

```
In [16]: lowercased = data['food'].str.lower()
```

```
In [17]: lowercased
```

```
Out[17]: 0      bacon  
1      pulled pork  
2      bacon  
3      pastrami  
4      corned beef  
5      bacon  
6      pastrami  
7      honey ham  
8      nova lox  
Name: food, dtype: object
```

```
In [18]: data['animal'] = lowercased.map(meat_to_animal)
```

```
In [19]: data
```

```
Out[19]:
```

	food	ounces	animal
0	bacon	4.0	pig
1	pulled pork	3.0	pig
2	bacon	12.0	pig
3	Pastrami	6.0	cow
4	corned beef	7.5	cow
5	Bacon	8.0	pig
6	pastrami	3.0	cow
7	honey ham	5.0	pig
8	nova lox	6.0	salmon

- Burada yeni bir sütun eklendi (map olarak). Hepsinin küçük harfli olması için str.lower komutu kullanıldı.

```
In [20]: data['food'].map(lambda x: meat_to_animal[x.lower()])
```

```
Out[20]: 0      pig
          1      pig
          2      pig
          3      cow
          4      cow
          5      pig
          6      cow
          7      pig
          8  salmon
Name: food, dtype: object
```

- Lambda fonksiyonuyla da bu işlemi yapabiliriz.

## Değerleri Değiştirme

```
In [21]: data = pd.Series([1., -999., 2., -999., -1000., 3.])
```

```
In [22]: data
```

```
Out[22]: 0      1.0
          1     -999.0
          2      2.0
          3     -999.0
          4    -1000.0
          5      3.0
dtype: float64
```

```
In [23]: data.replace(-999, np.nan)
```

```
Out[23]: 0      1.0
          1      NaN
          2      2.0
          3      NaN
          4    -1000.0
          5      3.0
dtype: float64
```

- Burada -999 olan değerler NaN olarak değiştirildi.
- `data.replace` yöntemi `data.str.replace`'den farklıdır. Bu, eleman bazında dizinin yerine geçme işlemini gerçekleştirir.

```
In [25]: data.replace([-999, -1000], np.nan)
```

```
Out[25]: 0    1.0
1    NaN
2    2.0
3    NaN
4    NaN
5    3.0
dtype: float64
```

```
In [26]: data.replace([-999, -1000], [np.nan, 0])
```

```
Out[26]: 0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

```
In [27]: data.replace({-999: np.nan, -1000: 0})
```

```
Out[27]: 0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

- Üç farklı gösterimle hangi değerin NaN ya da 0 yapılacağı gösterilmiştir.

## Eksen İndekslerini Yeniden Adlandırma

```
In [28]: data = pd.DataFrame(np.arange(12).reshape((3, 4)),
                           index=['Ohio', 'Colorado', 'New York'],
                           columns=['one', 'two', 'three', 'four'])
```

```
In [29]: data
```

```
Out[29]:
   one  two  three  four
Ohio    0    1     2     3
Colorado 4    5     6     7
New York 8    9    10    11
```

```
In [31]: transform = lambda x: x[:4].upper()
```

```
In [32]: data.index.map(transform)
```

```
Out[32]: Index(['OHIO', 'COLO', 'NEW '], dtype='object')
```

- Burada eksendeki değerlerin ilk dört değerinin hepsi büyük harf olarak dönüştürüldü.

```
In [33]: data.index = data.index.map(transform)
```

```
In [34]: data
```

```
Out[34]:
```

	one	two	three	four
OHIO	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

- Datanın title'ının değerlerinin hepsi büyük harfe dönüştürüldü.

```
In [35]: data.rename(index=str.title, columns=str.upper)
```

```
Out[35]:
```

	ONE	TWO	THREE	FOUR
Ohio	0	1	2	3
Colo	4	5	6	7
New	8	9	10	11

```
In [36]: data.rename(index={'OHIO': 'INDIANA'}, inplace=True)
```

```
In [37]: data
```

```
Out[37]:
```

	one	two	three	four
INDIANA	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

## Ayrıştırma ve Gruplama

- Sürekli veriler genellikle ayrık hale getirilir veya analiz için başka şekilde kutulara (bins) ayrılır.

```
In [38]: ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

```
In [39]: bins = [18, 25, 35, 60, 100]
```

```
In [40]: cats = pd.cut(ages, bins)
```

```
In [41]: cats
```

```
Out[41]: [(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100],
(35, 60], (35, 60], (25, 35])
Length: 12
Categories (4, interval[int64, right]): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]
```

- Cut fonksiyonu ile 18'den 25'e, 26'dan 35'e, 36'dan 60'a ve son olarak 61 ve üzeri kutulara bölündü.

```
In [42]: cats.codes
Out[42]: array([0, 0, 0, 1, 0, 0, 2, 1, 3, 2, 2, 1], dtype=int8)

In [43]: cats.categories
Out[43]: IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]], dtype='interval[int64, right]')

In [44]: pd.value_counts(cats)
Out[44]: (18, 25]      5
          (25, 35]      3
          (35, 60]      3
          (60, 100]     1
Name: count, dtype: int64
```

- Ayrıca bir liste veya diziyi labels seçeneğine ileterek kendi bin adınızı da verebilirsiniz.

```
In [45]: group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']
In [46]: pd.cut(ages, bins, labels=group_names)
Out[46]: ['Youth', 'Youth', 'Youth', 'YoungAdult', 'Youth', ..., 'YoungAdult', 'Senior',
          'MiddleAged', 'MiddleAged', 'YoungAdult']
Length: 12
Categories (4, object): ['Youth' < 'YoungAdult' < 'MiddleAged' < 'Senior']
```

- Burada rastgele oluşturduğumuz dizide 2'lük değerlerle dört tane kutulara bölündü.

```
In [47]: data = np.random.rand(20)
In [48]: data
Out[48]: array([0.28560152, 0.30153411, 0.40773112, 0.14709136, 0.2136246 ,
               0.19064143, 0.95136144, 0.97885695, 0.78381127, 0.00992525,
               0.35944159, 0.60815954, 0.3297958 , 0.37406865, 0.29602596,
               0.71404941, 0.92885134, 0.02126693, 0.60834438, 0.95142119])

In [49]: pd.cut(data, 4, precision=2)
Out[49]: [(0.25, 0.49], (0.25, 0.49], (0.25, 0.49], (0.009, 0.25], (0.009, 0.25], ...,
          (0.49, 0.74], (0.74, 0.98], (0.009, 0.25], (0.49, 0.74], (0.74, 0.98])
Length: 20
Categories (4, interval[float64, right]): [(0.009, 0.25] < (0.25, 0.49] < (0.
49, 0.74] < (0.74, 0.98)]
```

- Yakından ilişkili bir işlev olan qcut, verileri örnek niceliklere göre gruplandırır. qcut örnek yüzdelik dilimleri kullandığından tanım gereği kabaca eşit büyüklükte kutular elde edeceksiniz.

```
In [51]: data = np.random.randn(1000)
```

```
In [52]: cats = pd.qcut(data, 4)
```

```
In [53]: cats
```

```
Out[53]: [(-0.702, 0.0787], (-0.702, 0.0787], (-0.702, 0.0787], (0.706, 2.752], (-0.702, 0.0787], ..., (-0.702, 0.0787], (0.706, 2.752], (0.0787, 0.706], (-0.702, 0.0787], (0.0787, 0.706])
Length: 1000
Categories (4, interval[float64, right]): [(-3.42, -0.702] < (-0.702, 0.0787] < (0.0787, 0.706] < (0.706, 2.752]]
```

```
In [56]: pd.value_counts(cats)
```

```
Out[56]: (-3.42, -0.702]      250
(-0.702, 0.0787]      250
(0.0787, 0.706]      250
(0.706, 2.752]      250
Name: count, dtype: int64
```

## Aykırı Değerleri Tespit Etme ve Filtreleme

- Aykırı değerlerin filtrelenmesi veya dönüştürülmesi büyük ölçüde dizi işlemlerinin uygulanmasıyla ilgilidir.

```
In [57]: data = pd.DataFrame(np.random.randn(1000, 4))
```

```
In [58]: data.describe()
```

```
Out[58]:
```

	0	1	2	3
<b>count</b>	1000.000000	1000.000000	1000.000000	1000.000000
<b>mean</b>	0.006738	-0.026416	0.001911	0.024380
<b>std</b>	1.035627	0.985689	0.989342	0.988116
<b>min</b>	-3.410813	-3.262865	-3.566895	-3.640323
<b>25%</b>	-0.723862	-0.712966	-0.630752	-0.607872
<b>50%</b>	0.040344	-0.042249	0.024487	0.030729
<b>75%</b>	0.682244	0.616719	0.682073	0.696103
<b>max</b>	3.324574	3.543986	3.386966	3.179037

- Sütunlardan birinde mutlak değer olarak 3'ü aşan değerleri bulur.

```
In [59]: col = data[2]
```

```
In [60]: col[np.abs(col) > 3]
```

```
Out[60]: 137    -3.274237
158     3.386966
516     3.135317
979    -3.566895
Name: 2, dtype: float64
```

```
In [62]: data[np.abs(data) > 3] = np.sign(data) * 3
```

```
In [63]: data.describe()
```

```
Out[63]:
```

	0	1	2	3
<b>count</b>	1000.000000	1000.000000	1000.000000	1000.000000
<b>mean</b>	0.006696	-0.026697	0.002230	0.024841
<b>std</b>	1.032977	0.983034	0.984907	0.985387
<b>min</b>	-3.000000	-3.000000	-3.000000	-3.000000
<b>25%</b>	-0.723862	-0.712966	-0.630752	-0.607872
<b>50%</b>	0.040344	-0.042249	0.024487	0.030729
<b>75%</b>	0.682244	0.616719	0.682073	0.696103
<b>max</b>	3.000000	3.000000	3.000000	3.000000

- `np.sign(data)` ifadesi, verilerde pozitif veya negatif değerlerin olup olmadığına bağlı olarak 1 ve -1 değerleri üretir.

```
In [64]: np.sign(data).head()
```

```
Out[64]:
```

	0	1	2	3
<b>0</b>	1.0	1.0	1.0	-1.0
<b>1</b>	-1.0	1.0	1.0	1.0
<b>2</b>	-1.0	-1.0	1.0	-1.0
<b>3</b>	-1.0	1.0	1.0	1.0
<b>4</b>	1.0	1.0	1.0	-1.0

## Permütasyon ve Rastgele Örnekleme

- Oluşturulan dizinin 5'inci satırının permütasyonunu alır.

```
In [65]: df = pd.DataFrame(np.arange(5 * 4).reshape((5, 4)))
```

```
In [66]: sampler = np.random.permutation(5)
```

```
In [67]: sampler
```

```
Out[67]: array([3, 1, 0, 4, 2])
```

```
In [68]: df
```

```
Out[68]:
```

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19

```
In [69]: df.take(sampler)
```

```
Out[69]:
```

	0	1	2	3
3	12	13	14	15
1	4	5	6	7
0	0	1	2	3
4	16	17	18	19
2	8	9	10	11

```
In [70]: df.sample(n=3)
```

```
Out[70]:
```

	0	1	2	3
2	8	9	10	11
1	4	5	6	7
3	12	13	14	15

```
In [71]: choices = pd.Series([5, 7, -1, 6, 4])
```

```
In [72]: draws = choices.sample(n=10, replace=True)
```

```
In [73]: draws
```

```
Out[73]: 1    7  
        4    4  
        0    5  
        0    5  
        3    6  
        4    4  
        0    5  
        1    7  
        0    5  
        4    4  
dtype: int64
```

- Değiştirmeden rastgele bir alt küme seçmek için sample yöntemini kullanabilirsiniz.
- Değiştirilen bir örnek oluşturmak için (tekrarlanan seçimlere izin vermek için), replace=True yöntemini uygulayın.

## Hesaplama Göstergesi | Kukla (Dummy) Değişkenler

- İstatistiksel modelleme veya makine öğrenimi uygulamaları için başka bir dönüşüm türü kategorik bir değişkeni “kukla (dummy)” veya “gösterge (indicator)” matrisine dönüştürmektir.
- Eğer bir DataFrame'deki sütun k farklı değere sahipse, tüm 1'leri ve 0'ları içeren k sütunlu çerçeveli bir matris veya veri elde edersiniz.

```
In [74]: df = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                           'data1': range(6)})
```

```
In [76]: pd.get_dummies(df['key'])
```

Out[76]:

	a	b	c
0	False	True	False
1	False	True	False
2	True	False	False
3	False	False	True
4	True	False	False
5	False	True	False

- Bazı durumlarda, DataFrame'deki sütunlara Daha sonra diğer verilerle birleştirilebilen çerçeveli bir önek eklemek isteyebilirsiniz. get\_dummies'in bunu yapmak için prefix argümanı vardır.

```
In [78]: dummies = pd.get_dummies(df['key'], prefix='key')
```

```
In [79]: df_with_dummy = df[['data1']].join(dummies)
```

```
In [80]: df_with_dummy
```

Out[80]:

	data1	key_a	key_b	key_c
0	0	False	True	False
1	1	False	True	False
2	2	True	False	False
3	3	False	False	True
4	4	True	False	False
5	5	False	True	False

- İstatistiksel uygulamalar için yararlı bir uygulama, get\_dummies'i ayrık bir yaklaşımla birleştirmektir.

```
In [98]: np.random.seed(12345)

In [99]: values = np.random.rand(10)

In [100]: values
Out[100]: array([0.92961609, 0.31637555, 0.18391881, 0.20456028, 0.56772503,
       0.5955447 , 0.96451452, 0.6531771 , 0.74890664, 0.65356987])

In [102]: bins = [0, 0.2, 0.4, 0.6, 0.8, 1]

In [103]: pd.get_dummies(pd.cut(values, bins))

Out[103]:
      (0.0, 0.2]  (0.2, 0.4]  (0.4, 0.6]  (0.6, 0.8]  (0.8, 1.0]
0    False     False     False     False     True
1    False     True     False     False     False
2    True     False     False     False     False
3    False     True     False     False     False
4    False     False     True     False     False
5    False     False     True     False     False
6    False     False     False     False     True
7    False     False     False     True     False
8    False     False     False     True     False
9    False     False     False     True     False
```

## Dize Manipülasyonu

- Python uzun zamandır popüler bir ham veri işleme dili olmuştur. dize ve metin işleme için kullanım kolaylığı. string nesnesinin yerleşik yöntemleri ile çoğu metin işlemi basitleştirilmiştir.

## Dize Nesnesi Yöntemleri

```
In [104]: val = 'a,b, guido'
```

```
In [106]: val.split(',')
```

```
Out[106]: ['a', 'b', ' guido']
```

```
In [107]: pieces = [x.strip() for x in val.split(',')]
```

```
In [108]: pieces
```

```
Out[108]: ['a', 'b', 'guido']
```

```
In [109]: first, second, third = pieces
```

```
In [110]: first + '::' + second + '::' + third
```

```
Out[110]: 'a::b::guido'
```

```
In [111]: '::'.join(pieces)
```

```
Out[111]: 'a::b::guido'
```

- Burada string dizesini string bazı değerlerle ayırıp listelenir.

```
In [112]: 'guido' in val
```

```
Out[112]: True
```

```
In [113]: val.index(',')
```

```
Out[113]: 1
```

```
In [114]: val.find(':')
```

```
Out[114]: -1
```

```
In [115]: val.index(':')
```

```
-----  
ValueError  
Cell In[115], line 1  
----> 1 val.index(':')
```

```
Traceback (most recent call last)
```

```
ValueError: substring not found
```

Find ve index arasındaki farkın, index'in bir istisna oluşturması olduğunu unutmayın.

```
In [116]: val.replace(',', '::')
```

```
Out[116]: 'a::b:: guido'
```

```
In [117]: val.replace(',', '')
```

```
Out[117]: 'ab guido'
```

## Python Yerleşik Dize Yöntemleri

Argument	Description
count	Return the number of non-overlapping occurrences of substring in the string.
endswith	Returns True if string ends with suffix.
startswith	Returns True if string starts with prefix.
join	Use string as delimiter for concatenating a sequence of other strings.
index	Return position of first character in substring if found in the string; raises ValueError if not found.
find	Return position of first character of <i>first</i> occurrence of substring in the string; like index, but returns -1 if not found.
rfind	Return position of first character of <i>last</i> occurrence of substring in the string; returns -1 if not found.
replace	Replace occurrences of string with another string.
strip, rstrip, lstrip	Trim whitespace, including newlines; equivalent to x.strip() (and rstrip, lstrip, respectively) for each element.
split	Break string into list of substrings using passed delimiter.
lower	Convert alphabet characters to lowercase.
upper	Convert alphabet characters to uppercase.
casefold	Convert characters to lowercase, and convert any region-specific variable character combinations to a common comparable form.
ljust, rjust	Left justify or right justify, respectively; pad opposite side of string with spaces (or some other fill character) to return a string with a minimum width.

## Düzenli ifadeler

```
In [118]: import re
```

```
In [119]: text = "foo bar\tbaz \tqux"
```

```
In [120]: re.split('\s+', text)
```

```
Out[120]: ['foo', 'bar', 'baz', 'qux']
```

- Regex kütüphanesi tanımlandı ve \s+ boşluk ifadesiyle elemanlar ayırdı.

- `re.split('\s+', text)` ögesini çağrıdığınızda, ilk olarak normal ifade derlenir ve daha sonra iletilen metinde split yöntemi çağrılır.

```
In [121]: regex = re.compile('\s+')
In [122]: regex.split(text)
Out[122]: ['foo', 'bar', 'baz', 'qux']
```

- Kalıpların listesi için findall yöntemi kullanılır.

```
In [123]: regex.findall(text)
Out[123]: [' ', '\t ', '\t']
```

- Bir regex nesnesi oluşturuldu. Buradaki mail adreslerinin listesi findall ile oluşturuldu.

```
In [124]: text = """Dave dave@google.com
Steve steve@gmail.com
Rob rob@gmail.com
Ryan ryan@yahoo.com
"""
pattern = r'[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}'
```

```
In [125]: regex = re.compile(pattern, flags=re.IGNORECASE)
In [126]: regex.findall(text)
Out[126]: ['dave@google.com', 'steve@gmail.com', 'rob@gmail.com', 'ryan@yahoo.com']
```

- Search, metindeki ilk e-posta adresi için özel bir eşleşme nesnesi döndürür. Regex'ten önceki eşleşme nesnesi bize yalnızca ögenin başlangıç ve bitiş konumunu söyleyebilir.

```
In [127]: m = regex.search(text)
In [128]: m
Out[128]: <re.Match object; span=(5, 20), match='dave@google.com'>
In [129]: text[m.start():m.end()]
Out[129]: 'dave@google.com'
```

```
In [130]: print(regex.match(text))
```

```
None
```

```
In [131]: print(regex.sub('REDACTED', text))
```

```
Dave REDACTED  
Steve REDACTED  
Rob REDACTED  
Ryan REDACTED
```

- regex.match, yalnızca desenin başında oluşması durumunda eşleşeceğini None değerini döndürür. Buna bağlı olarak sub, yerine geçen modelin oluşumlarıyla birlikte yeni bir dize döndürecektir.

```
In [134]: m = regex.match('wesm@bright.net')
```

```
In [135]: m.groups()
```

```
Out[135]: ('wesm', 'bright', 'net')
```

```
In [136]: regex.findall(text)
```

```
Out[136]: [('dave', 'google', 'com'),  
           ('steve', 'gmail', 'com'),  
           ('rob', 'gmail', 'com'),  
           ('ryan', 'yahoo', 'com')]
```

- Bu değiştirilmiş normal ifade tarafından üretilen bir eşleşme nesnesi, kalıp bileşeninin bir demetini döndürür ve groups yöntemiyle birleştirir. Desende (pattern) gruplar olduğunda findall onları geri döndürür.

```
In [137]: print(regex.sub(r'Username: \1, Domain: \2, Suffix: \3', text))
```

```
Dave Username: dave, Domain: google, Suffix: com  
Steve Username: steve, Domain: gmail, Suffix: com  
Rob Username: rob, Domain: gmail, Suffix: com  
Ryan Username: ryan, Domain: yahoo, Suffix: com
```

- sub ayrıca \1 ve \2 gibi özel sembollerini kullanarak her matchteki gruplara erişebilir.
- \1 simbolü ilk eşleşen grup, \2 ikinciye karşılık gelir.

## Düzenli İfade Yöntemleri

Argument	Description
<code>findall</code>	Return all non-overlapping matching patterns in a string as a list
<code>finditer</code>	Like <code>findall</code> , but returns an iterator
<code>match</code>	Match pattern at start of string and optionally segment pattern components into groups; if the pattern matches, returns a match object, and otherwise <code>None</code>
<code>search</code>	Scan string for match to pattern; returning a match object if so; unlike <code>match</code> , the match can be anywhere in the string as opposed to only at the beginning
<code>split</code>	Break string into pieces at each occurrence of pattern
<code>sub</code> , <code>subn</code>	Replace all ( <code>sub</code> ) or first <code>n</code> occurrences ( <code>subn</code> ) of pattern in string with replacement expression; use symbols <code>\1</code> , <code>\2</code> , ... to refer to match group elements in the replacement string

## Pandalarda Vektörleştirilmiş Dize İşlevleri

- Dağınık bir veri kümelerini analiz için temizlemek çoğu zaman çok fazla dize düzenlemeyi gerektirir. Series'in dizi odaklı dizge işlemlerine yönelik NA değerlerini atlayan yöntemleri vardır.

```
In [138]: data = {'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com',  
             'Rob': 'rob@gmail.com', 'Wes': np.nan}
```

```
In [139]: data = pd.Series(data)
```

```
In [140]: data
```

```
Out[140]: Dave      dave@google.com  
          Steve     steve@gmail.com  
          Rob       rob@gmail.com  
          Wes        NaN  
          dtype: object
```

```
In [141]: data.isnull()
```

```
Out[141]: Dave      False  
          Steve     False  
          Rob       False  
          Wes       True  
          dtype: bool
```

```
In [18]: data.str.contains('gmail')
```

```
Out[18]: Dave      False  
          Steve     True  
          Rob       True  
          Wes       NaN  
          dtype: object
```

## Vektörleştirilmiş Dize Yöntemlerinin Kısmi Listesi

Method	Description
cat	Concatenate strings element-wise with optional delimiter
contains	Return boolean array if each string contains pattern/regex
count	Count occurrences of pattern
extract	Use a regular expression with groups to extract one or more strings from a Series of strings; the result will be a DataFrame with one column per group
endswith	Equivalent to <code>x.endswith(pattern)</code> for each element
startswith	Equivalent to <code>x.startswith(pattern)</code> for each element
findall	Compute list of all occurrences of pattern/regex for each string
get	Index into each element (retrieve <i>i</i> -th element)
isalnum	Equivalent to built-in <code>str.isalnum</code>
isalpha	Equivalent to built-in <code>str.isalpha</code>
isdecimal	Equivalent to built-in <code>str.isdecimal</code>
isdigit	Equivalent to built-in <code>str.isdigit</code>
islower	Equivalent to built-in <code>str.islower</code>
isnumeric	Equivalent to built-in <code>str.isnumeric</code>
isupper	Equivalent to built-in <code>str.isupper</code>
join	Join strings in each element of the Series with passed separator
len	Compute length of each string
lower, upper	Convert cases; equivalent to <code>x.lower()</code> or <code>x.upper()</code> for each element
match	Use <code>re.match</code> with the passed regular expression on each element, returning matched groups as list
pad	Add whitespace to left, right, or both sides of strings
center	Equivalent to <code>pad(side='both')</code>
repeat	Duplicate values (e.g., <code>s.str.repeat(3)</code> is equivalent to <code>x * 3</code> for each string)
replace	Replace occurrences of pattern/regex with some other string
slice	Slice each string in the Series
split	Split strings on delimiter or regular expression
strip	Trim whitespace from both sides, including newlines
rstrip	Trim whitespace on right side
lstrip	Trim whitespace on left side

## Çözüm

Etkili veri hazırlığı, şunları yapmanızı sağlayarak verimliliği önemli ölçüde artırabilir:

Verileri analiz etmeye daha fazla, analize hazırlamaya daha az zaman ayırın.

## Veri Düzenleme: Katılın (Join) , Birleştirin (Combine) ve Yeniden Şekillendirin (Reshape)

### Hiyerarşik İndeksleme

- Hiyerarşik indeksleme, pandaların bir eksen üzerinde tiple (iki veya daha fazla) indeks düzeyine sahip olması önemli bir özelliğidir. Biraz soyut olarak, bir yol sağlar daha düşük boyutlu bir formda daha yüksek boyutlu verilerle çalışmanızı sağlar.

```
In [22]: data = pd.Series(np.random.randn(9),
                         index=[['a', 'a', 'a', 'b', 'b', 'c', 'c', 'd', 'd'],
                                [1, 2, 3, 1, 3, 1, 2, 2, 3]])
```

```
In [23]: data
```

```
Out[23]: a    1    0.558745
          2    1.219325
          3   -1.084630
         b    1    0.394612
          3    1.983104
         c    1    0.134731
          2   -2.339326
         d    2    0.532818
          3   -0.678821
        dtype: float64
```

```
In [24]: data.index
```

```
Out[24]: MultiIndex([( 'a', 1),
                      ( 'a', 2),
                      ( 'a', 3),
                      ( 'b', 1),
                      ( 'b', 3),
                      ( 'c', 1),
                      ( 'c', 2),
                      ( 'd', 2),
                      ( 'd', 3)],
                     )
```

```
In [25]: data['b']
```

```
Out[25]: 1    0.394612
          3    1.983104
        dtype: float64
```

```
In [26]: data['b':'c']
```

```
Out[26]: b    1    0.394612
          3    1.983104
          c    1    0.134731
          2   -2.339326
          dtype: float64
```

```
In [27]: data.loc[['b', 'd']]
```

```
Out[27]: b    1    0.394612
          3    1.983104
          d    2    0.532818
          3   -0.678821
          dtype: float64
```

- Dizenin indexleriyle ilgili işlemler yapılmıştır.
- Hiyerarşik indeksleme, verilerin yeniden şekillendirilmesinde ve grup bazlı pivot tablo oluşturma gibi işlemlerde önemli bir rol oynar. Örneğin, verileri unstack yöntemini kullanarak yeniden düzenleyebilirsiniz.

```
In [29]: data.unstack()
```

```
Out[29]:
```

	1	2	3
a	0.558745	1.219325	-1.084630
b	0.394612	NaN	1.983104
c	0.134731	-2.339326	NaN
d	NaN	0.532818	-0.678821

```
In [30]: data.unstack().stack()
```

```
Out[30]: a    1    0.558745
          2    1.219325
          3   -1.084630
          b    1    0.394612
          3    1.983104
          c    1    0.134731
          2   -2.339326
          d    2    0.532818
          3   -0.678821
          dtype: float64
```

•

```
In [31]: frame = pd.DataFrame(np.arange(12).reshape((4, 3)),  
                           index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],  
                           columns[['Ohio', 'Ohio', 'Colorado'],  
                                    ['Green', 'Red', 'Green']])
```

```
In [32]: frame
```

Out[32]:

	Ohio	Colorado		
	Green	Red	Green	
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

```
In [36]: frame.index.names = ['key1', 'key2']
```

```
In [37]: frame.columns.names = ['state', 'color']
```

```
In [38]: frame
```

Out[38]:

	state	Ohio	Colorado	
	color	Green	Red	Green
key1	key2			
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

```
In [39]: frame['Ohio']
```

Out[39]:

	color	Green	Red
key1	key2		
a	1	0	1
	2	3	4
b	1	6	7
	2	9	10

```
In [40]: MultiIndex.from_arrays([['Ohio', 'Ohio', 'Colorado'], ['Green', 'Red', 'Green']],  
                           names=['state', 'color'])
```

- MultiIndex ile de aynı DataFrame yapılabilir.

## Yeniden sıralama ve Sıralama Düzeyleri

- Bazen bir eksendeki belirli bir seviyedeki değerlere göre düzeylerin sırasını yeniden düzenlemeniz veya verileri sıralamanız gerekebilir.

```
In [41]: frame.swaplevel('key1', 'key2')
```

Out[41]:

	state	Ohio	Colorado	
	color	Green	Red	Green
key2	key1			
1	a	0	1	2
2	a	3	4	5
1	b	6	7	8
2	b	9	10	11

- Swaplevel iki seviye numarası veya adı alır ve seviyeleri değiştirilen yeni bir nesne döndürür.
- sort\_index ise verileri yalnızca tek bir düzeydeki değerleri kullanarak sıralar.

```
In [42]: frame.sort_index(level=1)
```

Out[42]:

	state	Ohio	Colorado	
	color	Green	Red	Green
key1	key2			
a	1	0	1	2
b	1	6	7	8
a	2	3	4	5
b	2	9	10	11

```
In [43]: frame.swaplevel(0, 1).sort_index(level=0)
```

Out[43]:

	state	Ohio	Colorado	
	color	Green	Red	Green
key2	key1			
1	a	0	1	2
	b	6	7	8
2	a	3	4	5
	b	9	10	11

## Seviyeye Göre Özet İstatistikler

- DataFrame ve Serilerdeki pek çok tanımlayıcı ve özet istatistiğin bir düzeyi vardır. Belirli bir kategoride toplamak istediğiniz düzeyi belirleyebileceğiniz seçenek eksenlerdir.

```
In [27]: frame.sum(level='key2')
```

```
Out[27]:
```

```
state    Ohio      Colorado
color   Green   Red     Green
key2
1           6     8       10
2          12    14       16
```

```
In [28]: frame.sum(level='color', axis=1)
```

```
Out[28]:
```

```
color      Green   Red
key1 key2
a        1       2     1
         2       8     4
b        1      14     7
         2      20    10
```

## DataFrame'in Sütunlarıyla İndeksleme

- Satır olarak DataFrame'deki bir veya daha fazla sütunu kullanmak alışılmadık bir durum değildir. İndeks; alternatif olarak satır indeksini DataFrame sütununa taşımak isteyebilirsiniz.

```
In [46]: frame = pd.DataFrame({'a': range(7), 'b': range(7, 0, -1),
                               'c': ['one', 'one', 'one', 'two', 'two',
                                     'two', 'two'],
                               'd': [0, 1, 2, 0, 1, 2, 3]})
```

```
In [47]: frame
```

```
Out[47]:
```

	a	b	c	d
0	0	7	one	0
1	1	6	one	1
2	2	5	one	2
3	3	4	two	0
4	4	3	two	1
5	5	2	two	2
6	6	1	two	3

- DataFrame'in set\_index işlevi, bir veya daha fazlasını kullanarak yeni bir DataFrame oluşturacaktır.

```
In [48]: frame2 = frame.set_index(['c', 'd'])
```

```
In [49]: frame2
```

```
Out[49]:
```

	a	b	
c	d		
one	0	0	7
	1	1	6
	2	2	5
two	0	3	4
	1	4	3
	2	5	2
	3	6	1

```
In [50]: frame.set_index(['c', 'd'], drop=False)
```

```
Out[50]:
```

	a	b	c	d
c	d			
one	0	0	7	one 0
	1	1	6	one 1
	2	2	5	one 2
two	0	3	4	two 0
	1	4	3	two 1
	2	5	2	two 2
	3	6	1	two 3

- Varsayılan olarak sütunlar DataFrame'den kaldırılır ancak yine de onları bırakabilirsiniz.

```
In [52]: frame2.reset_index()
```

```
Out[52]:
```

	c	d	a	b
0	one	0	0	7
1	one	1	1	6
2	one	2	2	5
3	two	0	3	4
4	two	1	4	3
5	two	2	5	2
6	two	3	6	1

- reset\_index ise set\_index'in tersini yapar; hiyerarşik dizin düzeylerini sütunlara taşıır.

## Veri Kümelerini Birleştirme (Combining and Merging)

- Panda nesnelerinin içeriği veriler çeşitli yollarla birleştirilebilir:
- pandas.merge, DataFrame'lerdeki satırları bir veya daha fazla anahtara dayalı olarak bağlar. Bu uyguladığı SQL veya veri tabanı join işlemleri gibi diğer ilişkisel veri tabanlarının kullanıcılarına aşina olacaktır.
- pandas.concat nesneleri bir eksen boyunca birleştirir veya "yığar (stacks)".
- Combine\_first örnek yöntemi, bir nesnedeki eksik değerleri diğerindeki değerlerle doldurmak için çakışan verilerin birbirine eklenmesini sağlar.

## Veri Tabanı Stili | DataFrame Birleşimleri

- Merge veya Join işlemleri, bir veya daha fazla anahtar kullanarak satırları birbirine bağlayarak veri kümelerini birleştirir. Bu işlemler ilişkisel veri tabanlarının (örneğin, SQL tabanlı) merkezinde yer alır.

```
In [53]: df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                           'data1': range(7)})
```

```
In [54]: df2 = pd.DataFrame({'key': ['a', 'b', 'd'],
                           'data2': range(3)})
```

```
In [55]: df1
```

```
Out[55]:
```

key	data1
0	0
1	1
2	2
3	3
4	4
5	5
6	6

```
In [56]: df2
```

```
Out[56]:
```

key	data2
0	0
1	1
2	2

```
In [57]: pd.merge(df1, df2)
```

```
Out[57]:
```

key	data1	data2
0	0	1
1	1	1
2	6	1
3	2	0
4	4	0
5	5	0

- İki DataFrame'in ortak değerleri birleştirilir.

```
In [58]: pd.merge(df1, df2, on='key')
```

- Bu gösterim de aynı sonucu verecektir.

```
In [59]: df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                           'data1': range(7)})
```

```
In [60]: df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'],
                           'data2': range(3)})
```

```
In [61]: pd.merge(df3, df4, left_on='lkey', right_on='rkey')
```

Out[61]:

	lkey	data1	rkey	data2
0	b	0	b	1
1	b	1	b	1
2	b	6	b	1
3	a	2	a	0
4	a	4	a	0
5	a	5	a	0

- Sütun adları her nesnede farklısa bunları ayrı ayrı belirtebilirsiniz.
- Sonuç olarak 'c' ve 'd' değerlerinin ve ilgili verilerin eksik olduğunu fark edebilirsiniz. Varsayılan olarak birleştirme (merge) 'iç' birleştirme yapar; sonuçtaki anahtarlar kesişen anahtarlar veya her iki tabloda bulunan ortak kümelerdir. Diğer olası seçenekler 'left', 'right' ve 'outer'dır .

```
In [62]: pd.merge(df1, df2, how='outer')
```

Out[62]:

	key	data1	data2
0	b	0.0	1.0
1	b	1.0	1.0
2	b	6.0	1.0
3	a	2.0	0.0
4	a	4.0	0.0
5	a	5.0	0.0
6	c	3.0	NaN
7	d	NaN	2.0

## Nasıl Argümanıyla Farklı Birleştirme Türleri

Option	Behavior
'inner'	Use only the key combinations observed in both tables
'left'	Use all key combinations found in the left table
'right'	Use all key combinations found in the right table
'outer'	Use all key combinations observed in both tables together

```
In [65]: df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                           'data1': range(6)})
```

```
In [66]: df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],
                           'data2': range(5)})
```

```
In [69]: pd.merge(df1, df2, on='key', how='left')
```

```
Out[69]:
```

	key	data1	data2
0	b	0	1.0
1	b	0	3.0
2	b	1	1.0
3	b	1	3.0
4	a	2	0.0
5	a	2	2.0
6	c	3	NaN
7	a	4	0.0
8	a	4	2.0
9	b	5	1.0
10	b	5	3.0

```
In [70]: pd.merge(df1, df2, how='inner')
```

```
Out[70]:
```

	key	data1	data2
0	b	0	1
1	b	0	3
2	b	1	1
3	b	1	3
4	b	5	1
5	b	5	3
6	a	2	0
7	a	2	2
8	a	4	0
9	a	4	2

- Çoka çoğu birleştirmeler satırların Kartezyen çarpımını oluşturur. Sol DataFrame'de üç tane 'b' satırları olduğundan ve sağdakinde iki satır vardır. Sonuç olarak burada altı 'b' satırı vardır. Birleştirme yöntemi yalnızca sonuçta görünen farklı anahtar değerleri etkiler.

```
In [71]: left = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
                           'key2': ['one', 'two', 'one'],
                           'lval': [1, 2, 3]})
```

```
In [72]: right = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
                            'key2': ['one', 'one', 'one', 'two'],
                            'rval': [4, 5, 6, 7]})
```

```
In [73]: pd.merge(left, right, on=['key1', 'key2'], how='outer')
```

```
Out[73]:
```

	key1	key2	lval	rval
0	foo	one	1.0	4.0
1	foo	one	1.0	5.0
2	foo	two	2.0	NaN
3	bar	one	3.0	6.0
4	bar	two	NaN	7.0

- Birleştirme işlemlerinde dikkate alınması gereken son konu, örtüsen sütunlarının işlenmesidir.

```
In [74]: pd.merge(left, right, on='key1')
```

```
Out[74]:
```

	key1	key2_x	lval	key2_y	rval
0	foo	one	1	one	4
1	foo	one	1	one	5
2	foo	two	2	one	4
3	foo	two	2	one	5
4	bar	one	3	one	6
5	bar	one	3	two	7

```
In [75]: pd.merge(left, right, on='key1', suffixes=('_left', '_right'))
```

```
Out[75]:
```

	key1	key2_left	lval	key2_right	rval
0	foo	one	1	one	4
1	foo	one	1	one	5
2	foo	two	2	one	4
3	foo	two	2	one	5
4	bar	one	3	one	6
5	bar	one	3	two	7

## Merge Fonksiyonları

Argument	Description
left	DataFrame to be merged on the left side.
right	DataFrame to be merged on the right side.
how	One of 'inner', 'outer', 'left', or 'right'; defaults to 'inner'.
on	Column names to join on. Must be found in both DataFrame objects. If not specified and no other join keys given, will use the intersection of the column names in <code>left</code> and <code>right</code> as the join keys.
left_on	Columns in <code>left</code> DataFrame to use as join keys.
right_on	Analogous to <code>left_on</code> for <code>left</code> DataFrame.
left_index	Use row index in <code>left</code> as its join key (or keys, if a MultiIndex).
right_index	Analogous to <code>left_index</code> .
sort	Sort merged data lexicographically by join keys; <code>True</code> by default (disable to get better performance in some cases on large datasets).
suffixes	Tuple of string values to append to column names in case of overlap; defaults to ('_x', '_y') (e.g., if 'data' in both DataFrame objects, would appear as 'data_x' and 'data_y' in result).
copy	If <code>False</code> , avoid copying data into resulting data structure in some exceptional cases; by default always copies.
indicator	Adds a special column <code>_merge</code> that indicates the source of each row; values will be 'left_only', 'right_only', or 'both' based on the origin of the joined data in each row.

## Index'te Birleştirme

```
In [4]: left1 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'],
                           'value': range(6)})
```

```
In [5]: right1 = pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
```

```
In [6]: left1
```

```
out[6]:
```

	key	value
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

```
In [7]: right1
```

```
out[7]:
```

	group_val
a	3.5
b	7.0

```
In [8]: pd.merge(left1, right1, left_on='key', right_index=True)
```

```
Out[8]:
```

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0

```
In [9]: pd.merge(left1, right1, left_on='key', right_index=True, how='outer')
```

```
Out[9]:
```

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0
5	c	5	NaN

```
In [10]: left1 = pd.DataFrame({'key1': ['Ohio', 'Ohio', 'Ohio',
                                         'Nevada', 'Nevada'],
                               'key2': [2000, 2001, 2002, 2001, 2002],
                               'data': np.arange(5.)})
```

```
In [11]: right1 = pd.DataFrame(np.arange(12).reshape((6, 2)),
                             index=[['Nevada', 'Nevada', 'Ohio', 'Ohio',
                                     'Ohio', 'Ohio'],
                                     [2001, 2000, 2000, 2000, 2001, 2002]],
                             columns=['event1', 'event2'])
```

```
In [12]: left1
```

```
Out[12]:
```

	key1	key2	data
0	Ohio	2000	0.0
1	Ohio	2001	1.0
2	Ohio	2002	2.0
3	Nevada	2001	3.0
4	Nevada	2002	4.0

```
In [13]: right1
```

```
Out[13]:
```

		event1	event2
Nevada	2001	0	1
	2000	2	3
Ohio	2000	4	5
	2000	6	7
	2001	8	9
	2002	10	11

- Hiyerarşik olarak indekslenmiş verilerde işler daha karmaşıktır, çünkü indekse katılmak daha zordur. Bu durumda, birleştirilecek birden fazla sütunu liste olarak belirtmeniz gereklidir.

```
In [14]: pd.merge(lefth, righth, left_on=['key1', 'key2'], right_index=True)
```

Out[14]:

	key1	key2	data	event1	event2
0	Ohio	2000	0.0	4	5
0	Ohio	2000	0.0	6	7
1	Ohio	2001	1.0	8	9
2	Ohio	2002	2.0	10	11
3	Nevada	2001	3.0	0	1

```
In [15]: pd.merge(lefth, righth, left_on=['key1', 'key2'],
                 right_index=True, how='outer')
```

Out[15]:

	key1	key2	data	event1	event2
0	Ohio	2000	0.0	4.0	5.0
0	Ohio	2000	0.0	6.0	7.0
1	Ohio	2001	1.0	8.0	9.0
2	Ohio	2002	2.0	10.0	11.0
3	Nevada	2001	3.0	0.0	1.0
4	Nevada	2002	4.0	NaN	NaN
4	Nevada	2000	NaN	2.0	3.0

- Birleştirmenin her iki tarafının indekslerini kullanmak da mümkündür.

```
In [16]: left2 = pd.DataFrame([[1., 2.], [3., 4.], [5., 6.]],
                           index=['a', 'c', 'e'],
                           columns=['Ohio', 'Nevada'])
```

In [18]: left2

Out[18]:

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

```
In [17]: right2 = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [13., 14.]],  
                           index=['b', 'c', 'd', 'e'],  
                           columns=['Missouri', 'Alabama'])
```

```
In [19]: right2
```

```
Out[19]:
```

	Missouri	Alabama
b	7.0	8.0
c	9.0	10.0
d	11.0	12.0
e	13.0	14.0

- Bu birleştirme şekliyle tüm sütunlar değerleriyle birlikte birleştirilir.

```
In [20]: left2.join(right2, how='outer')
```

```
Out[20]:
```

	Ohio	Nevada	Missouri	Alabama
a	1.0	2.0	NaN	NaN
b	NaN	NaN	7.0	8.0
c	3.0	4.0	9.0	10.0
d	NaN	NaN	11.0	12.0
e	5.0	6.0	13.0	14.0

- Son olarak, basit indeks üzerinde indeks birleştirimi için, birleştirilecek DataFrames listesini iletебilirsiniz.

```
In [21]: another = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [16., 17.]],  
                           index=['a', 'c', 'e', 'f'],  
                           columns=['New York', 'Oregon'])
```

```
In [22]: another
```

```
Out[22]:
```

	New York	Oregon
a	7.0	8.0
c	9.0	10.0
e	11.0	12.0
f	16.0	17.0

```
In [23]: left2.join([right2, another])
```

Out[23]:

	Ohio	Nevada	Missouri	Alabama	New York	Oregon
a	1.0	2.0	NaN	NaN	7.0	8.0
c	3.0	4.0	9.0	10.0	9.0	10.0
e	5.0	6.0	13.0	14.0	11.0	12.0

```
In [24]: left2.join([right2, another], how='outer')
```

Out[24]:

	Ohio	Nevada	Missouri	Alabama	New York	Oregon
a	1.0	2.0	NaN	NaN	7.0	8.0
c	3.0	4.0	9.0	10.0	9.0	10.0
e	5.0	6.0	13.0	14.0	11.0	12.0
b	NaN	NaN	7.0	8.0	NaN	NaN
d	NaN	NaN	11.0	12.0	NaN	NaN
f	NaN	NaN	NaN	NaN	16.0	17.0

## Bir Eksen Boyunca Birleştirme

```
In [25]: arr = np.arange(12).reshape((3, 4))
```

```
In [26]: arr
```

```
Out[26]: array([[ 0,  1,  2,  3],
   [ 4,  5,  6,  7],
   [ 8,  9, 10, 11]])
```

```
In [27]: np.concatenate([arr, arr], axis=1)
```

```
Out[27]: array([[ 0,  1,  2,  3,  0,  1,  2,  3],
   [ 4,  5,  6,  7,  4,  5,  6,  7],
   [ 8,  9, 10, 11,  8,  9, 10, 11]])
```

- Başka bir veri birleştirme işlemi türü birbirinin yerine concat olarak anılır.
- Eksenleri etiketlenmiş olan Series ve DataFrame gibi pandas nesneleri bağlamında dizi birleştirmeyi daha da genelleştirmenizi sağlar.
- Nesneler diğer eksenlerde farklı şekilde indekslenmişse, bunları birleştirmeli miyiz bu eksenlerdeki farklı öğeler mi yoksa yalnızca paylaşılan değerleri mi (the intersection) kullanıyor?

- Birleştirilmiş veri parçalarının sonuçta tanımlanabilir olması gerekiyor mu?
- “Birleştirme ekseni” korunması gereken verileri içeriyor mu?
  - Pandalardaki concat işlevi bunların her birini ele almak için tutarlı bir yol sağlar.

```
In [28]: s1 = pd.Series([0, 1], index=['a', 'b'])
```

```
In [29]: s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])
```

```
In [30]: s3 = pd.Series([5, 6], index=['f', 'g'])
```

```
In [31]: pd.concat([s1, s2, s3])
```

```
Out[31]: a    0
          b    1
          c    2
          d    3
          e    4
          f    5
          g    6
          dtype: int64
```

```
In [32]: pd.concat([s1, s2, s3], axis=1)
```

```
Out[32]:
      0   1   2
a  0.0  NaN  NaN
b  1.0  NaN  NaN
c  NaN  2.0  NaN
d  NaN  3.0  NaN
e  NaN  4.0  NaN
f  NaN  NaN  5.0
g  NaN  NaN  6.0
```

```
In [33]: s4 = pd.concat([s1, s3])
```

```
In [34]: s4
```

```
Out[34]: a    0
          b    1
          f    5
          g    6
          dtype: int64
```

```
In [35]: pd.concat([s1, s4], axis=1)
```

```
Out[35]:
```

	0	1
a	0.0	0
b	1.0	1
f	NaN	5
g	NaN	6

- Bu son örnekte, join='inner' nedeniyle 'f' ve 'g' etiketleri kayboldu.

```
In [36]: pd.concat([s1, s4], axis=1, join='inner')
```

```
Out[36]:
```

	0	1
a	0	0
b	1	1

- Serilerin eksen=1 boyunca birleştirilmesi durumunda anahtarlar DataFrame sütun başlıklarını haline gelir.

```
In [37]: pd.concat([s1, s2, s3], axis=1, keys=['one', 'two', 'three'])
```

Out[37]:

	one	two	three
a	0.0	NaN	NaN
b	1.0	NaN	NaN
c	NaN	2.0	NaN
d	NaN	3.0	NaN
e	NaN	4.0	NaN
f	NaN	NaN	5.0
g	NaN	NaN	6.0

- Aynı mantık DataFrame nesnelerine de uygulanır.

```
In [38]: df1 = pd.DataFrame(np.arange(6).reshape(3, 2), index=['a', 'b', 'c'],
columns=['one', 'two'])
```

```
In [39]: df2 = pd.DataFrame(5 + np.arange(4).reshape(2, 2), index=['a', 'c'],
columns=['three', 'four'])
```

```
In [40]: df1
```

Out[40]:

	one	two
a	0	1
b	2	3
c	4	5

```
In [41]: df2
```

Out[41]:

	three	four
a	5	6
c	7	8

```
In [42]: pd.concat([df1, df2], axis=1, keys=['level1', 'level2'])
```

Out[42]:

	level1		level2	
	one	two	three	four
a	0	1	5.0	6.0
b	2	3	NaN	NaN
c	4	5	7.0	8.0

- Bir liste yerine nesnelerin bir dict'ini iletişeniz, key'ler için dict'in key'leri kullanılacaktır.
- Hiyerarşik index'lerin nasıl oluşturulduğunu belirleyen ek argümanlar vardır.

```
In [43]: pd.concat({'level1': df1, 'level2': df2}, axis=1)
```

Out[43]:

	level1		level2	
	one	two	three	four
a	0	1	5.0	6.0
b	2	3	NaN	NaN
c	4	5	7.0	8.0

```
In [44]: pd.concat([df1, df2], axis=1, keys=['level1', 'level2'],
                 names=['upper', 'lower'])
```

Out[44]:

	upper	level1		level2	
	lower	one	two	three	four
a	0	1	5.0	6.0	
b	2	3	NaN	NaN	
c	4	5	7.0	8.0	

```
In [45]: df1 = pd.DataFrame(np.random.randn(3, 4), columns=['a', 'b', 'c', 'd'])
```

```
In [46]: df2 = pd.DataFrame(np.random.randn(2, 3), columns=['b', 'd', 'a'])
```

In [47]: df1

Out[47]:

	a	b	c	d
0	1.133696	-1.949646	1.196921	2.619263
1	-0.535513	1.497089	-1.495896	-0.326038
2	-1.671339	-0.023965	-0.059736	-0.819926

In [48]: df2

Out[48]:

	b	d	a
0	-0.143653	0.930544	0.443620
1	0.657675	-1.976482	0.291221

- Dikkate alınması gereken son nokta, satır indeksinin içermediği DataFrame'lerle ilgilidir.

```
In [49]: pd.concat([df1, df2], ignore_index=True)
```

Out[49]:

	a	b	c	d
0	1.133696	-1.949646	1.196921	2.619263
1	-0.535513	1.497089	-1.495896	-0.326038
2	-1.671339	-0.023965	-0.059736	-0.819926
3	0.443620	-0.143653	NaN	0.930544
4	0.291221	0.657675	NaN	-1.976482

## Concat Fonksiyonları

Argument	Description
objs	List or dict of pandas objects to be concatenated; this is the only required argument
axis	Axis to concatenate along; defaults to 0 (along rows)
join	Either 'inner' or 'outer' ('outer' by default); whether to intersection (inner) or union (outer) together indexes along the other axes
join_axes	Specific indexes to use for the other $n-1$ axes instead of performing union/intersection logic
keys	Values to associate with objects being concatenated, forming a hierarchical index along the concatenation axis; can either be a list or array of arbitrary values, an array of tuples, or a list of arrays (if multiple-level arrays passed in levels)
levels	Specific indexes to use as hierarchical index level or levels if keys passed
names	Names for created hierarchical levels if keys and/or levels passed
verify_integrity	Check new axis in concatenated object for duplicates and raise exception if so; by default (False) allows duplicates
ignore_index	Do not preserve indexes along concatenation axis, instead producing a new range(total_length) index

## Verileri Örtüşmeyle Birleştirme

```
In [50]: a = pd.Series([np.nan, 2.5, np.nan, 3.5, 4.5, np.nan],  
                      index=['f', 'e', 'd', 'c', 'b', 'a'])
```

```
In [51]: b = pd.Series(np.arange(len(a)), dtype=np.float64),  
                      index=['f', 'e', 'd', 'c', 'b', 'a'])
```

```
In [52]: b[-1] = np.nan
```

```
In [53]: a
```

```
Out[53]: f      NaN  
         e      2.5  
         d      NaN  
         c      3.5  
         b      4.5  
         a      NaN  
        dtype: float64
```

```
In [54]: b
```

```
Out[54]: f      0.0  
         e      1.0  
         d      2.0  
         c      3.0  
         b      4.0  
         a      NaN  
        dtype: float64
```

```
In [55]: np.where(pd.isnull(a), b, a)
```

```
Out[55]: array([0. , 2.5, 2. , 3.5, 4.5, nan])
```

- Series, pandaların olağan veri hizalama mantığıyla birlikte bu işlemin eşdeğerini gerçekleştiriren bir `combine_first` yöntemine sahiptir.

```
In [56]: b[:-2].combine_first(a[2:])
```

```
Out[56]: a    NaN  
         b    4.5  
         c    3.0  
         d    2.0  
         e    1.0  
         f    0.0  
        dtype: float64
```

- DataFrames ile, `combine_first` aynı şeyi sütun sütun yapar, böylece bunu, çağrıran nesnedeki eksik verileri, nesneden gelen verilerle "yama (patching)" olarak düşünebiliriz.

```
In [57]: df1 = pd.DataFrame({'a': [1., np.nan, 5., np.nan],  
                           'b': [np.nan, 2., np.nan, 6.],  
                           'c': range(2, 18, 4)})
```

```
In [58]: df2 = pd.DataFrame({'a': [5., 4., np.nan, 3., 7.],  
                           'b': [np.nan, 3., 4., 6., 8.]})
```

```
In [59]: df1
```

```
Out[59]:
```

	a	b	c
0	1.0	NaN	2
1	NaN	2.0	6
2	5.0	NaN	10
3	NaN	6.0	14

```
In [62]: df2
```

```
Out[62]:
```

	a	b
0	5.0	NaN
1	4.0	3.0
2	NaN	4.0
3	3.0	6.0
4	7.0	8.0

```
In [61]: df1.combine_first(df2)
```

```
Out[61]:
```

	a	b	c
0	1.0	NaN	2.0
1	4.0	2.0	6.0
2	5.0	4.0	10.0
3	3.0	6.0	14.0
4	7.0	8.0	NaN

## Yeniden Şekillendirme (Reshaping) ve Döndürme (Pivoting)

- Tablo verilerini yeniden düzenlemek için bir dizi temel işlem vardır. Bunlar başka doğal olarak yeniden şekillendirme veya pivot işlemleri olarak anılır.

### Hiyerarşik İndeksleme ile Yeniden Şekillendirme

- Hiyerarşik indeksleme, DataFrame'deki verileri yeniden düzenlemek için tutarlı bir yol sağlar.

İki temel eylem vardır:

#### Stack

- Verileri kolonlardan satırlara döndürür.

#### Unstack

- Verileri satırlardan kolonlara döndürür.

```
In [4]: data = pd.DataFrame(np.arange(6).reshape((2, 3)),  
                           index=pd.Index(['Ohio', 'Colorado'], name='state'),  
                           columns=pd.Index(['one', 'two', 'three'],  
                                         name='number'))
```

```
In [5]: data
```

```
Out[5]:
```

	number	one	two	three
state				
Ohio	0	1	2	
Colorado	3	4	5	

```
In [6]: result = data.stack()
```

```
In [7]: result
```

```
Out[7]: state      number
          Ohio       one      0
                      two      1
                      three     2
          Colorado   one      3
                      two      4
                      three     5
        dtype: int32
```

```
In [8]: result.unstack()
```

```
Out[8]:      number  one  two  three
              state
              Ohio    0    1    2
Colorado    3    4    5
```

- Varsayılan olarak en içteki düzey yiğinsizdir (yiğinla aynı). Bir seviye numarası veya adı belirterek farklı bir seviyeye ulaşabilirsiniz.

```
In [9]: result.unstack(0)
```

```
Out[9]:      state  Ohio  Colorado
              number
              one    0    3
              two    1    4
              three  2    5
```

```
In [10]: result.unstack('state')
```

```
Out[10]:      state  Ohio  Colorado
              number
              one    0    3
              two    1    4
              three  2    5
```

```
In [7]: s1 = pd.Series([0, 1, 2, 3], index=['a', 'b', 'c', 'd'])
```

```
In [8]: s2 = pd.Series([4, 5, 6], index=['c', 'd', 'e'])
```

```
In [9]: data2 = pd.concat([s1, s2], keys=['one', 'two'])
```

```
In [10]: data2
```

```
Out[10]: one   a    0  
          b    1  
          c    2  
          d    3  
         two   c    4  
                  d    5  
                  e    6  
dtype: int64
```

```
In [11]: data2.unstack()
```

```
Out[11]:
```

	a	b	c	d	e
one	0.0	1.0	2.0	3.0	NaN
two	NaN	NaN	4.0	5.0	6.0

- Alt grupların her birinde düzeydeki değerlerin tümü bulunamazsa yığının kaldırılması eksik verilere neden olabilir.
- Yığınlama (Stacking), varsayılan olarak eksik verileri filtreler, böylece işlem daha kolay tersine çevrilebilir

```
In [13]: data2.unstack().stack()
```

```
Out[13]: one   a    0.0  
          b    1.0  
          c    2.0  
          d    3.0  
         two   c    4.0  
                  d    5.0  
                  e    6.0  
dtype: float64
```

```
In [14]: data2.unstack().stack(dropna=False)
```

```
Out[14]: one   a    0.0  
          b    1.0  
          c    2.0  
          d    3.0  
          e    NaN  
         two   a    NaN  
                  b    NaN  
                  c    4.0  
                  d    5.0  
                  e    6.0  
dtype: float64
```

- Bir DataFrame'de yiğini kaldırığınızda, yiğinden çıkarılan düzey, içindeki en düşük düzey olur.

```
In [31]: df = pd.DataFrame({'left': result, 'right': result + 5},
columns=pd.Index(['left', 'right'], name='side'))
```

```
In [32]: df
```

Out[32]:

	side	left	right
state	number		
Ohio	one	0	5
	two	1	6
	three	2	7
Colorado	one	3	8
	two	4	9
	three	5	10

```
In [33]: df.unstack('state')
```

Out[33]:

	side	left		right	
state		Ohio	Colorado	Ohio	Colorado
number					
one		0		3	5
two		1		4	6
three		2		5	7
					10

- Stack'i çağrıırken, stack yapılacak eksenin adını belirtebiliriz.

```
In [34]: df.unstack('state').stack('side')
```

Out[34]:

	state	Colorado	Ohio
number	side		
one	left	3	0
	right	8	5
two	left	4	1
	right	9	6
three	left	5	2
	right	10	7

## “Uzun” Formattan “Geniş” Formata Döndürme

- Veri tabanlarında ve CSV'de veya stack formatında birden çok zaman serisini saklamadan yaygın bir yol, uzun zaman dizileridir.

## “Geniş” Formattan “Uzun” Formata Döndürme

- DataFrameler için pivot işleminin tersi bir işlem pandas.melt'tir. Aktarmak yerine yeni bir DataFrame'de bir sütunu birçok sütuna dönüştürerek birden çok sütunu birleştirir.

```
In [62]: df = pd.DataFrame({'key': ['foo', 'bar', 'baz'],
                           'A': [1, 2, 3],
                           'B': [4, 5, 6],
                           'C': [7, 8, 9]})
```

```
In [63]: df
```

```
Out[63]:
```

	key	A	B	C
0	foo	1	4	7
1	bar	2	5	8
2	baz	3	6	9

```
In [64]: melted = pd.melt(df, ['key'])
```

```
In [65]: melted
```

```
Out[65]:
```

	key	variable	value
0	foo	A	1
1	bar	A	2
2	baz	A	3
3	foo	B	4
4	bar	B	5
5	baz	B	6
6	foo	C	7
7	bar	C	8
8	baz	C	9

- 'key' sütunu bir grup göstergesi olabilir ve diğer sütunlar veri değerleridir. Pandas.melt'i kullanırken hangi sütunların (varsayımsa) grup göstergesi olduğunu belirtmeliyiz.

- Pivot kullanarak orijinal DataFrame'e geri dönebiliriz:

```
In [75]: reshaped = melted.pivot('key', 'variable', 'value')
```

```
In [76]: reshaped
```

```
Out[76]:
variable  A  B  C
key
bar      2  5  8
baz      3  6  9
foo      1  4  7
```

- Pivot sonucu satır etiketleri olarak kullanılan sütundan bir dizin oluşturduğundan, verileri tekrar bir sütuna taşımak için reset\_index kullanılabilir.

```
In [77]: reshaped.reset_index()
```

```
Out[77]:
```

	variable	key	A	B	C
0	bar	2	5	8	
1	baz	3	6	9	
2	foo	1	4	7	

- Değer sütunları olarak kullanılacak bir sütun alt kümesi de belirtebilirsiniz.

```
In [78]: pd.melt(df, id_vars=['key'], value_vars=['A', 'B'])
```

```
Out[78]:
```

	key	variable	value
0	foo	A	1
1	bar	A	2
2	baz	A	3
3	foo	B	4
4	bar	B	5
5	baz	B	6

- pandas.melt herhangi bir grup tanımlayıcısı olmadan da kullanılabilir.

```
In [79]: pd.melt(df, value_vars=['A', 'B', 'C'])
```

```
out[79]:
```

	variable	value
0	A	1
1	A	2
2	A	3
3	B	4
4	B	5
5	B	6
6	C	7
7	C	8
8	C	9

```
In [80]: pd.melt(df, value_vars=['key', 'A', 'B'])
```

```
out[80]:
```

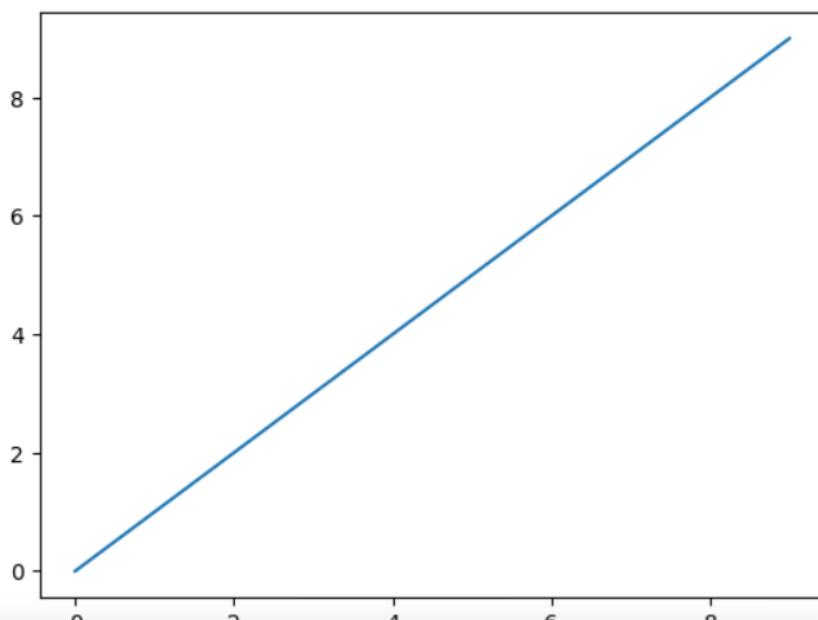
	variable	value
0	key	foo
1	key	bar
2	key	baz
3	A	1
4	A	2
5	A	3
6	B	4
7	B	5
8	B	6

# Çizim ve Görselleştirme

- Bilgilendirici görselleştirmeler yapmak (bazen olay örgüsü de denir) en önemli konulardan biridir. Veri analizinde önemli görevleri vardır. Araştırma sürecinin bir parçası olabilir; örneğin, Aykırı değerlerin veya gerekli veri dönüşümlerinin belirlenmesine yardımcı olmak veya veri oluşturmanın bir yolu olarak modeller için fikir verir.
- Python'un statik veya dinamik görseller oluşturmak için birçok ekleni kütüphanesi vardır ama esas olarak matplotlib ve onun üzerine inşa edilen kütüphaneler daha çok kullanılır.
- Matplotlib, (çoğunlukla iki boyutlu) yayın kalitesinde grafikler oluşturmak için tasarlanmış bir masaüstü çizim paketidir. Matplotlib, hepsinde çeşitli GUI arka uçlarını destekler. İşletim sistemlerine ek olarak görselleştirmeleri tüm ortak uygulamalara vektör ve raster grafik formatları ile aktarabilir(PDF, SVG, JPG, PNG, BMP, GIF vb.).

## Kısa Bir Matplotlib API Başlangıç Bilgisi

```
In [81]: import matplotlib.pyplot as plt  
In [82]: import numpy as np  
In [83]: data = np.arange(10)  
In [84]: data  
In [85]: plt.plot(data)  
Out[85]: [
```



## Şekiller ve Alt Grafikler

- Matplotlib'deki grafikler bir Figure nesnesinin içinde bulunur. plt.figure ile yeni bir figür oluşturabilirsiniz.

```
In [86]: fig = plt.figure()  
<Figure size 640x480 with 0 Axes>
```

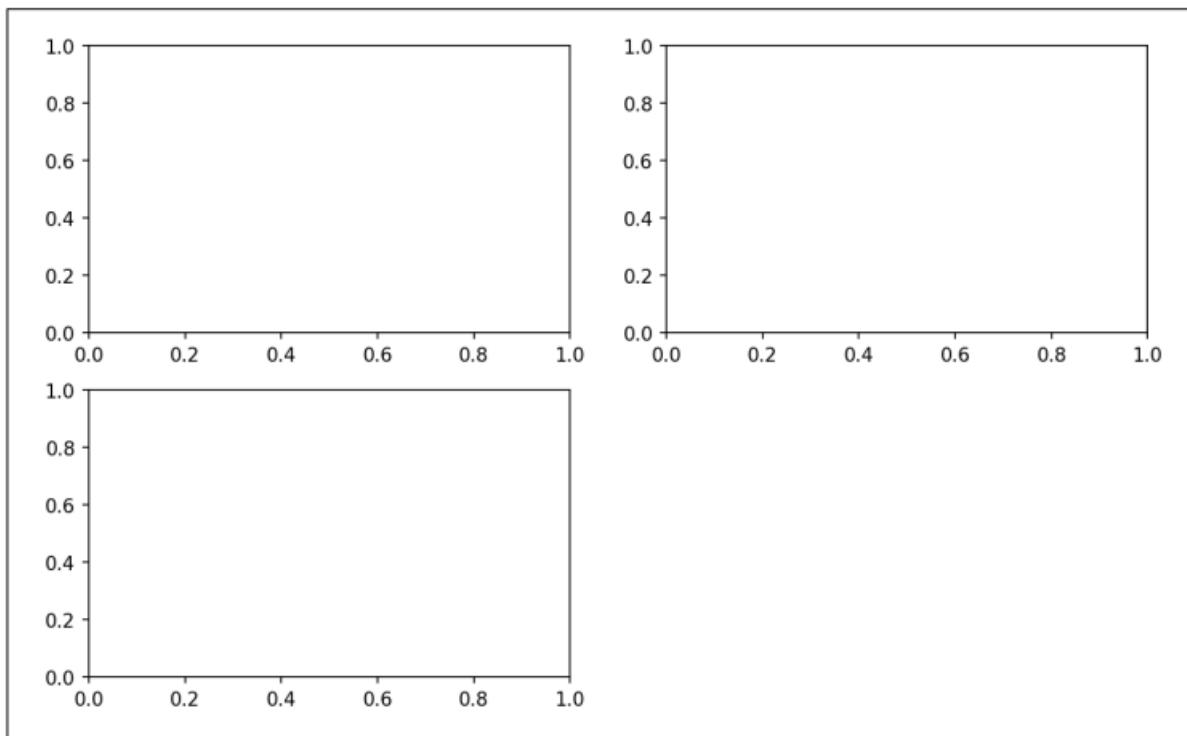
- Boş bir figürle çizim yapamazsınız. add\_subplot kullanarak bir veya daha fazla alt grafik oluşturmalısınız.

```
In [90]: ax1 = fig.add_subplot(2, 2, 1)
```

```
In [91]: ax2 = fig.add_subplot(2, 2, 2)
```

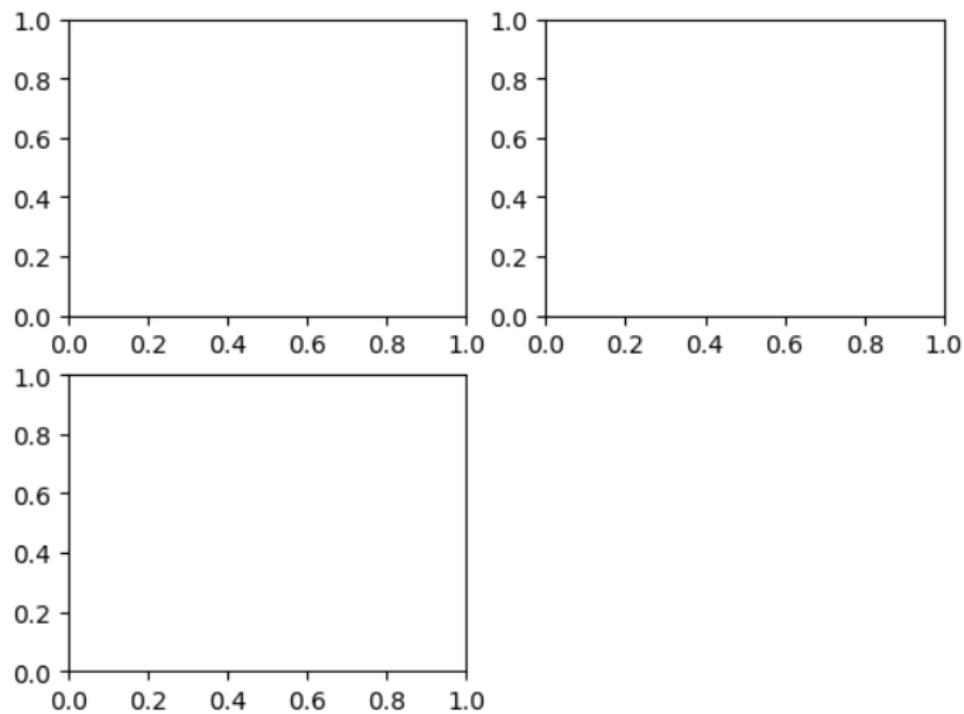
```
In [92]: ax3 = fig.add_subplot(2, 2, 3)
```

- Bu, şeklin  $2 \times 2$  olması gerektiği anlamına gelir (yani toplamda dört parsele kadar) ve dört alt grafikten ilkinin seçilmesi (1'den numaralandırılmış). Sonraki iki alt öğeyi oluşturursanız aşağıdaki şekillere benzeyen bir görselleştirme elde edeceksiniz.



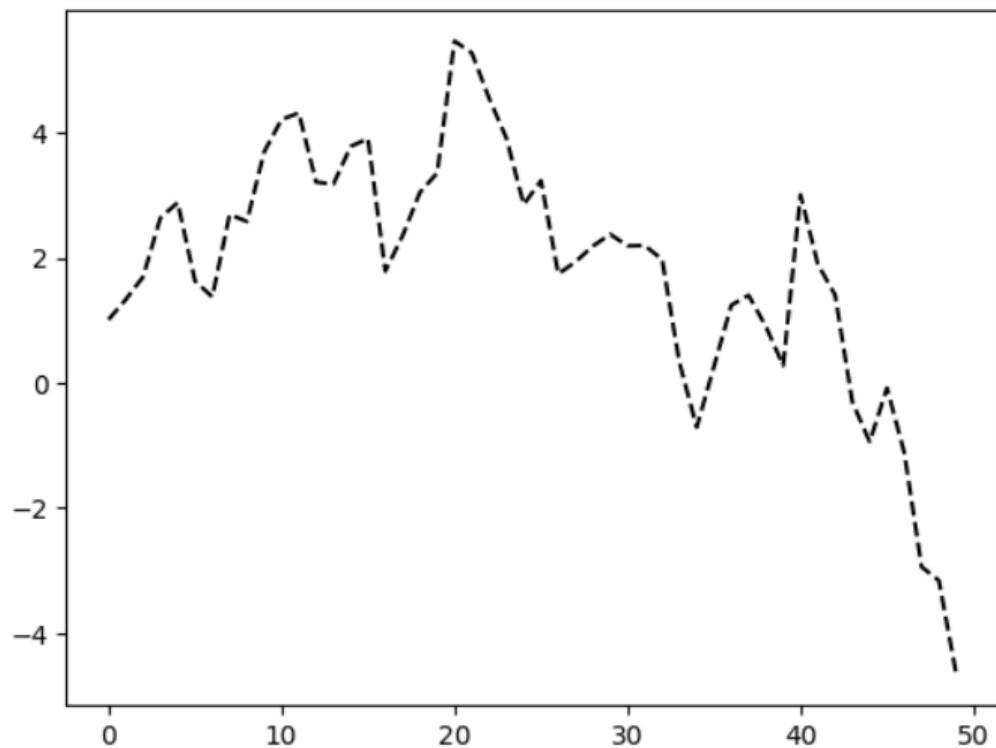
üç alt grafiği olan boş bir matplotlib şekli

```
In [97]: fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
```



```
In [98]: plt.plot(np.random.randn(50).cumsum(), 'k--')
```

```
Out[98]: [
```

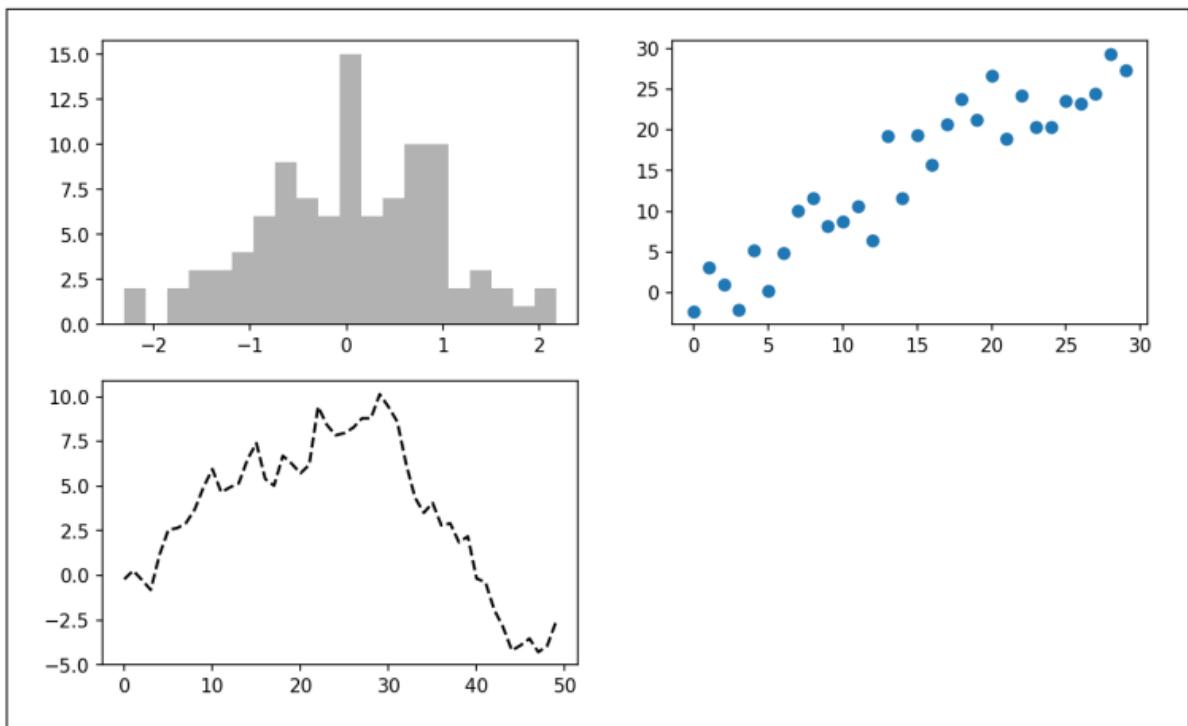


- plt.plot([1.5, 3.5, -2, 1.6]) gibi bir çizim komutu verdığınızda, matplotlib, kullanılan son şekil ve alt çizimden yararlanır (gerekirse bir tane oluşturur), böylece şekil ve alt çizim gizlenir. Son komutu eklersek, son şekil elde edilir.
- 'k--' matplotlib'e siyah kesikli çizgi çizmesi talimatını veren bir stil seçeneğidir. fig.add\_subplot tarafından döndürülen nesneler AxesSubplot nesneleridir; her birinin örnek yöntemini çağırarak diğer boş alt noktalar üzerinde doğrudan çizim yapılabilir.

```
In [99]: _ = ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)
```

```
In [100]: ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```

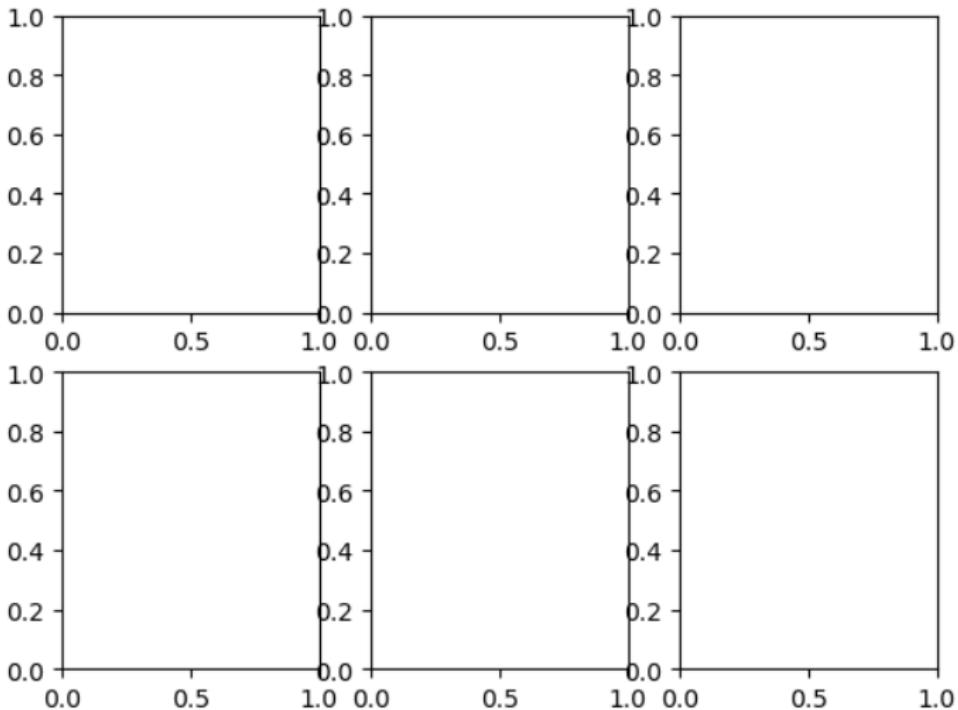
```
Out[100]: <matplotlib.collections.PathCollection at 0x16e12b03a90>
```



### Ek grafiklerden sonra veri görselleştirme

- Alt grafiklerden oluşan bir ızgaraya (grid) sahip bir şekil oluşturmak çok yaygın bir iştir, dolayısıyla matplotlib yeni bir şekil oluşturan ve geri dönen bir kolaylık yöntemi olan plt.subplots'u içerir.

```
In [103]: fig, axes = plt.subplots(2, 3)
```



```
In [104]: axes
```

```
Out[104]: array([[[<Axes: >, <Axes: >, <Axes: >],
                   [<Axes: >, <Axes: >, <Axes: >]], dtype=object)
```

- Eksenler dizisi iki boyutlu bir dizi gibi kolayca indekslenebildiğinden bu çok kullanışlıdır (örneğin eksenler[0, 1]). Ayrıca alt grafiklerin şu özelliklere sahip olması gerektiğini de belirtebilirsiniz: sırasıyla sharex ve sharey kullanılarak aynı x veya y eksenini.

## pyplot.subplots Seçenekleri

Argument	Description
nrows	Number of rows of subplots
ncols	Number of columns of subplots
sharex	All subplots should use the same x-axis ticks (adjusting the <code>xlim</code> will affect all subplots)
sharey	All subplots should use the same y-axis ticks (adjusting the <code>ylim</code> will affect all subplots)
subplot_kw	Dict of keywords passed to <code>add_subplot</code> call used to create each subplot
**fig_kw	Additional keywords to <code>subplots</code> are used when creating the figure, such as <code>plt.subplots(2, 2, figsize=(8, 6))</code>

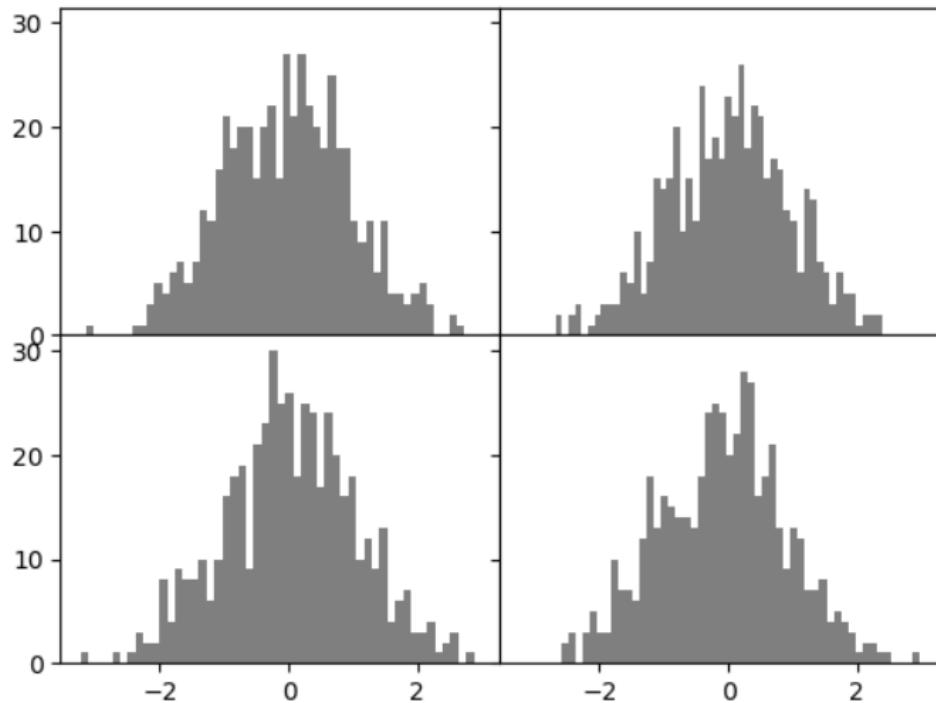
## Alt Noktalar Etrafındaki Aralığı Ayarlama

- Varsayılan olarak matplotlib, alt noktaların dış çevresinde ve alt noktalar arasında belirli bir miktarda boşluk bırakır. Bu aralığın tamamı grafiğin yüksekliğine ve genişliğine göre belirlenir, böylece grafiği GUI penceresini kullanarak programlı veya manuel olarak yeniden boyutlandırırsanız grafik kendisini dinamik olarak ayarlayacaktır. Şekil nesnelerinde üst düzey bir işlev olarak da kullanılabilen subplots\_adjust yöntemini kullanarak aralığı değiştirebilirsiniz:

```
subplots_adjust(left=None, bottom=None, right=None, top=None,
                 wspace=None, hspace=None)
```

- wspace ve hspace, alt noktalar arasında boşluk olarak kullanılacak sırasıyla şekil genişliğinin ve şekil yüksekliğinin yüzdesini kontrol eder.

```
In [106]: fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
for i in range(2):
    for j in range(2):
        axes[i, j].hist(np.random.randn(500), bins=50, color='k', alpha=0.5)
plt.subplots_adjust(wspace=0, hspace=0)
```



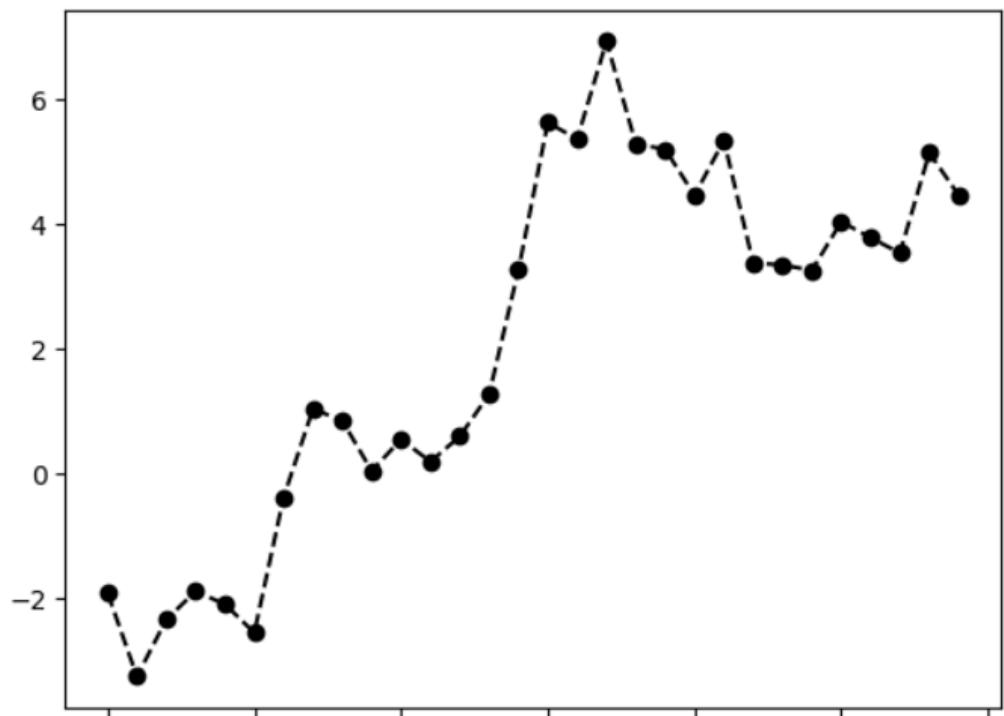
alt noktalar arası boşluk olmadan veri görselleştirme

- Eksen etiketlerinin üst üste geldiğini fark edebilirsiniz. matplotlib, etiketlerin çakışıp örtüşmediğini kontrol etmez; dolayısıyla böyle bir durumda, açık onay konumlarını ve onay etiketlerini belirterek etiketleri kendiniz düzeltmeniz gereklidir.
- Çizgi grafikleri ayrıca gerçek veri noktalarını vurgulamak için işaretleyicilere sahip olabilir. matplotlib, noktalar arasında interpolasyon yapan sürekli bir çizgi grafiği oluşturduğundan, bazen noktaların nerede olduğu belirsiz olabilir. İşaretleyici, rengin ardından işaretleyici tipi ve çizgi stilinin gelmesi gereken stil dizisinin bir parçası olabilir.

```
In [107]: from numpy.random import randn
```

```
In [108]: plt.plot(randn(30).cumsum(), 'ko--')
```

```
Out[108]: <matplotlib.lines.Line2D at 0x16e13275a50>
```



İşaretleyicilerle çizgi grafiği

- Bu aynı zamanda daha açık bir şekilde şu şekilde de yazılabildi:

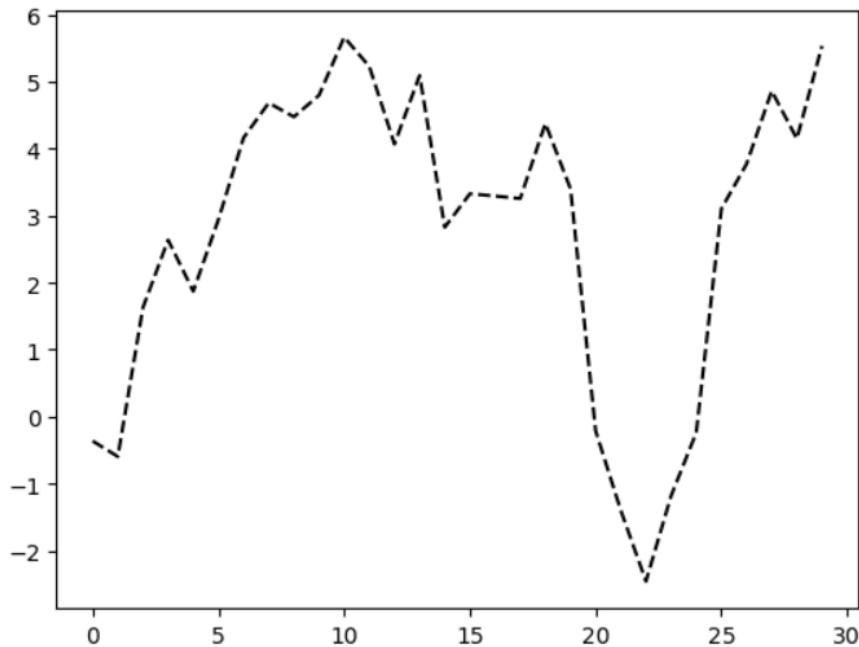
```
In [110]: plt.plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
```

- Çizgi grafikleri için sonraki noktaların varsayılan olarak doğrusal olarak interpolasyona tabi tutulduğunu fark edeceksiniz. Bu, çizim stili seçeneğiyle değiştirilebilir.

```
In [111]: data = np.random.randn(30).cumsum()
```

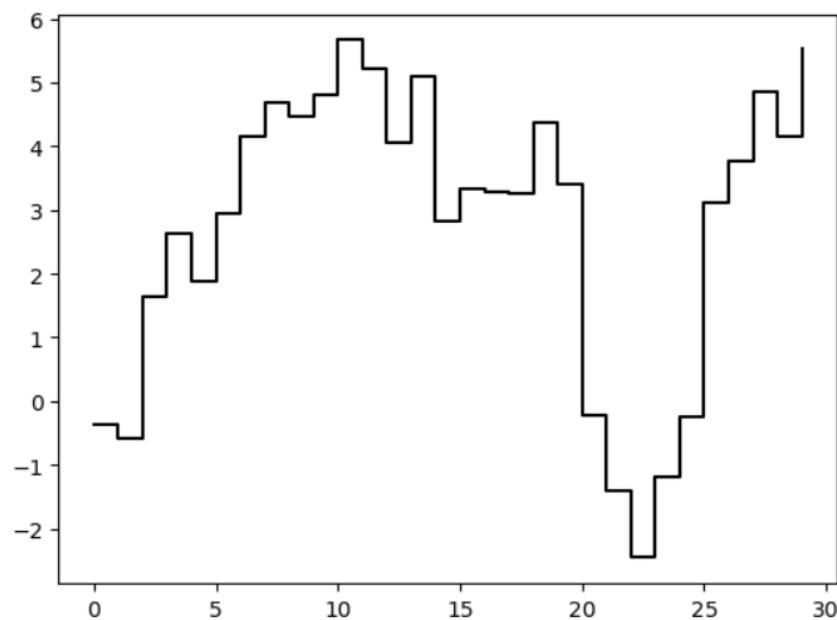
```
In [112]: plt.plot(data, 'k--', label='Default')
```

```
Out[112]: [<matplotlib.lines.Line2D at 0x16e132f5450>]
```



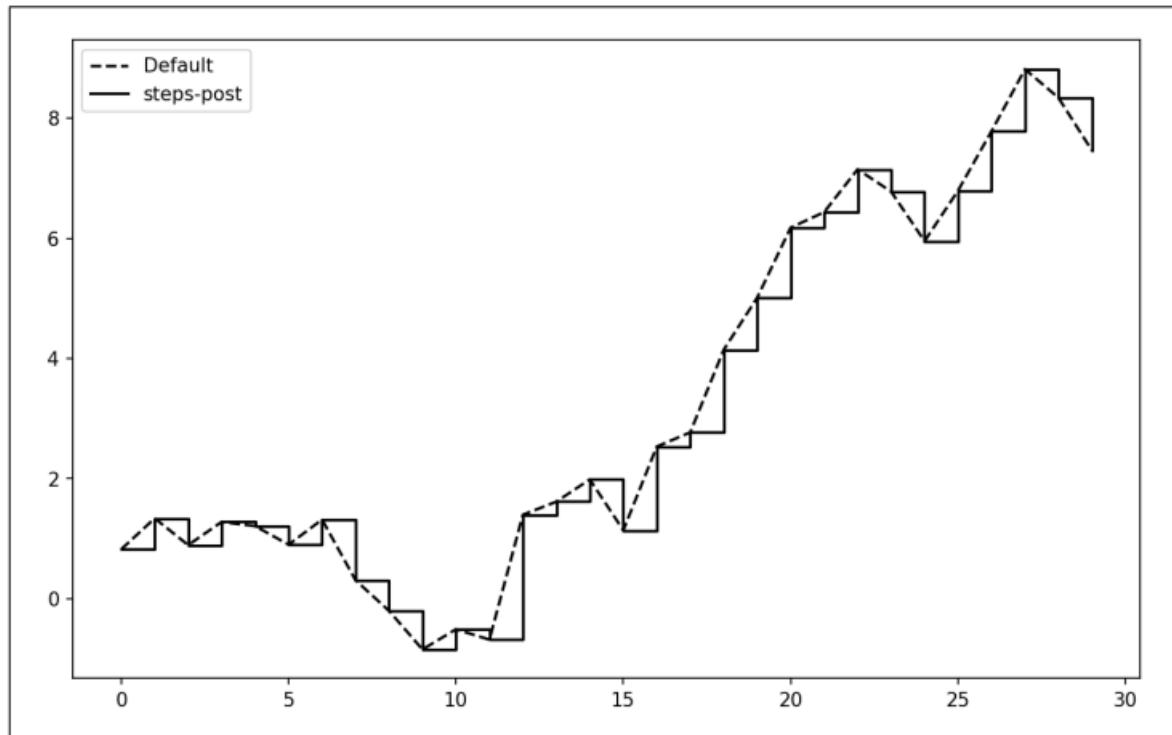
```
In [113]: plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')
```

```
Out[113]: [<matplotlib.lines.Line2D at 0x16e131fd890>]
```



```
In [116]: plt.legend(loc='best')
```

```
Out[116]: <matplotlib.legend.Legend at 0x16e13590390>
```



farklı çizim stili seçenekleriyle çizgi grafiği

- Bunu çalıştırığınızda `<matplotlib.lines.Line2D at ...>` gibi bir çıktı görebilirsiniz. matplotlib, yeni eklenen çizim alt bileşenine başvuran nesneleri döndürür. Burada, etiket argümanlarını çizime ilettigimiz için, `plt.legend()` kullanarak her satırı tanımlamak için bir çizim efsanesi oluşturuyoruz.

## İşaretler (Ticks), Etiketler (Labels) ve Göstergeler (Legends)

- Çoğu çizim dekorasyonu türü için, işleri yapmanın iki ana yolu vardır: prosedürel pyplot arayüzü (örn., `matplotlib.pyplot`) ve daha nesne yönelimli yerel matplotlib API'sini kullanmak.
- İnteraktif kullanım için tasarlanan pyplot arayüzü `xlim`, `xticks`, `xticlabels` gibi yöntemlerden oluşmaktadır. Bunlar çizim aralığını, işaret konumlarını ve işaret etiketlerini sırasıyla kontrol eder. İki şekilde kullanılabilirler:
  - Bağımsız değişken olmadan çağrıldığında geçerli parametre değeri döndürür (örn. `plt.xlim()`, geçerli x ekseni çizim aralığını döndürür)
  - Parametrelerle çağrıldığında parametre değeri ayarlanır (örn. `plt.xlim([0, 10])`, x ekseni aralığını 0 ile 10 arasında ayarlar)

- Bu tür yöntemlerin tümü, aktif veya en son oluşturulan AxesSubplot'a etki eder. Bunların her biri alt çizim nesnesinin kendisindeki iki yönteme karşılık gelir; xlim durumunda bunlar ax.get\_xlim ve ax.set\_xlim'dir.

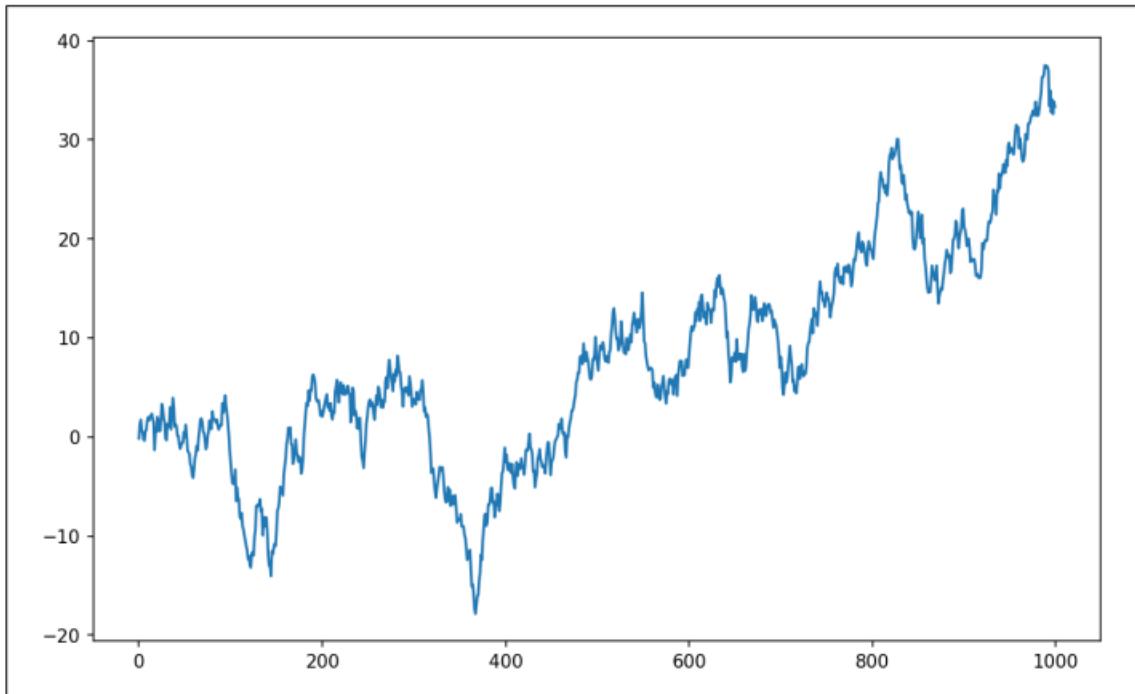
## Başlığı, Eksen Etiketlerini, Onay İşaretlerini ve Onay Etiketlerini Ayarlama

```
In [5]: fig = plt.figure()
<Figure size 640x480 with 0 Axes>

In [6]: ax = fig.add_subplot(1, 1, 1)

In [7]: ax.plot(np.random.randn(1000).cumsum())
Out[7]: [

```



xtick'leri göstermek için basit çizim

- X ekseni işaretlerini değiştirmek için set\_xticks ve set\_xticklabels'i kullanmak en kolay yoldur. İlk, matplotlib'e veri aralığı boyunca işaretlerin nereye yerleştirileceğini bildirir; varsayılan olarak bu konumlar aynı zamanda etiketler olacaktır. Ancak set\_xticklabels kullanarak diğer değerleri etiket olarak ayarlayabiliriz:

```
In [8]: ticks = ax.set_xticks([0, 250, 500, 750, 1000])
```

```
In [9]: labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'],
                                rotation=30, fontsize='small')
```

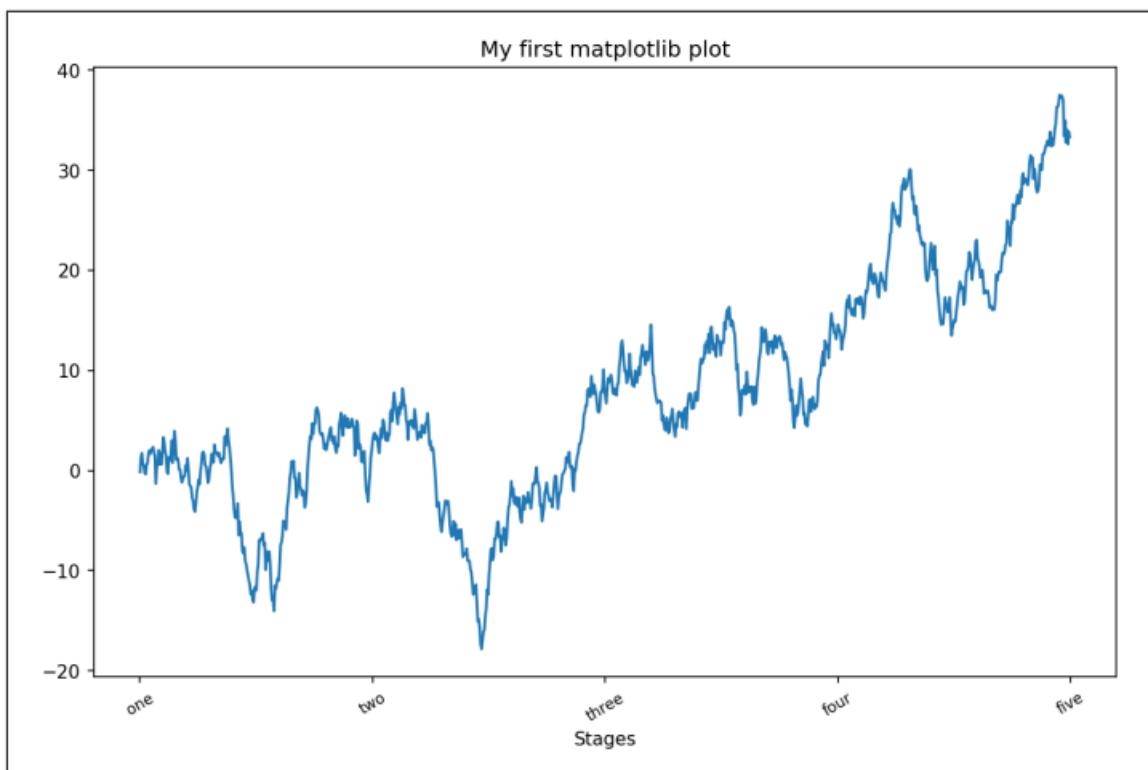
- Döndürme seçeneği, x işaret etiketlerini 30 derecelik bir dönüşe ayarlar. Son olarak set\_xlabel, x eksenine bir isim verir ve set\_title alt çizim başlığını verir.

```
In [10]: ax.set_title('My first matplotlib plot')
```

```
Out[10]: Text(0.5, 1.0, 'My first matplotlib plot')
```

```
In [11]: ax.set_xlabel('Stages')
```

```
Out[11]: Text(0.5, 0, 'Stages')
```



xticks'i göstermek için basit çizim

- Y eksenini değiştirmek, yukarıda x'in yerine y'yi koymakla aynı işlemden oluşur. Axes sınıfı, çizim özelliklerinin toplu olarak ayarlanması için izin veren bir set yöntemine sahiptir.

Önceki örneği şu şekilde de yapabilirdik:

```
In [12]: props = {
    'title': 'My first matplotlib plot',
    'xlabel': 'Stages'
}
ax.set(**props)
```

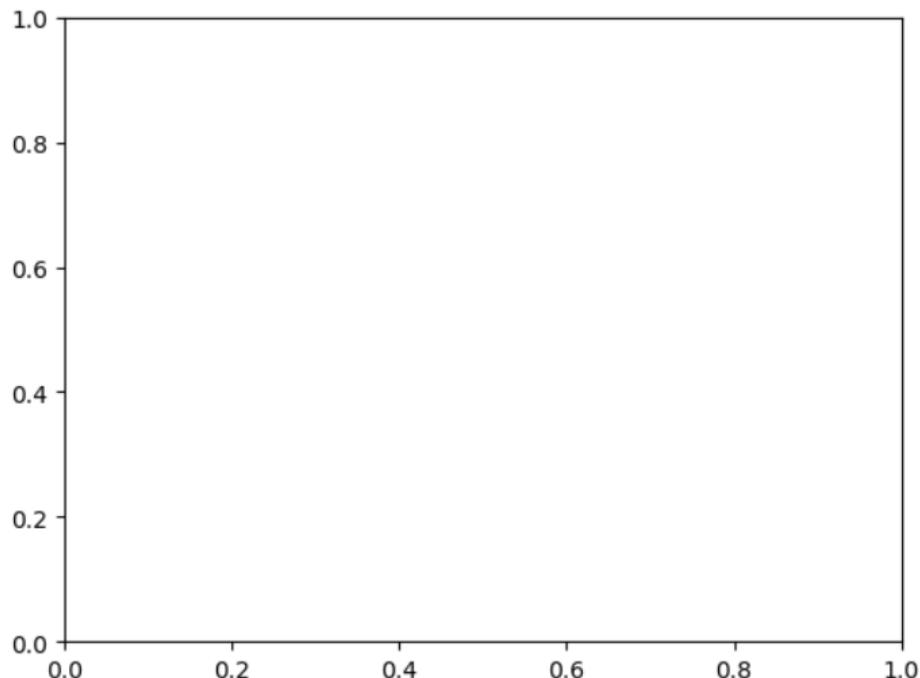
```
Out[12]: [Text(0.5, 1.0, 'My first matplotlib plot'), Text(0.5, 0, 'stages')]
```

## Legendlar Ekleme

- Legend olay örgüsünün öğelerini tanımlamak için başka bir kritik ögedir. Bir tane eklemenin birkaç yolu vardır. En kolayı, grafiğin her bir parçasını eklerken label argümanını iletmektir.

```
In [13]: from numpy.random import randn
```

```
In [14]: fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
```



```
In [15]: ax.plot(randn(1000).cumsum(), 'k', label='one')
```

```
Out[15]: [
```

```
In [16]: ax.plot(randn(1000).cumsum(), 'k--', label='two')
```

```
Out[16]: [
```

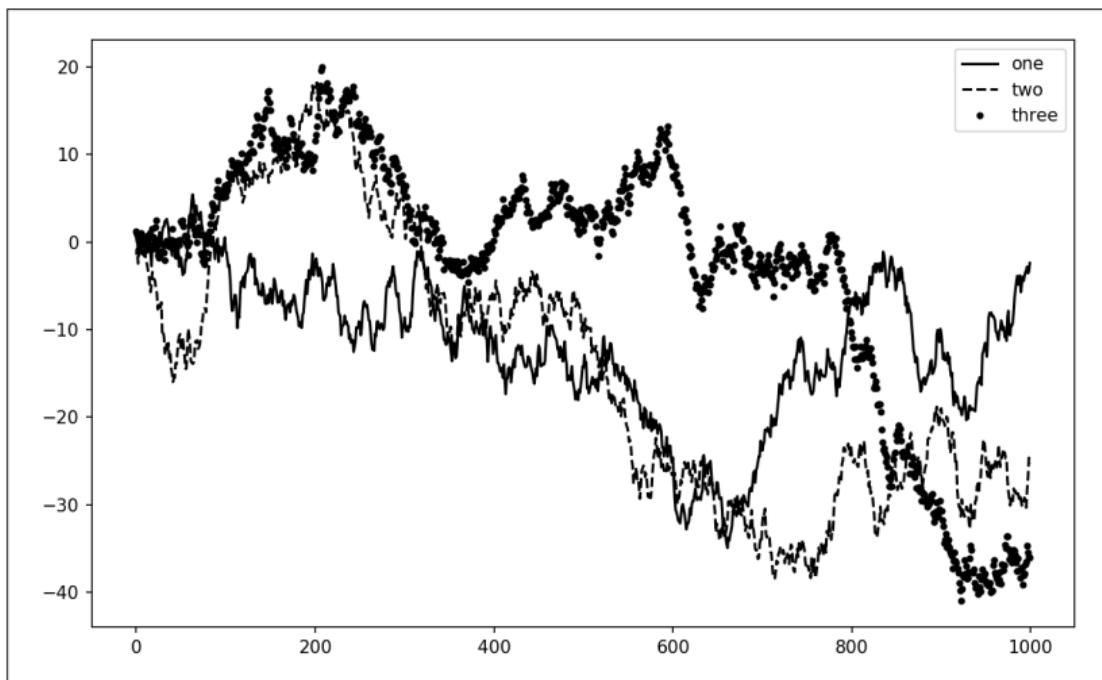
```
In [17]: ax.plot(randn(1000).cumsum(), 'k.', label='three')
```

```
Out[17]: [
```

- Bunları yaptıktan sonra, otomatik olarak bir gösterge oluşturmak için `ax.legend()` veya `plt.legend()` ögesini çağrılabilirsiniz. Ortaya çıkan grafik:

```
In [25]: ax.legend(loc='best')
```

```
Out[25]: <matplotlib.legend.Legend at 0x23d15eb89d0>
```



## Bir Alt Grafik (Subplot) Üzerindeki Açıklamalar ve Çizim

- Standart çizim türlerine ek olarak, metin, oklar veya diğer şekillerden oluşan kendi çizim açıklamalarınızı da çizmek isteyebilirsiniz. Metin, ok ve açıklama ekleme işlevlerini kullanarak açıklamalar ve metin ekleyebilirsiniz. `text`, isteğe bağlı özel stil ile metni grafik üzerinde verilen koordinatlarda (x, y) çizer.

```
In [32]: ax.text(x, y, 'Hello world!',  
                 family='monospace', fontsize=10)
```

```
In [33]: from datetime import datetime

In [34]: fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

In [ ]: data = pd.read_csv('examples/spx.csv', index_col=0, parse_dates=True)
spx = data['SPX']

In [ ]: spx.plot(ax=ax, style='k-')

In [ ]: crisis_data = [
    (datetime(2007, 10, 11), 'Peak of bull market'),
    (datetime(2008, 3, 12), 'Bear Stearns Fails'),
    (datetime(2008, 9, 15), 'Lehman Bankruptcy')
]

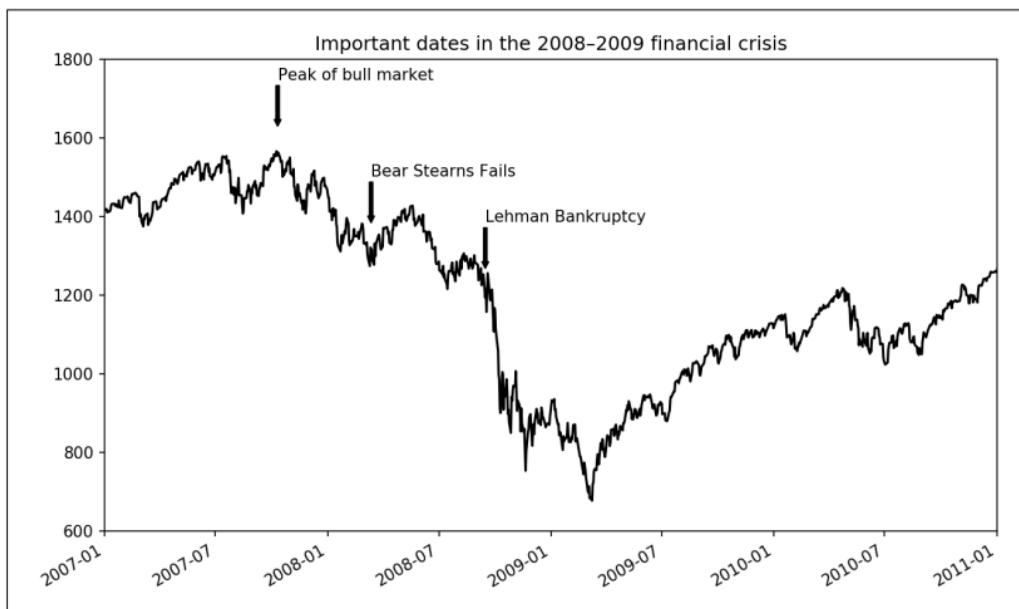
In [ ]: for date, label in crisis_data:
    ax.annotate(label, xy=(date, spx.asof(date) + 75),
                xytext=(date, spx.asof(date) + 225),
                arrowprops=dict(facecolor='black', headwidth=4, width=2,
                                headlength=4),
                horizontalalignment='left', verticalalignment='top')
```

```
In [42]: ax.set_xlim(['1/1/2007', '1/1/2011'])
ax.set_ylim([600, 1800])

Out[42]: (600.0, 1800.0)
```

```
In [43]: ax.set_title('Important dates in the 2008-2009 financial crisis')

Out[43]: Text(0.5, 1.0, 'Important dates in the 2008-2009 financial crisis')
```



2008–2009 mali krizindeki önemli tarihler

- Bu çizimde vurgulanması gereken birkaç önemli nokta vardır: `ax.annotate` yöntemi, belirtilen x ve y koordinatlarında etiketler çizebilir. Grafiğin başlangıç ve bitiş sınırlarını manuel olarak ayarlamak için `matplotlib`'in varsayılanını kullanmak yerine `set_xlim` ve `set_ylim` yöntemlerini kullanırız. Son olarak `ax.set_title` olay örgüsüne bir ana başlık ekler.
- Şekil çizmek biraz daha özen gerektirir. `matplotlib`, yamalar olarak adlandırılan birçok yaygın şekli temsil eden nesnelere sahiptir. Bunlardan Dikdörtgen ve Daire gibi bazıları `matplotlib.pyplot`'ta bulunur, ancak tam `matplotlib.patches`'te bulunur.
- Grafiğe şekil eklemek için `shp patch` nesnesini oluşturursunuz ve onu `ax.add_patch(shp)` çağrarak bir alt çizime eklersiniz.

```
In [46]: fig = plt.figure()
```

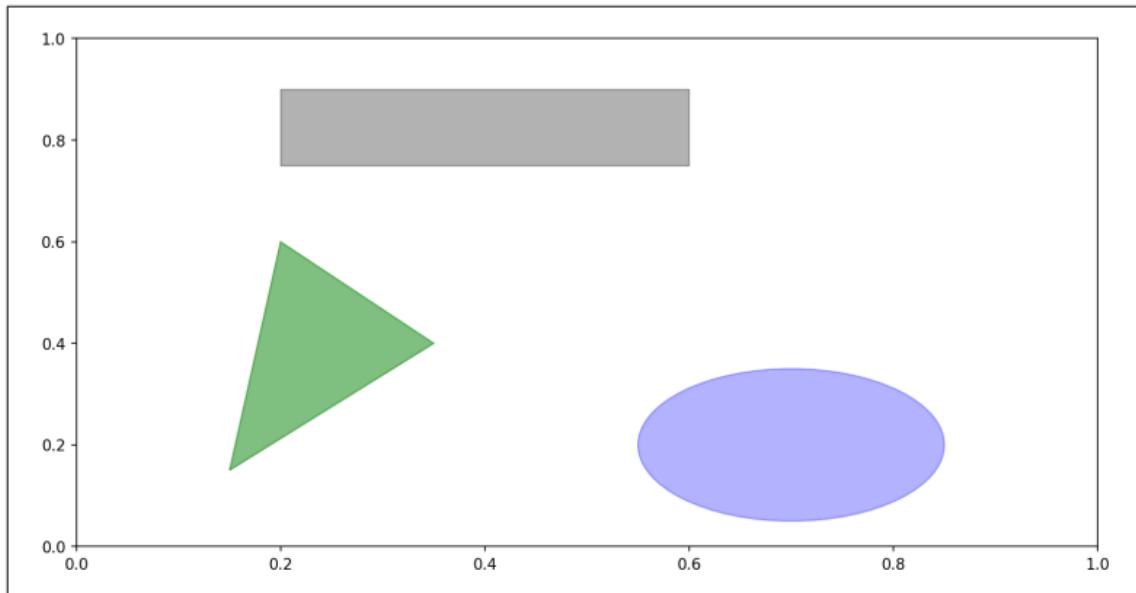
```
<Figure size 640x480 with 0 Axes>
```

```
In [47]: ax = fig.add_subplot(1, 1, 1)
```

```
In [48]: rect = plt.Rectangle((0.2, 0.75), 0.4, 0.15, color='k', alpha=0.3)
circ = plt.Circle((0.7, 0.2), 0.15, color='b', alpha=0.3)
pgon = plt.Polygon([[0.15, 0.15], [0.35, 0.4], [0.2, 0.6]],
                   color='g', alpha=0.5)
```

```
In [49]: ax.add_patch(rect)
ax.add_patch(circ)
ax.add_patch(pgon)
```

```
Out[49]: <matplotlib.patches.Polygon at 0x23d1a289650>
```



üç farklı patch'den oluşan veri görselleştirmesi

## Grafikleri Dosyaya Kaydetme

- Aktif şekli plt.savefig kullanarak dosyaya kaydedebilirsiniz. Bu yöntem, şekil nesnesinin savefig örneği yöntemine eşdeğerdir. Örneğin, bir şeklin SVG sürümünü kaydetmek için yalnızca şunu yazmanız gereklidir:

```
In [50]: plt.savefig('figpath.svg')
<Figure size 640x480 with 0 Axes>
```

- Dosya türü dosya uzantısından anlaşılmaktadır. Yani bunun yerine .pdf kullanısaydınız bir PDF elde edersiniz.
- Grafik yayılmak için sıkılıkla kullanılan birkaç önemli seçenek var: inch başına nokta çözünürlüğünü kontrol eden dpi ve bbox\_inches, gerçek şeklin etrafındaki boşlukları kırpabilir. PNG ile aynı çizimi, çizim çevresinde minimum boşlukla ve 400 DPI'da elde etmek için şunları yaparsınız:

```
In [51]: plt.savefig('figpath.png', dpi=400, bbox_inches='tight')
<Figure size 640x480 with 0 Axes>
```

- savefig'in diske yazmasına gerek yoktur; ayrıca BytesIO gibi herhangi bir dosya benzeri nesneye de yazabilir:

```
In [52]: from io import BytesIO
buffer = BytesIO()
plt.savefig(buffer)
plot_data = buffer.getvalue()

<Figure size 640x480 with 0 Axes>
```

## Figure.savefig Seçenekleri

Argument	Description
fname	String containing a filepath or a Python file-like object. The figure format is inferred from the file extension (e.g., .pdf for PDF or .png for PNG)
dpi	The figure resolution in dots per inch; defaults to 100 out of the box but can be configured
facecolor , edgecolor	The color of the figure background outside of the subplots; 'w' (white), by default
format	The explicit file format to use ('png', 'pdf', 'svg', 'ps', 'eps', ...)
bbox_inches	The portion of the figure to save; if 'tight' is passed, will attempt to trim the empty space around the figure

## matplotlib Konfigürasyonu

- matplotlib, öncelikle rakamları yayına hazırlamaya yönelik renk şemaları ve varsayılanlarla yapılandırılmış olarak gelir. Neyse ki, varsayılan davranışın neredeyse tamamı, şekil boyutunu, alt çizim aralığını, renkleri, yazı tipi boyutlarını, izgara stillerini vb. yöneten kapsamlı bir küresel parametreler kümesi aracılığıyla özelleştirilebilir. Yapılandırmayı Python'dan programlı olarak değiştirmenin bir yolu rc yöntemini kullanmaktadır; örneğin, genel varsayılan şekil boyutunu  $10 \times 10$  olarak ayarladığınızda şunu girebilirsiniz:

```
In [53]: plt.rc('figure', figsize=(10, 10))
```

- rc'nin ilk argümanı, özelleştirmek istediğiniz bileşendir; örneğin 'figure', 'axes', 'xtick', 'ytick', 'grid', 'legend' veya diğerleri. Bundan sonra yeni parametreleri gösteren bir dizi anahtar kelime argümanı takip edilebilir. Programınızdaki seçenekleri yazmanın kolay bir yolu bir dict'dir:

```
In [54]: font_options = {'family': 'monospace',
                     'weight': 'bold',
                     'size': 'small'}
plt.rc('font', **font_options)
```

## Pandas ve Seaborn ile Grafik Çizmek

- matplotlib oldukça düşük seviyeli bir araç olabilir. Bir grafiği temel bileşenlerinden bir araya getirirsiniz: veri ekranı (yani çizimin türü: çizgi, çubuk, kutu, dağılım, kontur vb.), açıklama, başlık, onay etiketleri ve diğer ek açıklamalar.
- Pandalarda satır ve sütun etiketlerinin yanı sıra birden fazla veri sütunumuz olabilir. pandas'ın kendisi Verilerden görselleştirmeler oluşturmayı kolaylaştıran yerleşik yöntemlere sahiptir. DataFrame ve Series nesneleri. Başka bir kütüphane ise Michael Waskom tarafından oluşturulan istatistiksel bir grafik kütüphanesi olan Seaborn'dur. Seaborn birçok yaygın görselleştirmeyi oluşturmayı basitleştirir.

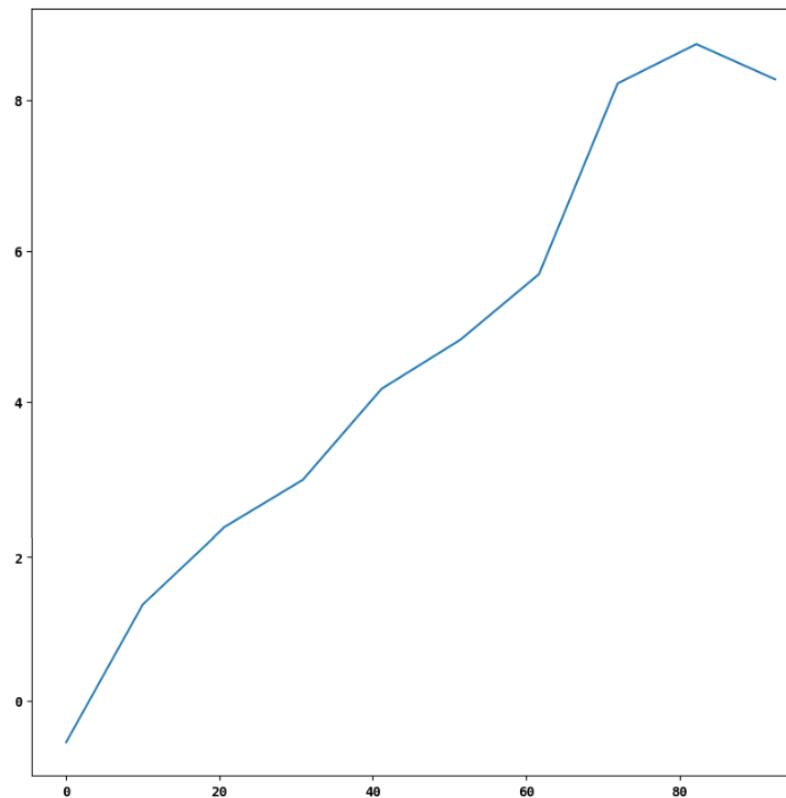
## Çizgi Grafikleri

- Series ve DataFrame'in her biri, bazı temel çizim türlerini oluşturmak için bir çizim özelliğine sahiptir. Varsayılan olarak, plot() fonksiyonu çizgi çizimleri yapar.

```
In [55]: s = pd.Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10))
```

```
In [56]: s.plot()
```

```
Out[56]: <Axes: >
```

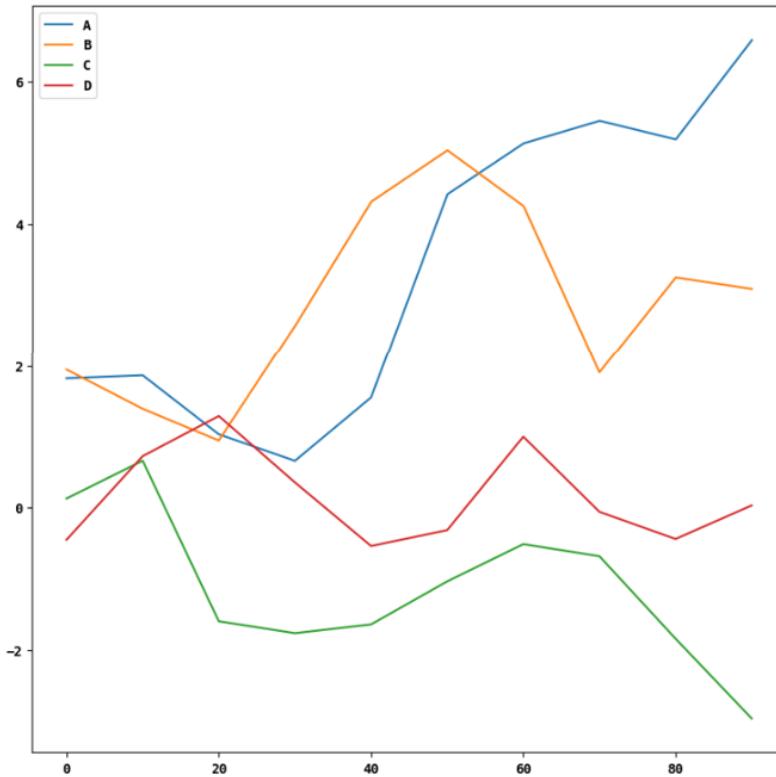


- Series nesnesinin dizini, x ekseninde çizim yapmak için matplotlib'e iletilir, ancak bunu use\_index=False ileterek devre dışı bırakabilirsiniz. X eksenini işaretleri ve sınırları xticks ve xlim seçenekleriyle, yaxis ise sırasıyla yticks ve ylim ile ayarlanabilir.
- DataFrame'in plot yöntemi, sütunlarının her birini aynı alt grafikte farklı bir çizgi olarak çizerek otomatik olarak bir açıklama oluşturur

```
In [60]: df = pd.DataFrame(np.random.randn(10, 4).cumsum(0),
                        columns=['A', 'B', 'C', 'D'],
                        index=np.arange(0, 100, 10))
```

```
In [61]: df.plot()
```

```
Out[61]: <Axes: >
```



- Grafik özelliği, farklı çizim türleri için bir yöntem ailesi içerir. Örneğin, `df.plot()`, `df.plot.line()`'a eşdeğerdir.

## Series.plot Yöntemi Bağımsız Değişkenleri

Argument	Description
<code>label</code>	Label for plot legend
<code>ax</code>	matplotlib subplot object to plot on; if nothing passed, uses active matplotlib subplot
<code>style</code>	Style string, like ' <code>ko--</code> ', to be passed to matplotlib
<code>alpha</code>	The plot fill opacity (from 0 to 1)
<code>kind</code>	Can be ' <code>area</code> ', ' <code>bar</code> ', ' <code>barh</code> ', ' <code>density</code> ', ' <code>hist</code> ', ' <code>kde</code> ', ' <code>line</code> ', ' <code>pie</code> '
<code>logy</code>	Use logarithmic scaling on the y-axis
<code>use_index</code>	Use the object index for tick labels
<code>rot</code>	Rotation of tick labels (0 through 360)
<code>xticks</code>	Values to use for x-axis ticks
<code>yticks</code>	Values to use for y-axis ticks
<code>xlim</code>	x-axis limits (e.g., <code>[0, 10]</code> )
<code>ylim</code>	y-axis limits
<code>grid</code>	Display axis grid (on by default)

- DataFrame'in, sütunların nasıl işleneceği konusunda bir miktar esneklik sağlayan bir dizi seçeneği vardır; örneğin, bunların hepsini aynı alt grafik üzerinde mi çizeceğiz yoksa ayrı alt noktalar mı oluşturacağız.

## DataFrame'e Özgü Çizim Bağımsız Değişkenleri

Argument	Description
subplots	Plot each DataFrame column in a separate subplot
sharex	If subplots=True, share the same x-axis, linking ticks and limits
sharey	If subplots=True, share the same y-axis
figsize	Size of figure to create as tuple
title	Plot title as string
legend	Add a subplot legend (True by default)
sort_columns	Plot columns in alphabetical order; by default uses existing column order

## Çubuk Grafikleri

- plot.bar() ve plot.barh() sırasıyla dikey ve yatay çubuk grafikleri oluşturur. Bu durumda Series veya DataFrame dizini x (bar) veya y (barh) işaretleri olarak kullanılacaktır.

```
In [62]: fig, axes = plt.subplots(2, 1)
```

```
In [67]: data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnp'))
```

```
In [68]: data.plot.bar(ax=axes[0], color='k', alpha=0.7)
```

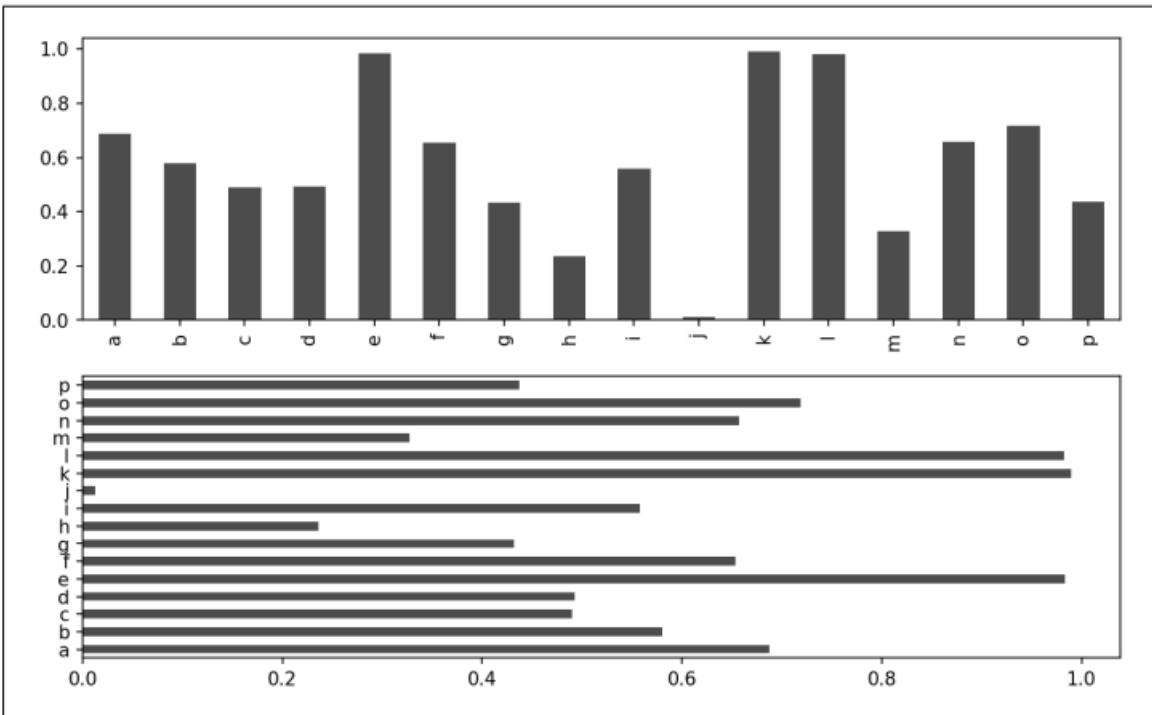
```
Out[68]: <Axes: >
```

```
In [69]: data.plot.barh(ax=axes[1], color='k', alpha=0.7)
```

```
Out[69]: <Axes: >
```

```
In [70]: data.plot()
```

```
Out[70]: <Axes: >
```



- color='k' ve alpha=0,7 seçenekleri grafiklerin rengini siyah olarak ayarlar ve dolguda kısmi şeffaflık kullanır.

```
In [71]: df = pd.DataFrame(np.random.rand(6, 4),
                      index=['one', 'two', 'three', 'four', 'five', 'six'],
                      columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))
```

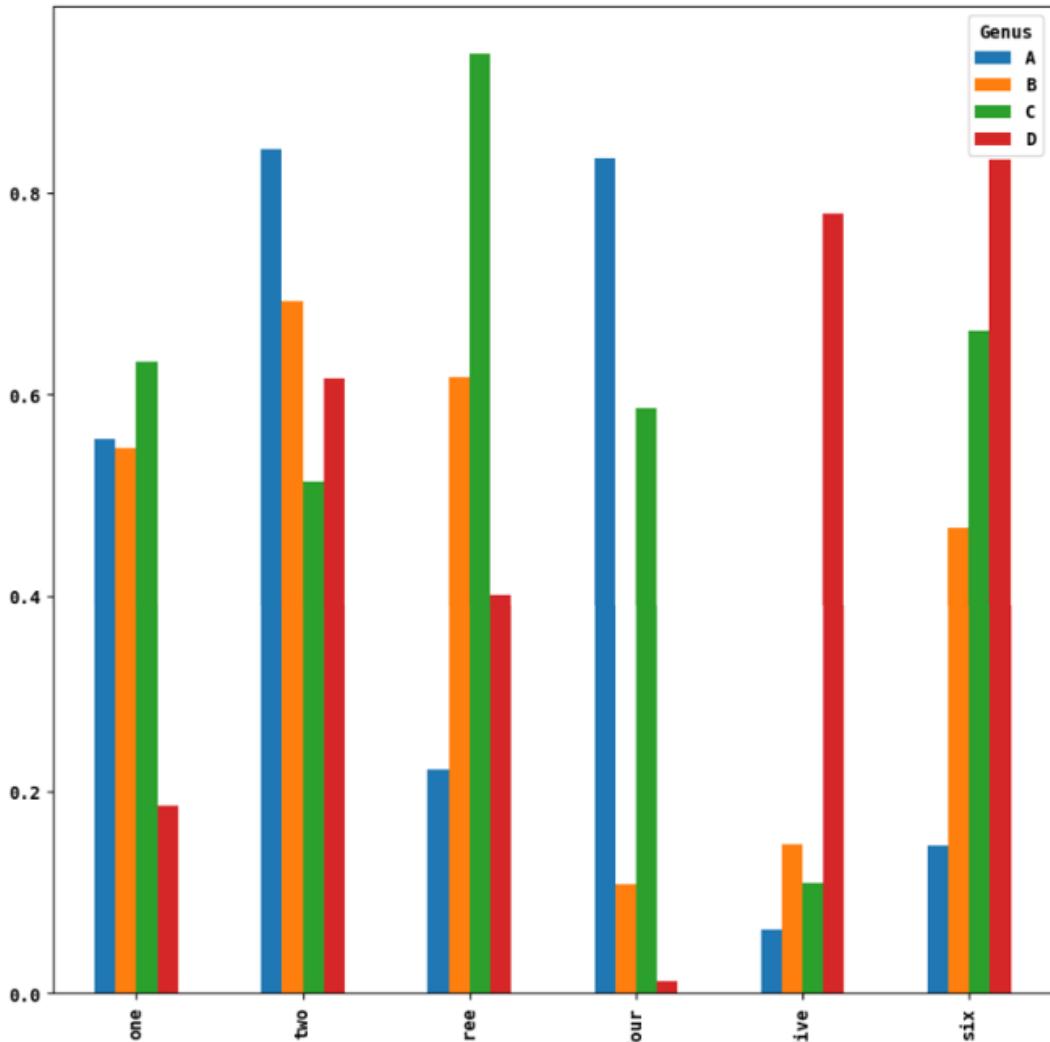
```
In [72]: df
```

```
Out[72]:
```

Genus	A	B	C	D
one	0.556126	0.547129	0.632309	0.185722
two	0.843234	0.693345	0.514242	0.616070
three	0.222189	0.617800	0.938330	0.401604
four	0.834258	0.107578	0.587156	0.012285
five	0.063243	0.147170	0.109188	0.779356
six	0.145964	0.467484	0.663601	0.834115

```
In [73]: df.plot.bar()
```

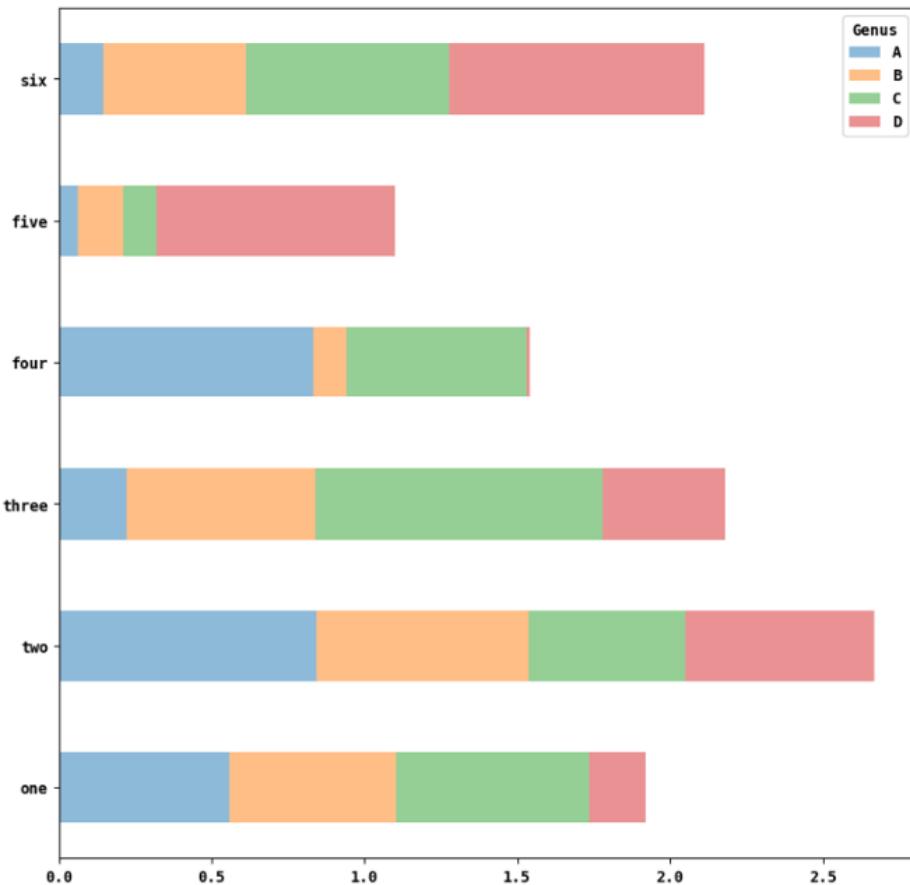
```
Out[73]: <Axes: >
```



- DataFrame'in sütunlarındaki "Genus" adının legendlara başlık vermek için kullanıldığını unutmayın.
- Stacked=True'u ileterek bir DataFrame'den yiğilmiş çubuk grafikleri oluştururuz, bu da her satırındaki değerin birlikte yiğilmasına neden olur.
- Çubuk grafikleri için kullanışlı bir yol, bir Serinin değer sıklığını görselleştirmektir value\_counts kullanımı: s.value\_counts().plot.bar().

```
In [74]: df.plot.barh(stacked=True, alpha=0.5)
```

```
Out[74]: <Axes: >
```



```
In [128]: tips = pd.read_csv('tips (2).csv')
```

```
In [129]: party_counts = pd.crosstab(tips['day'], tips['size'])
```

```
In [130]: party_counts
```

```
Out[130]:
size  1   2   3   4   5   6
day
Fri   1   16   1   1   0   0
Sat   2   53   18  13   1   0
Sun   0   39   15  18   3   1
Thur  1   48   4   5   1   3
```

- Daha sonra, her satırın toplamı 1 olacak şekilde normalleştirin ve çizimi yapın.

```
In [131]: party_counts = party_counts.loc[:, 2:5]
```

```
In [132]: party_pcts = party_counts.div(party_counts.sum(1), axis=0)
```

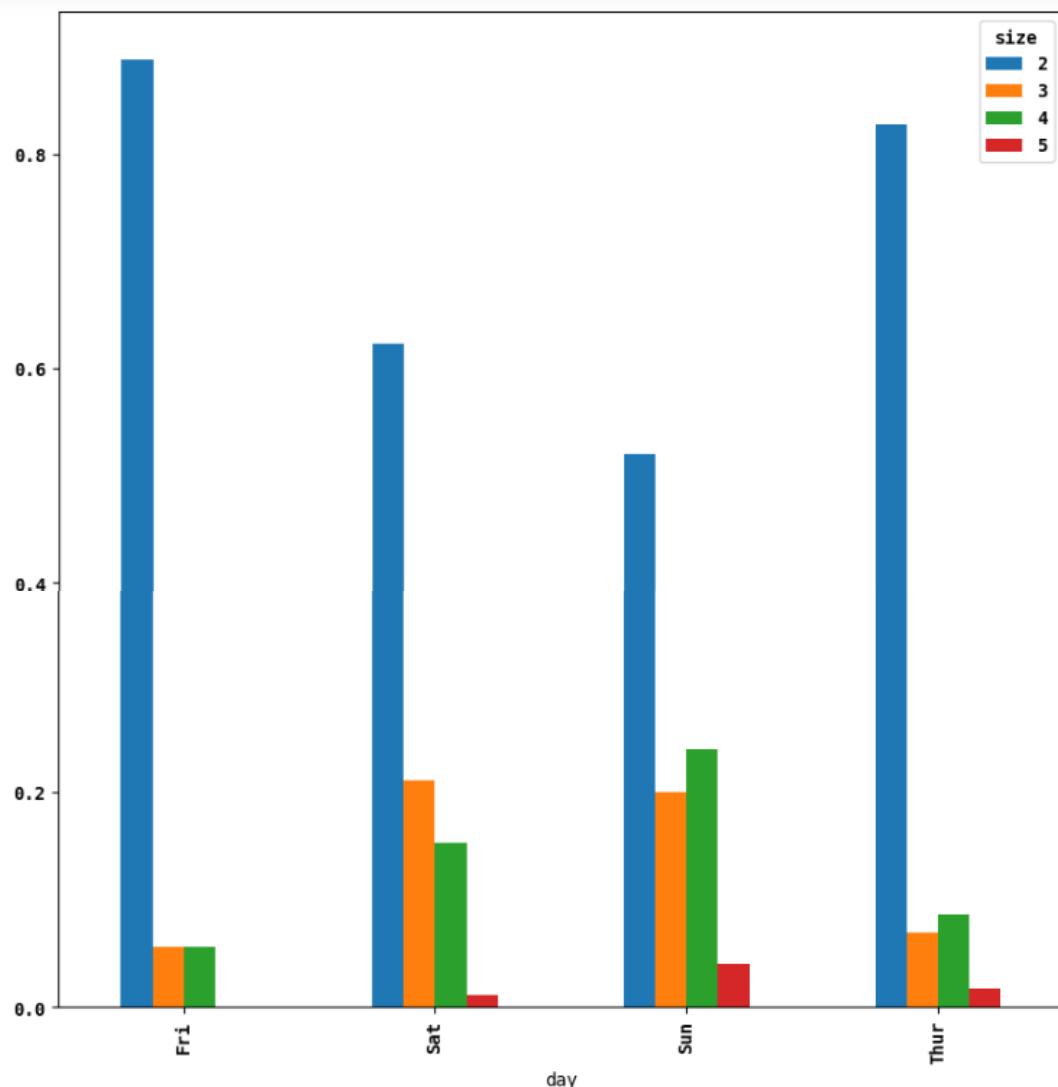
```
In [133]: party_pcts
```

```
Out[133]:
```

size	2	3	4	5
day				
Fri	0.888889	0.055556	0.055556	0.000000
Sat	0.623529	0.211765	0.152941	0.011765
Sun	0.520000	0.200000	0.240000	0.040000
Thur	0.827586	0.068966	0.086207	0.017241

```
In [134]: party_pcts.plot.bar()
```

```
Out[134]: <Axes: xlabel='day'>
```



- Yani bu veri setinde parti boyutlarının hafta sonları arttığını görebilirsiniz. Grafik oluşturmadan önce toplama veya özetleme gerektiren verilerde, seaborn paketini kullanmak işleri çok daha basit hale getirebilir.

```
In [135]: import seaborn as sns
```

```
In [136]: tips['tip_pct'] = tips['tip'] / (tips['total_bill'] - tips['tip'])
```

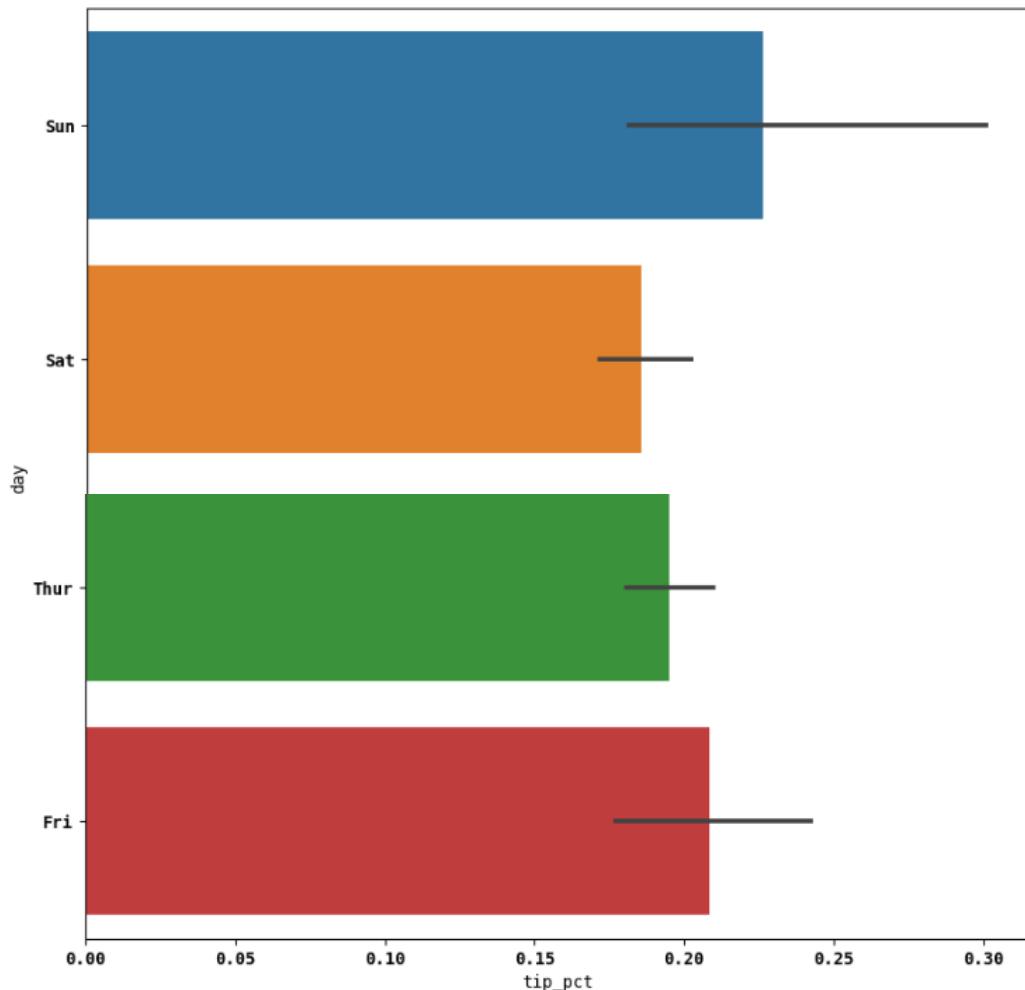
```
In [137]: tips.head()
```

Out[137]:

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01		No	Sun	Dinner	0.063204
1	10.34	1.66		No	Sun	Dinner	0.191244
2	21.01	3.50		No	Sun	Dinner	0.199886
3	23.68	3.31		No	Sun	Dinner	0.162494
4	24.59	3.61		No	Sun	Dinner	0.172069

```
In [138]: sns.barplot(x='tip_pct', y='day', data=tips, orient='h')
```

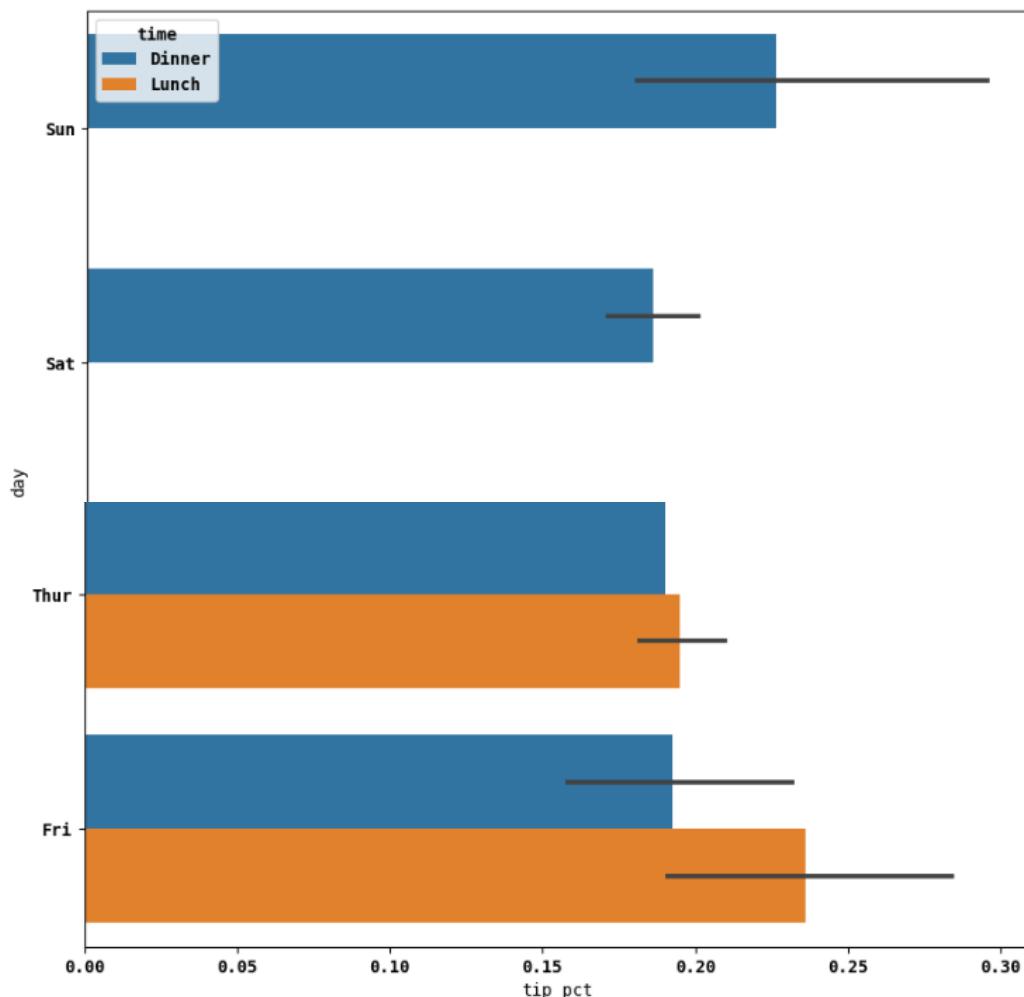
Out[138]: <Axes: xlabel='tip\_pct', ylabel='day'>



- Seaborn'daki çizim işlevleri, pandanın DataFrame'i olabilecek bir veri argümanı alır. Diğer argümanlar sütun adlarına atıfta bulunur. Gün içindeki her değer için birden fazla gözlem olduğundan çubuklar tip\_pct'nin ortalama değeridir. Çubukların üzerine çizilen siyah çizgiler %95 güven aralığını temsil eder (bu, isteğe bağlı bağımsız değişkenler aracılığıyla yapılandırılabilir).
- seaborn.barplot'un ek bir kategorik değere göre bölmemizi sağlayan bir renk tonu seçeneği var.

```
In [139]: sns.barplot(x='tip_pct', y='day', hue='time', data=tips, orient='h')
```

```
Out[139]: <Axes: xlabel='tip_pct', ylabel='day'>
```



- Seaborn plotların estetiğini otomatik olarak değiştirdiğine dikkat edin: varsayılan renk paleti, çizim arka planı ve ızgara çizgisi renkleri. Seaborn.set'i kullanarak farklı olay örgüsü görünümleri arasında geçiş yapabilirsiniz:

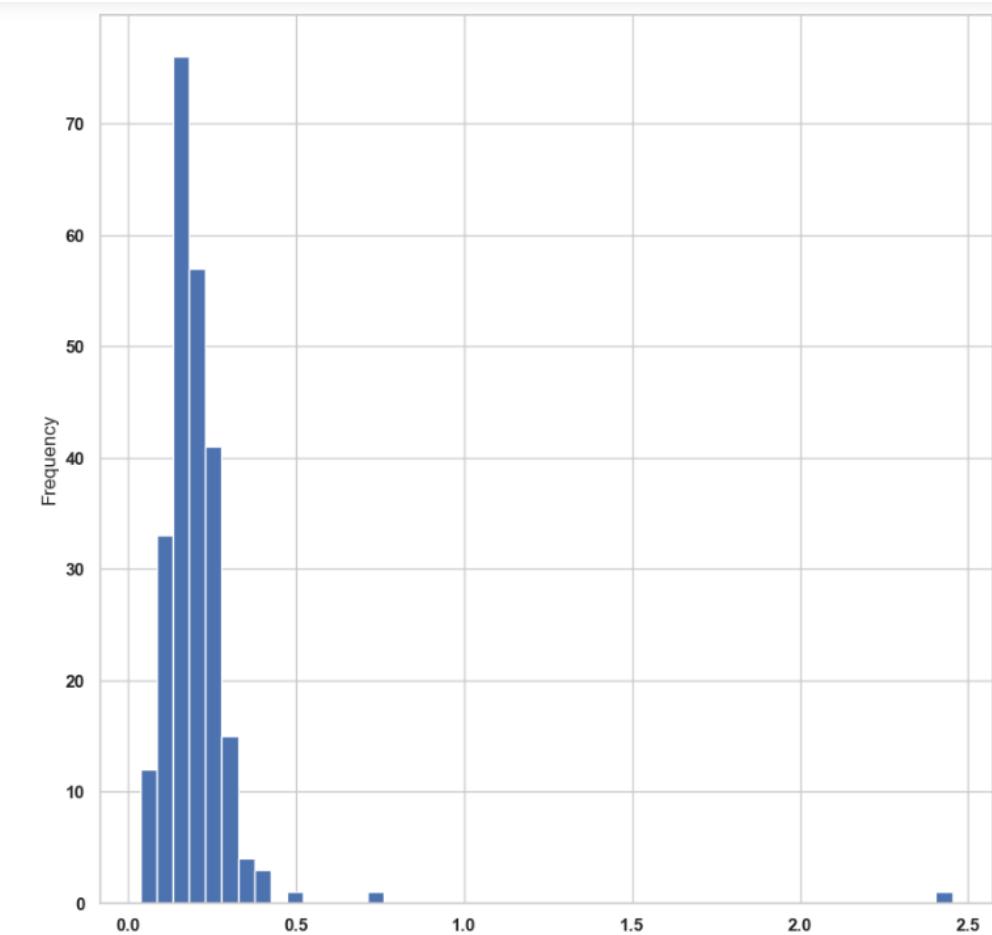
```
In [140]: sns.set(style="whitegrid")
```

## Histogramlar ve Yoğunluk Grafikleri

- Histogram, değer frekansının ayrıklığılmış bir görüntüsünü veren bir tür çubuk grafiğidir. Veri noktaları ayrı, eşit aralıklı bölmelere bölünür ve her bölmedeki veri noktalarının sayısı çizilir.

```
In [141]: tips['tip_pct'].plot.hist(bins=50)
```

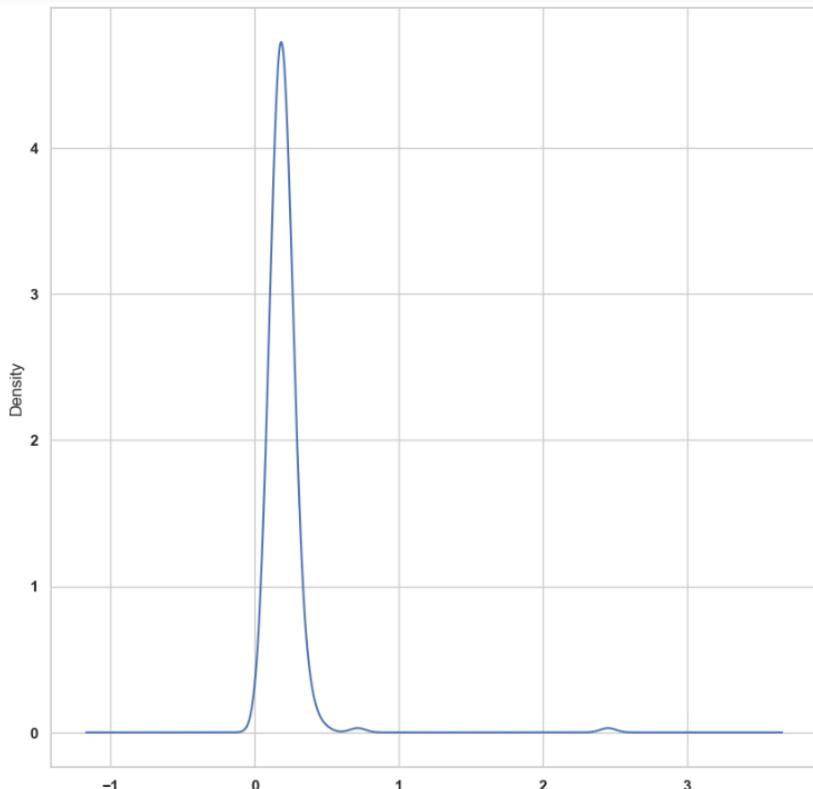
```
Out[141]: <Axes: ylabel='Frequency'>
```



- İlgili bir çizim türü, gözlemlenen verileri oluşturmuş olabilecek sürekli bir olasılık dağılımı tahmininin hesaplanmasıyla oluşturulan bir yoğunluk grafiğidir. Genel prosedür, bu dağılımı "çekirdeklerin" bir karışımı olarak, yani normal dağılım gibi daha basit dağılımlar olarak yaklaşık olarak hesaplamaktır. Bu nedenle yoğunluk grafikleri aynı zamanda çekirdek yoğunluk tahmini (KDE) grafikleri olarak da bilinir. Plot.kde'yi kullanmak, aşağıdakileri kullanarak bir yoğunluk grafiği oluşturur: geleneksel normallerin karışımı tahmini.

```
In [142]: tips['tip_pct'].plot.density()
```

```
Out[142]: <Axes: ylabel='Density'>
```



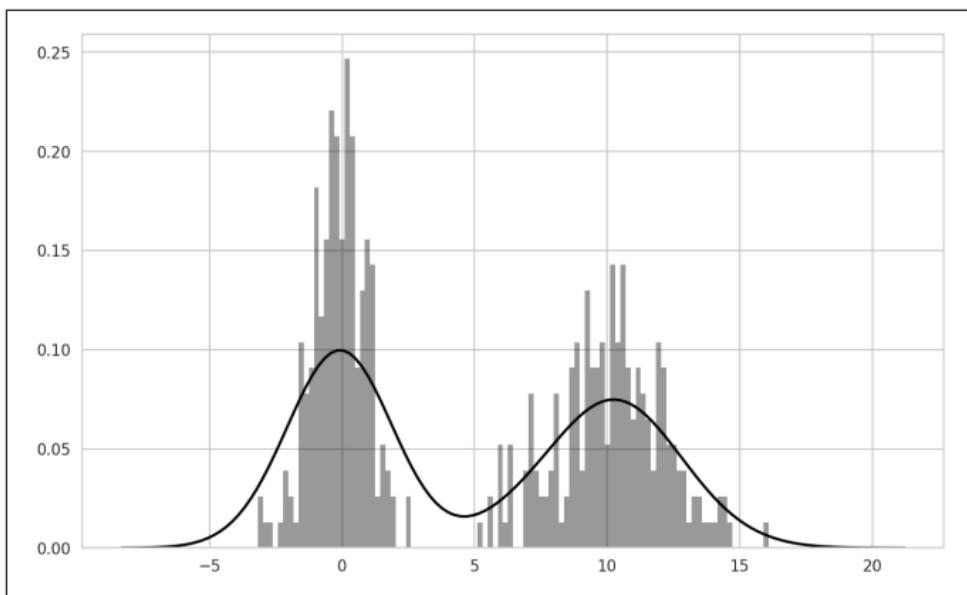
- Seaborn, hem histogramı hem de sürekli yoğunluk tahminini aynı anda çizebilen distplot yöntemiyle histogramları ve yoğunluk grafiklerini daha da kolaylaştırır. Örnek olarak, iki farklı standart normal dağılımdan oluşan iki modlu bir dağılımı düşünün.

```
In [100]: comp1 = np.random.normal(0, 1, size=200)
```

```
In [101]: comp2 = np.random.normal(10, 2, size=200)
```

```
In [102]: values = pd.Series(np.concatenate([comp1, comp2]))
```

```
In [103]: sns.distplot(values, bins=100, color='k')
```



- Seaborn, hem histogramı hem de sürekli yoğunluk tahminini aynı anda çizebilen distplot yöntemiyle histogramları ve yoğunluk grafiklerini daha da kolaylaştırır. Örnek olarak, iki farklı standart normal dağılımdan oluşan iki modlu bir dağılımı düşünün.

## Dağılım veya Nokta Grafikleri

```
In [105]: macro = pd.read_csv('macrodata (3).csv')
```

```
In [106]: data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]
```

```
In [107]: trans_data = np.log(data).diff().dropna()
```

```
In [108]: trans_data[-5:]
```

Out[108]:

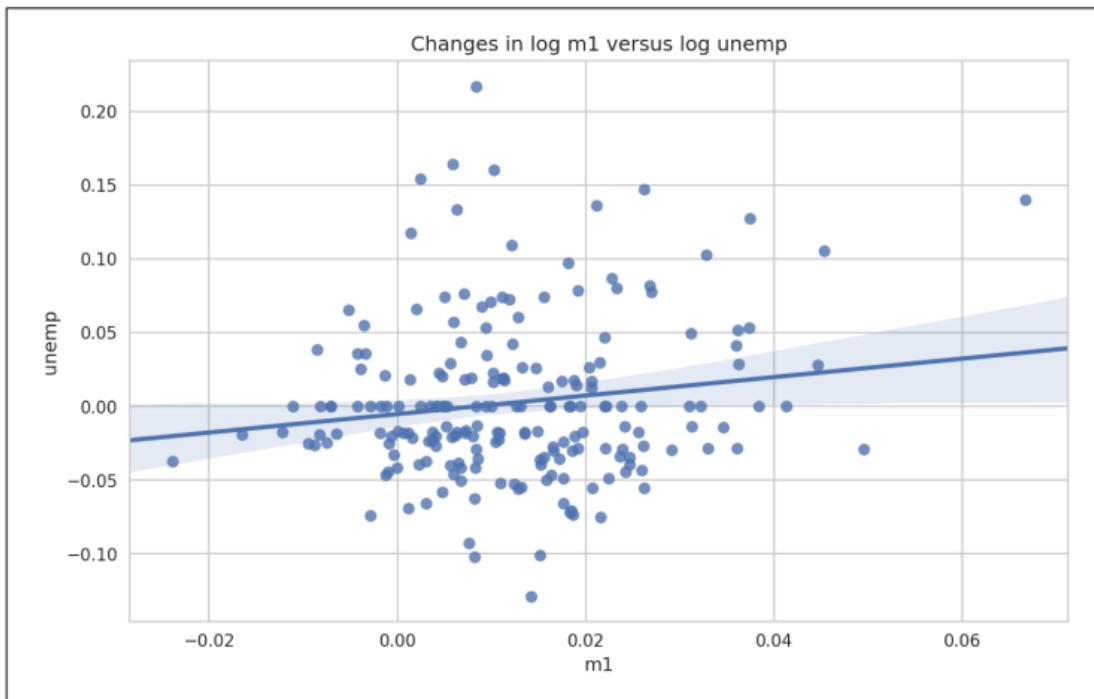
	cpi	m1	tbilrate	unemp
198	-0.007904	0.045361	-0.396881	0.105361
199	-0.021979	0.066753	-2.277267	0.139762
200	0.002340	0.010286	0.606136	0.160343
201	0.008419	0.037461	-0.200671	0.127339
202	0.008894	0.012202	-0.405465	0.042560

- Daha sonra bir dağılım grafiği oluşturan ve doğrusal bir regresyon çizgisine uyan Seaborn'un regplot yöntemini kullanabiliriz.

```
In [109]: sns.regplot('m1', 'unemp', data=trans_data)
```

```
In [110]: plt.title('Changes in log %s versus log %s' % ('m1', 'unemp'))
```

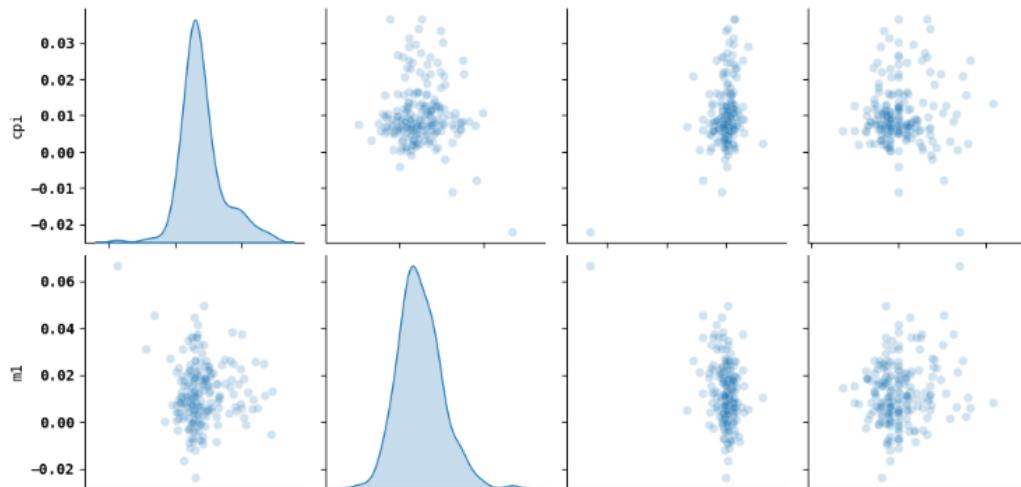
```
Out[110]: Text(0.5, 1.0, 'Changes in log m1 versus log unemp')
```

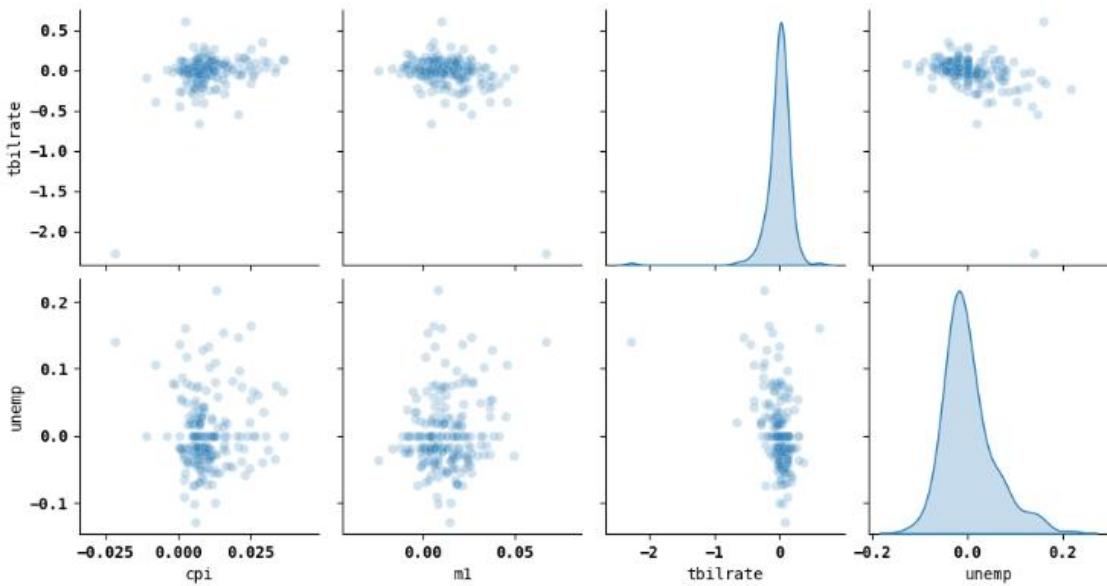


```
In [111]: sns.pairplot(trans_data, diag_kind='kde', plot_kws={'alpha': 0.2})
```

```
C:\Users\Eda\.templateengine\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

```
Out[111]: <seaborn.axisgrid.PairGrid at 0x23d1d0ff950>
```



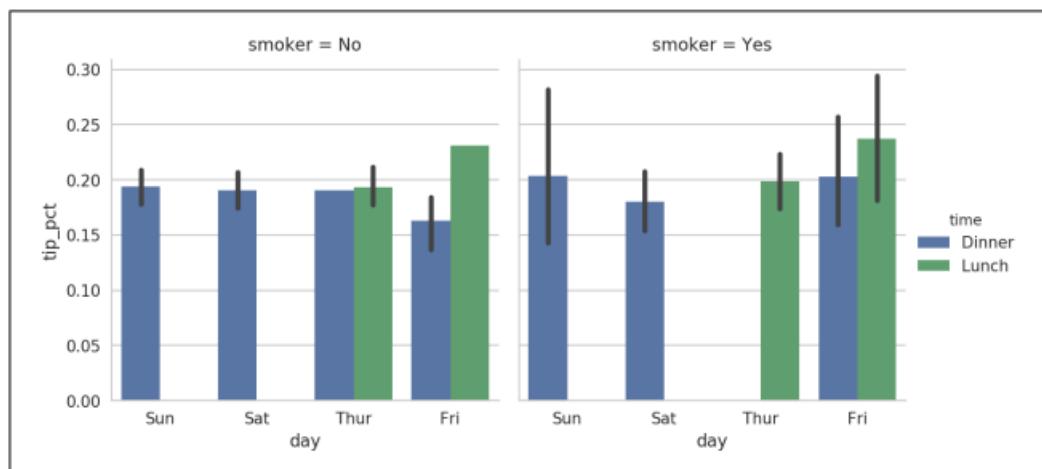


- plot\_kws argümanını fark edebilirsiniz. Bu, konfigürasyon seçeneklerini diyagonal olmayan elemanlar üzerindeki bireysel planlama çağrılarına aktarmamızı sağlar.

## Faset Izgaraları ve Kategorik Veriler

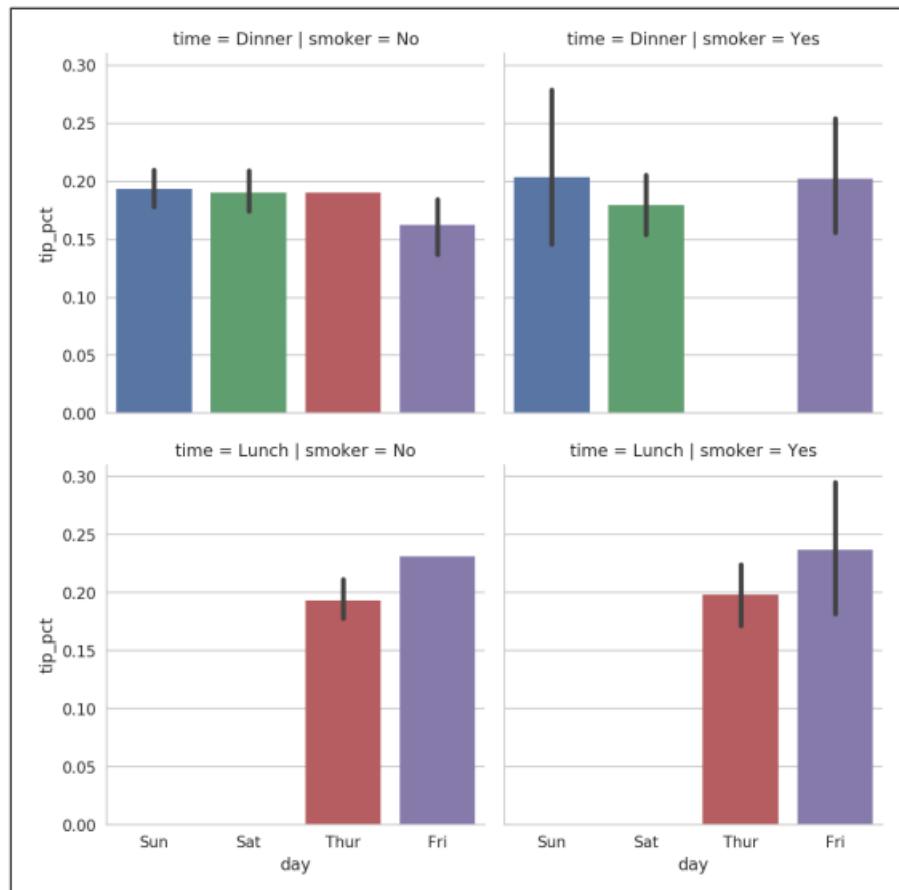
- Ek graplama boyutlarına sahip olduğumuz veri kümeleri ne olacak? Verileri birçok kategorik değişkenle görselleştirmenin bir yolu, faset izgarası kullanmaktadır. Seaborn'un kullanışlı bir özelliği var. Birçok çeşit yönlü grafiğin oluşturulmasını kolaylaştıran yerleşik fonksiyon faktör grafiği:

```
In [117]: sns.factorplot(x='day', y='tip_pct', hue='time', col='smoker',
kind='bar', data=tips[tips.tip_pct < 1])
```



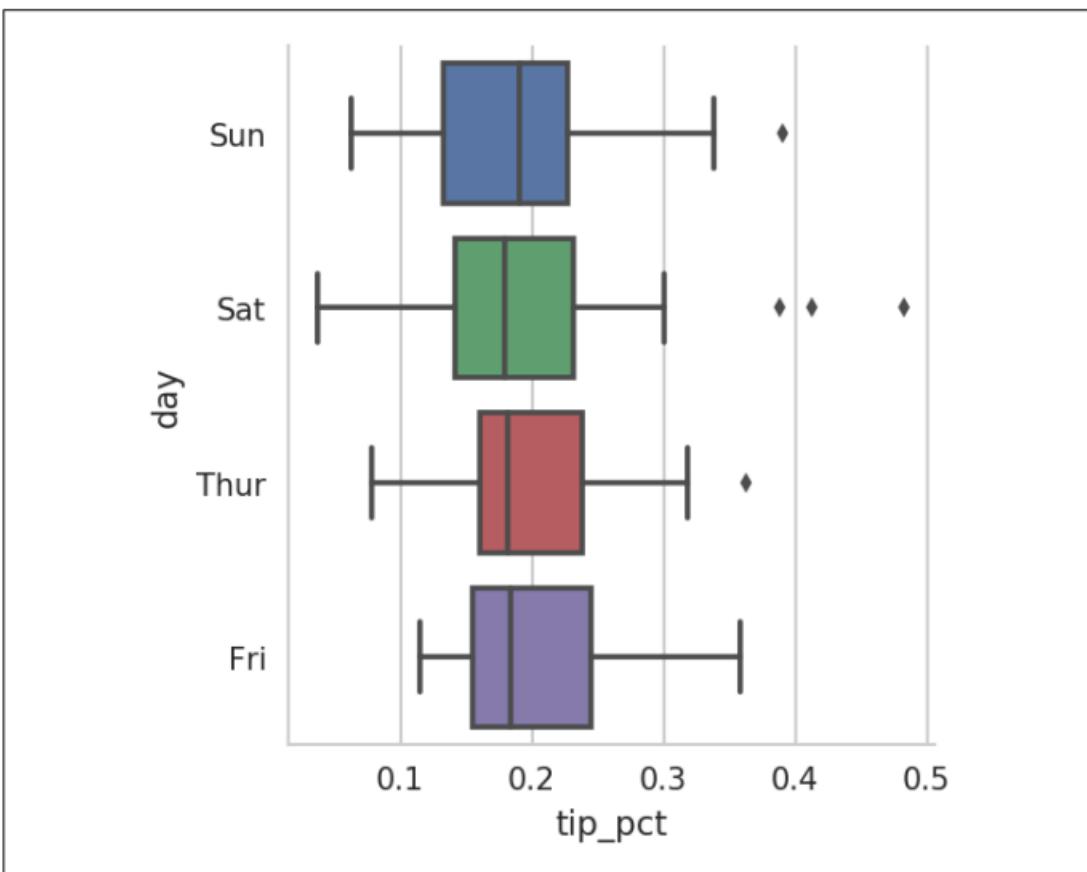
- Bir özellik içindeki farklı çubuk renklerine göre, 'zamana' göre gruplamak yerine, zaman değeri başına bir satır ekleyerek model izgarasını da genişletebiliriz

```
In [143]: sns.factorplot(x='day', y='tip_pct', row='time',
                      col='smoker',
                      kind='bar', data=tips[tips.tip_pct < 1])
```



- faktör grafiği, görüntülemeye çalışığınız şeye bağlı olarak faydalı olabilecek diğer çizim türlerini destekler. Örneğin, kutu grafikleri (medyanı, çeyrekleri ve aynırı değerleri gösteren) etkili bir görselleştirme türü olabilir.

```
In [144]: sns.factorplot(x='tip_pct', y='day', kind='box',
                      data=tips[tips.tip_pct < 0.5])
```



- Daha genel olan seaborn.FacetGrid sınıfını kullanarak kendi faset izgara grafiklerinizi oluşturabilirsiniz.

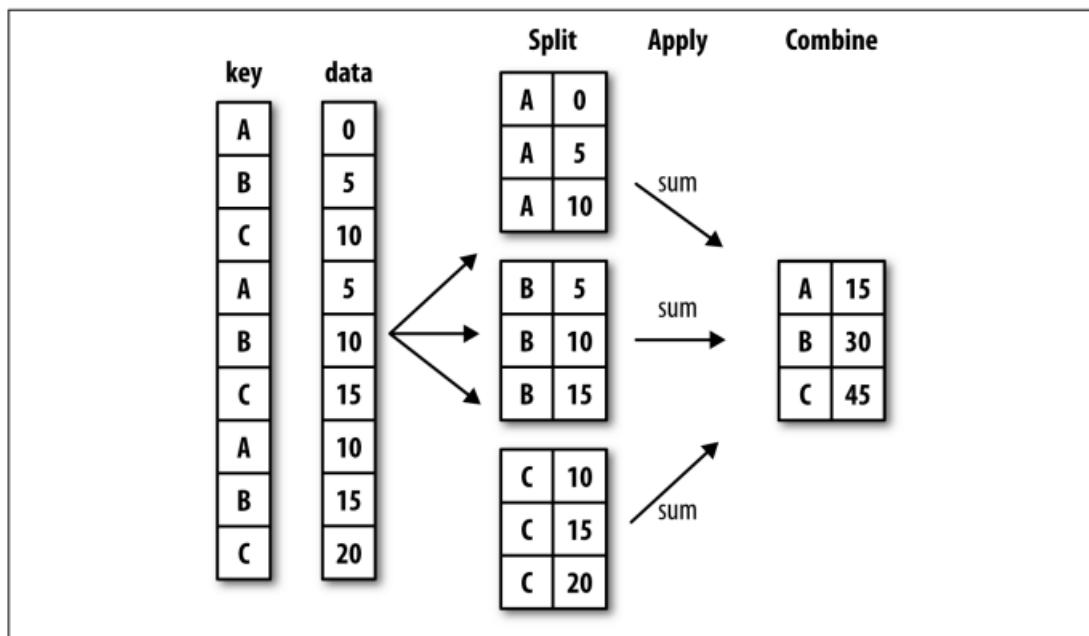
## Veri Toplama ve Grup Operasyonları

- Bir veri kümelerini kategorilere ayırmak ve ister toplama ister dönüştürme olsun, her gruba bir işlev uygulamak genellikle veri analizi iş akışının kritik bir bileşenidir. Bir veri kümelerini yükledikten, birleştirdikten ve hazırladıktan sonra raporlama veya görselleştirme amacıyla grup istatistiklerini veya muhtemelen pivot tabloları hesaplamaz gerekebilir. pandas, veri kümelerini doğal bir şekilde dilimlemenize, parçalara ayırmaya ve özetlemenize olanak tanıyan esnek bir gruplama arayüzü sağlar.

- İlişkisel veritabanlarının ve SQL'in ("yapilandırılmış sorgulama dili" anlamına gelir) popüleritesinin bir nedeni, verilerin birleştirilebilmesi, filtrelenebilmesi, dönüştürülebilmesi kolaylığıdır. Oluşturulmuş ve bir araya getirilmiştir. Ancak SQL gibi sorgulama dilleri, gerçekleştirebilecek grup işlemleri açısından bir miktar kısıtlıdır. Göreceğiniz gibi, Python ve pandasın ifade gücü sayesinde, bir pandas nesnesini veya NumPy dizisini kabul eden herhangi bir fonksiyonu kullanarak oldukça karmaşık grup işlemlerini gerçekleştirebiliriz.

## GroupBy Mekanikleri

- R programlama dili için birçok popüler paketin yazarı Hadley Wickham, grup işlemlerini tanımlamak için böl-ugula-birleştir terimini icat etti. İçinde sürecin ilk aşamasında, bir Seri, DataFrame veya başka bir pandas nesnesinde bulunan veriler, sağladığınız bir veya daha fazla anahtara göre gruptara ayrılır.
- Bölme, bir nesnenin belirli bir eksenin üzerinde gerçekleştirilir. Örneğin, bir DataFrame, satırlarına (eksen=0) veya sütunlarına (eksen=1) göre gruplandırılabilir. Bu yapıldıktan sonra her gruba yeni bir değer üreten bir işlev uygulanır. Son olarak, tüm bu işlev uygulamalarının sonuçları bir sonuç nesnesinde birleştirilir.



grup toplamanın çizimi

- Her gruplama anahtarı birçok biçimde olabilir ve anahtarların hepsinin aynı türde olması gerekmek:
- Gruplandırılan eksenle aynı uzunlukta olan bir değer listesi veya dizisi
- Bir değeri belirten DataFrame'deki sütun adı
- Gruplandırılan eksendeki değerler ile grup adları arasındaki karşılığı veren bir dict veya Series
- Eksen indeksinde veya indeksteki bireysel etiketlerde çağrılabilecek bir fonksiyon

```
In [147]: df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
                           'key2' : ['one', 'two', 'one', 'two', 'one'],
                           'data1' : np.random.randn(5),
                           'data2' : np.random.randn(5)})
```

```
In [148]: df
```

Out[148]:

	key1	key2	data1	data2
0	a	one	1.333848	-1.822735
1	a	two	-1.214233	0.495932
2	b	one	-0.118147	0.206244
3	b	two	-0.958841	0.215998
4	a	one	-1.903295	2.317593

- Anahtar1'deki etiketleri kullanarak data1 sütununun ortalamasını hesaplamak istedığınızı varsayıyoruz. Bunu yapmanın çeşitli yolları var. Bunlardan biri, data1'e erişmek ve key1'deki sütun (bir Seri) ile groupby'yi çağrılmaktır.

```
In [8]: grouped = df['data1'].groupby(df['key1'])
```

```
In [9]: grouped
```

Out[9]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x000001F456380F50>

- Bu gruplandırılmış değişken artık bir GroupBy nesnesidir. Aslında df['key1'] grup anahtarıyla ilgili bazı ara veriler dışında henüz hiçbir şey hesaplamadı. Fikir şu bu nesnenin daha sonra grupların her birine bazı işlemler uygulamak için gereken tüm bilgilere sahip olduğu. Örneğin, grubu hesaplamak için GroupBy'ın ortalama yöntemini çağrılabiliriz:

```
In [10]: grouped.mean()
```

Out[10]: key1  
a -0.734605  
b 0.206269  
Name: data1, dtype: float64

```
In [11]: means = df['data1'].groupby([df['key1'], df['key2']]).mean()

In [12]: means

Out[12]: key1  key2
          a    one    -0.185317
                  two    -1.833182
          b    one    -0.028764
                  two     0.441301
Name: data1, dtype: float64
```

- Burada verileri iki anahtar kullanarak grupladık ve ortaya çıkan Seri artık gözlemlenen benzersiz anahtar çiftlerinden oluşan hiyerarşik bir dizine sahip.

```
In [13]: means.unstack()

Out[13]:
      key2      one      two
      key1
      a   -0.185317  -1.833182
      b   -0.028764   0.441301

In [14]: states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])

In [15]: years = np.array([2005, 2005, 2006, 2005, 2006])

In [16]: df['data1'].groupby([states, years]).mean()

Out[16]: California  2005    -1.833182
                  2006    -0.028764
          Ohio       2005     0.046719
                  2006    -0.022770
Name: data1, dtype: float64
```

- Gruplandırma bilgileri sıklıkla üzerinde çalışmak istediğiniz verilerle aynı DataFrame'de bulunur. Bu durumda, sütun adlarını (bunlar dize, sayı veya diğer Python nesneleri olsun) grup anahtarları olarak iletebilirsiniz.

```
In [18]: df.groupby('key1').mean()

In [19]: df.groupby(['key1', 'key2']).mean()

Out[19]:
      data1      data2
      key1  key2
      a    one  -0.185317  0.331034
            two  -1.833182  0.840015
            b    one  -0.028764  0.999911
                  two   0.441301  0.824766
```

- İlk durumda df.groupby('key1').mean() sonucunda sonuçta key2 sütununun olmadığını fark etmiş olabilirsiniz. df['key2'] sayısal bir veri olmadığından, nuisance sütun olduğu söylenir ve bu nedenle sonuçtan hariç tutulur. Varsayılan olarak tüm sayısal sütunlar toplanır ve alt kümeye kadar filtrelemek de mümkündür.
- Gruplandırmayı kullanmanın amacı ne olursa olsun, genel olarak yararlı bir GroupBy yöntemi ve grup boyutlarını içeren bir seri döndürmektir.

```
In [20]: df.groupby(['key1', 'key2']).size()
```

```
Out[20]: key1  key2
          a    one      2
                  two      1
          b    one      1
                  two      1
          dtype: int64
```

## Gruplar Üzerinde Yineleme

- GroupBy nesnesi, veri yığınıyla birlikte grup adını içeren 2'li bir dizi oluşturarak yinelemeyi destekler.

```
In [21]: for name, group in df.groupby('key1'):
    print(name)
    print(group)
```

```
a
  key1  key2      data1      data2
0   a    one -0.347863 -0.041325
1   a    two -1.833182  0.840015
4   a    one -0.022770  0.703392
b
  key1  key2      data1      data2
2   b    one -0.028764  0.999911
3   b    two  0.441301  0.824766
```

```
In [22]: for (k1, k2), group in df.groupby(['key1', 'key2']):
    print((k1, k2))
    print(group)
```

```
('a', 'one')
  key1  key2      data1      data2
0   a    one -0.347863 -0.041325
4   a    one -0.022770  0.703392
('a', 'two')
  key1  key2      data1      data2
1   a    two -1.833182  0.840015
('b', 'one')
  key1  key2      data1      data2
2   b    one -0.028764  0.999911
('b', 'two')
  key1  key2      data1      data2
3   b    two  0.441301  0.824766
```

- Veri parçalarının bir dict'ini tek satırlık olarak hesaplamak:

```
In [23]: pieces = dict(list(df.groupby('key1')))
```

```
In [24]: pieces['b']
```

Out[24]:

	key1	key2	data1	data2
2	b	one	-0.028764	0.999911
3	b	two	0.441301	0.824766

```
In [25]: df.dtypes
```

```
Out[25]: key1      object
          key2      object
          data1     float64
          data2     float64
          dtype: object
```

```
In [26]: grouped = df.groupby(df.dtypes, axis=1)
```

- Grupları şu şekilde yazdırabiliriz:

```
In [27]: for dtype, group in grouped:
    print(dtype)
    print(group)
```

```
float64
      data1      data2
0 -0.347863 -0.041325
1 -1.833182  0.840015
2 -0.028764  0.999911
3  0.441301  0.824766
4 -0.022770  0.703392
object
  key1 key2
0     a   one
1     a   two
2     b   one
3     b   two
4     a   one
```

## Bir Sütun veya Sütun Alt Kümesini Seçme

- DataFrame'den oluşturulan bir GroupBy nesnesinin bir sütun adı veya sütun adları dizisi ile dizine eklenmesi, toplama için sütun alt kümesi etkisine sahiptir. Bu şu anlama gelir.

```
In [28]: df.groupby('key1')['data1']
df.groupby('key1')[['data2']]
```

```
Out[28]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001F45D834890>
```

```
In [29]: df[['data1']].groupby(df['key1'])
df[['data2']].groupby(df['key1'])
```

```
Out[29]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001F458F11890>
```

- Özellikle büyük veri kümeleri için yalnızca birkaç sütunun toplanması istenebilir. Örneğin, önceki veri kümelerinde, yalnızca data2 sütununun ortalamalarını hesaplamak ve sonucu DataFrame olarak almak için şunu yazabiliriz:

```
In [30]: df.groupby(['key1', 'key2'])[['data2']].mean()
```

```
Out[30]:
      data2
  key1  key2
    a   one  0.331034
        two  0.840015
    b   one  0.999911
        two  0.824766
```

- Bu indeksleme işlemi tarafından döndürülen nesne, bir liste veya dizi iletilirse gruplandırılmış bir DataFrame'dir veya yalnızca tek bir sütun adı skaler olarak aktarılırsa gruplandırılmış bir seridir.

```
In [31]: s_grouped = df.groupby(['key1', 'key2'])['data2']
```

```
In [32]: s_grouped
```

```
Out[32]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x000001F45D8D7310>
```

```
In [33]: s_grouped.mean()
```

```
Out[33]: key1  key2
    a   one      0.331034
        two      0.840015
    b   one      0.999911
        two      0.824766
Name: data2, dtype: float64
```

## Sözcükler ve Dizilerle Gruplama

```
In [34]: people = pd.DataFrame(np.random.randn(5, 5),
                           columns=['a', 'b', 'c', 'd', 'e'],
                           index=['Joe', 'Steve', 'Wes', 'Jim', 'Travis'])
```

```
In [35]: people.iloc[2:3, [1, 2]] = np.nan #NA değeri ekleme
```

```
In [36]: people
```

Out[36]:

	a	b	c	d	e
<b>Joe</b>	0.228942	0.513983	-0.963285	-0.495457	0.897208
<b>Steve</b>	1.176018	-1.238140	-0.461328	1.380002	0.797385
<b>Wes</b>	-1.386838	NaN	NaN	0.080698	-0.745481
<b>Jim</b>	-0.625174	-0.850007	-0.335212	-1.097315	1.141030
<b>Travis</b>	0.114730	0.351934	0.256703	0.741406	-0.206729

- Şimdi, sütunlar için bir grup yazışması olduğunu ve sütunları gruba göre toplanmak istedığını varsayıyalım:

```
In [37]: mapping = {'a': 'red', 'b': 'red', 'c': 'blue',
                  'd': 'blue', 'e': 'red', 'f' : 'orange'}
```

```
In [38]: by_column = people.groupby(mapping, axis=1)
```

```
In [39]: by_column.sum()
```

Out[39]:

	blue	red
<b>Joe</b>	-1.458742	1.640133
<b>Steve</b>	0.918674	0.735262
<b>Wes</b>	0.080698	-2.132319
<b>Jim</b>	-1.432527	-0.334150
<b>Travis</b>	0.998108	0.259935

```
In [40]: map_series = pd.Series(mapping)
```

```
In [41]: map_series
```

Out[41]:

a	red
b	red
c	blue
d	blue
e	red
f	orange
	dtype: object

## Fonksiyonlarla Gruplama

- Python işlevlerini kullanmak, bir dict veya Series ile karşılaşıldığında grup eşlemesini tanımlamanın daha genel bir yoludur. Grup anahtarı olarak iletilen herhangi bir işlev, dizin değeri başına bir kez çağrılabilecek ve dönüş değerleri grup adları olarak kullanılacaktır.

```
In [43]: people.groupby(len).sum()
```

```
Out[43]:
```

	a	b	c	d	e
3	-1.783070	-0.336024	-1.298497	-1.512075	1.292757
5	1.176018	-1.238140	-0.461328	1.380002	0.797385
6	0.114730	0.351934	0.256703	0.741406	-0.206729

```
In [44]: key_list = ['one', 'one', 'one', 'two', 'two']
```

```
In [45]: people.groupby([len, key_list]).min()
```

```
Out[45]:
```

	a	b	c	d	e
3	one	-1.386838	0.513983	-0.963285	-0.495457
	two	-0.625174	-0.850007	-0.335212	-1.097315
5	one	1.176018	-1.238140	-0.461328	1.380002
	two	0.114730	0.351934	0.256703	0.741406
6					-0.206729

## Dizin Düzeylerine Göre Gruplama

- Hiyerarşik olarak indekslenmiş veri kümelerinin eksen indeksi düzeylerinden birini kullanarak toplama yeteneğidir.

```
In [46]: columns = pd.MultiIndex.from_arrays([[ 'US', 'US', 'US', 'JP', 'JP'],
                                             [1, 3, 5, 1, 3]],
                                             names=['cty', 'tenor'])
```

```
In [47]: hier_df = pd.DataFrame(np.random.randn(4, 5), columns=columns)
```

```
In [48]: hier_df
```

```
Out[48]:
```

cty	US			JP		
	tenor	1	3	5	1	3
0	0.357649	0.619077	0.466824	0.060026	-0.776818	
1	-1.159130	-0.807472	-2.388609	-0.777451	-0.256909	
2	2.040350	-0.253406	-0.646530	1.309016	-0.984473	
3	-0.842593	1.322439	-0.438059	-0.875746	-1.302300	

- Seviyeye göre gruplamak için seviye anahtar sözcüğünü kullanarak seviye numarasını veya adını iletin.

```
In [49]: hier_df.groupby(level='cty', axis=1).count()
```

Out[49]:

cty	JP	US
0	2	3
1	2	3
2	2	3
3	2	3

## Veri Toplama

- Toplamalar, dizilerden skaler değerler üreten herhangi bir veri dönüşümünü ifade eder. Önceki örneklerde ortalama, sayıım dahil olmak üzere bunlardan birkaç kullanılmıştır.

## Optimize edilmiş gruplandırma yöntemleri

Function name	Description
count	Number of non-NA values in the group
sum	Sum of non-NA values
mean	Mean of non-NA values
median	Arithmetic median of non-NA values
std, var	Unbiased ( $n - 1$ denominator) standard deviation and variance
min, max	Minimum and maximum of non-NA values
prod	Product of non-NA values
first, last	First and last non-NA values

```
In [50]: df
```

Out[50]:

	key1	key2	data1	data2
0	a	one	-0.347863	-0.041325
1	a	two	-1.833182	0.840015
2	b	one	-0.028764	0.999911
3	b	two	0.441301	0.824766
4	a	one	-0.022770	0.703392

```
In [51]: grouped = df.groupby('key1')
```

```
In [52]: grouped['data1'].quantile(0.9)
```

```
Out[52]: key1
          a    -0.087789
          b     0.394295
          Name: data1, dtype: float64
```

- Her parça için `adet.quantile(0.9)` komutunu kullanır ve ardından bu sonuçları, sonuç nesnesinde birleştirir.
- Kendi toplama işlevlerinizi kullanmak için bir diziyi toplayan herhangi bir işlevi iletin toplama veya toplama yöntemi.

```
In [53]: def peak_to_peak(arr):
           return arr.max() - arr.min()
```

```
In [54]: grouped.agg(peak_to_peak)
```

```
In [55]: grouped.describe()
Out[55]:
          data1                               data2
   count  mean    std      min    25%    50%    75%  max  count  mean    std      min    25%    50%    75%  max
key1
  a    3.0 -0.734605  0.965181 -1.833182 -1.090523 -0.347863 -0.185317 -0.022770  3.0  0.500694  0.474347 -0.041325  0.331034  0.703392  0.771703  0.84
  b    2.0  0.206269  0.332386 -0.028764  0.088753  0.206269  0.323785  0.441301  2.0  0.912339  0.123847  0.824766  0.868552  0.912339  0.956125  0.98
```

## Sütun Bazlı ve Çok Fonksiyonlu Uygulama

```
In [56]: tips = pd.read_csv('tips (3).csv')
```

```
In [57]: tips['tip_pct'] = tips['tip'] / tips['total_bill']
```

```
In [58]: tips[:6]
```

```
Out[58]:
   total_bill  tip  smoker  day  time  size  tip_pct
0       16.99  1.01     No  Sun Dinner    2  0.059447
1       10.34  1.66     No  Sun Dinner    3  0.160542
2       21.01  3.50     No  Sun Dinner    3  0.166587
3       23.68  3.31     No  Sun Dinner    2  0.139780
4       24.59  3.61     No  Sun Dinner    4  0.146808
5       25.29  4.71     No  Sun Dinner    4  0.186240
```

- Daha önce gördüğünüz gibi, bir Seriyi veya bir DataFrame'in tüm sütunlarını toplamak, toplamanın istenen işlevle kullanılması veya ortalama veya gibi bir yöntemin çağrılması meselesidir.

```
In [59]: grouped = tips.groupby(['day', 'smoker'])
```

```
In [60]: grouped_pct = grouped['tip_pct']
```

```
In [61]: grouped_pct.agg('mean')
```

```
Out[61]: day  smoker
          Fri  No      0.151650
                  Yes     0.174783
          Sat  No      0.158048
                  Yes     0.147906
          Sun  No      0.160113
                  Yes     0.187250
          Thur  No      0.160298
                  Yes     0.163863
Name: tip_pct, dtype: float64
```

- Bunun yerine fonksiyonlar veya fonksiyon adları listesi iletirseniz, işlevlerden alınan sütun adlarını içeren bir DataFrame elde edersiniz.

```
In [63]: grouped_pct.agg(['mean', 'std', 'peak_to_peak'])
```

```
Out[63]:
           mean      std  peak_to_peak
day  smoker
Fri   No    0.151650  0.028123    0.067349
      Yes   0.174783  0.051293    0.159925
Sat   No    0.158048  0.039767    0.235193
      Yes   0.147906  0.061375    0.290095
Sun   No    0.160113  0.042347    0.193226
      Yes   0.187250  0.154134    0.644685
Thur  No    0.160298  0.038774    0.193350
      Yes   0.163863  0.039389    0.151240
```

- Burada, veri grupları üzerinde bağımsız olarak değerlendirme yapmak üzere toplamaya yönelik toplama işlevlerinin bir listesini aktardık.

- GroupBy'nin sütunlara verdiği adları kabul etmeniz gerekmektedir; özellikle lambda fonksiyonları '`<lambda>`' ismine sahiptir, bu da onların tanımlanmasını zorlaştırır (bir fonksiyonun `_name_` niteliğine bakarak bunu kendiniz görebilirsiniz).

```
In [64]: grouped_pct.agg([('foo', 'mean'), ('bar', np.std)])
```

Out[64]:

		foo	bar
day	smoker		
Fri	No	0.151650	0.028123
	Yes	0.174783	0.051293
Sat	No	0.158048	0.039767
	Yes	0.147906	0.061375
Sun	No	0.160113	0.042347
	Yes	0.187250	0.154134
Thur	No	0.160298	0.038774
	Yes	0.163863	0.039389

- DataFrame ile daha fazla seçenekiniz olur, çünkü tüm sütunlara uygulanacak işlevlerin bir listesini veya sütun başına farklı işlevleri belirleyebilirsiniz. Başlangıç olarak, `tip_pct` ve `total_bill` sütunları için aynı üç istatistiği hesaplamak istediğimizi varsayıyalım:

```
In [67]: functions = ['count', 'mean', 'max']
```

```
In [68]: result = grouped[['tip_pct', 'total_bill']].agg(functions)
```

Out[67]:

day	smoker	tip_pct			total_bill		
		count	mean	max	count	mean	max
Fri	No	4	0.151650	0.187735	4	18.420000	22.75
	Yes	15	0.174783	0.263480	15	16.813333	40.17
Sat	No	45	0.158048	0.291990	45	19.661778	48.33
	Yes	42	0.147906	0.325733	42	21.276667	50.81
Sun	No	57	0.160113	0.252672	57	20.506667	48.17
	Yes	19	0.187250	0.710345	19	24.120000	45.35
Thur	No	45	0.160298	0.266312	45	17.113111	41.19
	Yes	17	0.163863	0.241255	17	19.190588	43.11

- Ortaya çıkan DataFrame'in hiyerarşik sütunları var. Her sütunu ayrı ayrı toplayacak ve concat kullanarak sonuçları bir araya getirecek ve sütun adlarını anahtar argümanı olarak kullanacak.

```
In [69]: result['tip_pct']
```

		count	mean	max
day	smoker			
Fri	No	4	0.151650	0.187735
	Yes	15	0.174783	0.263480
Sat	No	45	0.158048	0.291990
	Yes	42	0.147906	0.325733
Sun	No	57	0.160113	0.252672
	Yes	19	0.187250	0.710345
Thur	No	45	0.160298	0.266312
	Yes	17	0.163863	0.241255

- Şimdi, bir veya daha fazla sütuna potansiyel olarak farklı işlevler uygulamak istediğiniz varsayıyalım. Bunu yapmak için, sütun adlarının şu ana kadar listelenen işlev özelliklerinden herhangi birine eşlenmesini içeren bir dict'i agg'ye iletin.

```
In [72]: grouped.agg({'tip' : np.max, 'size' : 'sum'})
```

```
Out[72]:
```

		tip	size
day	smoker		
Fri	No	3.50	9
	Yes	4.73	31
Sat	No	9.00	115
	Yes	10.00	104
Sun	No	6.00	167
	Yes	6.50	49
Thur	No	6.70	112
	Yes	5.00	40

```
In [73]: grouped.agg({'tip_pct' : ['min', 'max', 'mean', 'std'],
                     'size' : 'sum'})
```

```
Out[73]:
```

		tip_pct				size
day	smoker	min	max	mean	std	sum
Fri	No	0.120385	0.187735	0.151650	0.028123	9
	Yes	0.103555	0.263480	0.174783	0.051293	31
Sat	No	0.056797	0.291990	0.158048	0.039767	115
	Yes	0.035638	0.325733	0.147906	0.061375	104
Sun	No	0.059447	0.252672	0.160113	0.042347	167
	Yes	0.065660	0.710345	0.187250	0.154134	49
Thur	No	0.072961	0.266312	0.160298	0.038774	112
	Yes	0.090014	0.241255	0.163863	0.039389	40

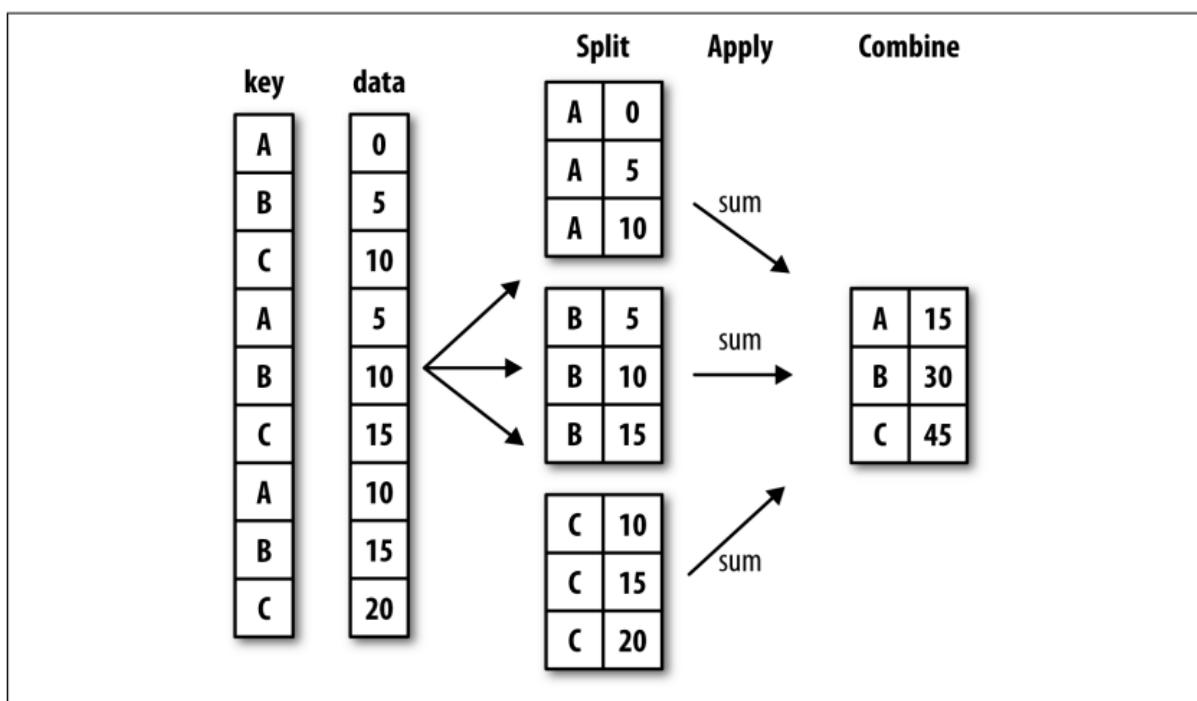
## Toplu Verileri Satır Indexleri Olmadan Döndürme

- Şu ana kadarki tüm örneklerde, bir araya getirilen veriler, benzersiz grup key kombinasyonlarından oluşan, potansiyel olarak hiyerarşik bir index'le geri geliyor. Bu her zaman tercih edilmediğinden, çoğu durumda `as_index=False` ögesini `groupby`'ye ileterek bu davranışını devre dışı bırakabilirsiniz.

```
In [74]: tips.groupby(['day', 'smoker'], as_index=False).mean()
```

## Apply: Genel split-apply-combine (böl - uygula - birleştir)

- Şekilde gösterildiği gibi, uygulama manipüle edilen nesneyi böler parçalara ayırır, her parçada aktarılan işlevi çağırır ve ardından parçaları bir araya getirmeye çalışır.



- Önceki tips veri kümesine dönersek, ilk beş `tip_pct` değerini grubu göre seçmek istedığınızı varsayıyalım. Öncelikle belirli bir sütundaki en büyük değerlere sahip satırları seçen bir fonksiyon yazın:

```
In [75]: def top(df, n=5, column='tip_pct'):
    return df.sort_values(by=column)[-n:]
```

```
In [76]: top(tips, n=6)
```

Out[76]:

	total_bill	tip	smoker	day	time	size	tip_pct
109	14.31	4.00	Yes	Sat	Dinner	2	0.279525
183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
232	11.61	3.39	No	Sat	Dinner	2	0.291990
67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

- Şimdi, sigara içen kişiye göre gruplandırırsak, diyelim ve bu işlevle başvuruyu çağırırsak, aşağıdakileri elde ederiz.

```
In [78]: tips.groupby('smoker').apply(top)
```

Out[78]:

		total_bill	tip	smoker	day	time	size	tip_pct
<b>smoker</b>								
No	88	24.71	5.85	No	Thur	Lunch	2	0.236746
	185	20.69	5.00	No	Sun	Dinner	5	0.241663
	51	10.29	2.60	No	Sun	Dinner	2	0.252672
	149	7.51	2.00	No	Thur	Lunch	2	0.266312
	232	11.61	3.39	No	Sat	Dinner	2	0.291990
Yes	109	14.31	4.00	Yes	Sat	Dinner	2	0.279525
	183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
	67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
	178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
	172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

- DataFrame'deki her satır grubunda top işlevi çağrılır ve ardından sonuçlar pandas.concat kullanılarak birbirine yapıştırılır ve parçalar grup adlarıyla etiketlenir. Bu nedenle sonuç, iç düzeyi orijinal DataFrame'den index değerlerini içeren hiyerarşik bir dizine sahiptir.
- GroupBox nesnesini tanımlayın.

```
In [79]: tips.groupby(['smoker', 'day']).apply(top, n=1, column='total_bill')
```

```
Out[79]:
```

smoker	day		total_bill	tip	smoker	day	time	size	tip_pct
No	Fri	94	22.75	3.25	No	Fri	Dinner	2	0.142857
	Sat	212	48.33	9.00	No	Sat	Dinner	4	0.186220
	Sun	156	48.17	5.00	No	Sun	Dinner	6	0.103799
	Thur	142	41.19	5.00	No	Thur	Lunch	5	0.121389
Yes	Fri	95	40.17	4.73	Yes	Fri	Dinner	4	0.117750
	Sat	170	50.81	10.00	Yes	Sat	Dinner	3	0.196812
	Sun	182	45.35	3.50	Yes	Sun	Dinner	3	0.077178
	Thur	197	43.11	5.00	Yes	Thur	Lunch	4	0.115982

```
In [80]: result = tips.groupby('smoker')['tip_pct'].describe()
```

```
In [81]: result
```

smoker	count	mean	std	min	25%	50%	75%	max
No	151.0	0.159328	0.039910	0.056797	0.136906	0.155625	0.185014	0.291990
Yes	93.0	0.163196	0.085119	0.035638	0.106771	0.153846	0.195059	0.710345

```
In [82]: result.unstack('smoker')
```

```
Out[82]:
```

	smoker	
count	No	151.000000
	Yes	93.000000
mean	No	0.159328
	Yes	0.163196
std	No	0.039910
	Yes	0.085119
min	No	0.056797
	Yes	0.035638
25%	No	0.136906
	Yes	0.106771
50%	No	0.155625
	Yes	0.153846
75%	No	0.185014
	Yes	0.195059
max	No	0.291990
	Yes	0.710345

dtype: float64

- GroupBy'nin içinde, tanımla gibi bir yöntemi çağrıdığınızda, bu aslında aşağıdakiler için yalnızca bir kısayoldur:

```
In [83]: f = lambda x: x.describe()
grouped.apply(f)
```

Out[83]:

			total_bill	tip	size	tip_pct
day	smoker					
Fri	No	count	4.000000	4.000000	4.00	4.000000
		mean	18.420000	2.812500	2.25	0.151650
		std	5.059282	0.898494	0.50	0.028123
		min	12.460000	1.500000	2.00	0.120385
		25%	15.100000	2.625000	2.00	0.137239
...	...	...	...	...	...	...
Thur	Yes	min	10.340000	2.000000	2.00	0.090014
		25%	13.510000	2.000000	2.00	0.148038
		50%	16.470000	2.560000	2.00	0.153846
		75%	19.810000	4.000000	2.00	0.194837
		max	43.110000	5.000000	4.00	0.241255

64 rows × 4 columns

## Grup Anahtarlarını Ortadan Kaldırmak

- Önceki örneklerde, ortaya çıkan nesnenin, orijinal nesnenin her bir parçasının dizinleriyle birlikte grup anahtarlarından oluşan hiyerarşik bir dizine sahip olduğunu görüyorsunuz. Bunu group\_keys=False komutunu groupby'ye ileterek devre dışı bırakabilirsiniz.

```
In [84]: tips.groupby('smoker', group_keys=False).apply(top)
```

Out[84]:

	total_bill	tip	smoker	day	time	size	tip_pct
88	24.71	5.85	No	Thur	Lunch	2	0.236746
185	20.69	5.00	No	Sun	Dinner	5	0.241663
51	10.29	2.60	No	Sun	Dinner	2	0.252672
149	7.51	2.00	No	Thur	Lunch	2	0.266312
232	11.61	3.39	No	Sat	Dinner	2	0.291990
109	14.31	4.00	Yes	Sat	Dinner	2	0.279525
183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

## Kantil ve Kova Analizi

- Pandaların, verileri seçtiğiniz kutularla veya örnek niceliklerle bölmelere dilimlemek için özellikle cut ve qcut gibi bazı araçları vardır. Bu işlevleri groupby ile birleştirmek, bucket veya quantile işlemlerini gerçekleştirmeyi kolaylaştırır.
- Basit bir rastgele veri kümesini ve kesmeyi kullanarak eşit uzunlukta bir grup kategorizasyonunu düşünün:

```
In [85]: frame = pd.DataFrame({'data1': np.random.randn(1000),
                             'data2': np.random.randn(1000)})
```

```
In [86]: quartiles = pd.cut(frame.data1, 4)
```

```
In [87]: quartiles[:10]
```

```
Out[87]: 0      (0.0783, 1.655]
          1     (-1.498, 0.0783]
          2     (-1.498, 0.0783]
          3     (-1.498, 0.0783]
          4     (0.0783, 1.655]
          5     (-1.498, 0.0783]
          6     (0.0783, 1.655]
          7     (0.0783, 1.655]
          8     (-1.498, 0.0783]
          9     (0.0783, 1.655]
Name: data1, dtype: category
Categories (4, interval[float64, right]): [(-3.08, -1.498] < (-1.498, 0.078
3] < (0.0783, 1.655] < (1.655, 3.231]]
```

- Cut tarafından döndürülen Kategorik nesne doğrudan groupby'ye aktarılabilir. Böylece data2 sütunu için şu şekilde bir istatistik kümesi hesaplayabiliriz:

```
In [88]: def get_stats(group):
           return {'min': group.min(), 'max': group.max(),
                   'count': group.count(), 'mean': group.mean()}
```

```
In [89]: grouped = frame.data2.groupby(quartiles)
```

```
In [90]: grouped.apply(get_stats).unstack()
```

```
out[90]:
```

	min	max	count	mean
data1				
(-3.08, -1.498]	-1.665190	3.083124	63.0	0.196110
(-1.498, 0.0783]	-2.722140	3.384366	455.0	0.038250
(0.0783, 1.655]	-2.669148	3.624345	424.0	-0.013742
(1.655, 3.231]	-1.956236	2.506811	58.0	0.067131

- Bunlar eşit uzunlukta kovalardı; örneğe dayalı olarak eşit boyutlu kapları hesaplamak için yüzdelik dilimler için qcut kullanın. Sadece niceliksel sayıları elde etmek için labels=False değerini iletin.

```
In [91]: grouping = pd.qcut(frame.data1, 10, labels=False)
```

```
In [92]: grouped = frame.data2.groupby(grouping)
```

```
In [93]: grouped.apply(get_stats).unstack()
```

Out[93]:

	min	max	count	mean
<b>data1</b>				
0	-2.034553	3.083124	100.0	0.103528
1	-2.722140	1.976994	100.0	-0.073521
2	-2.341110	2.721289	100.0	0.077512
3	-2.116507	2.141797	100.0	0.010238
4	-1.814770	3.384366	100.0	0.121344
5	-2.526597	2.583939	100.0	-0.130912
6	-2.669148	2.810023	100.0	-0.048476
7	-2.524792	2.246476	100.0	-0.034756
8	-2.172900	2.967996	100.0	0.151604
9	-1.986553	3.624345	100.0	0.101695

## Örnek: Eksik Değerleri Gruba Özgü Değerlerle Doldurma

```
In [94]: s = pd.Series(np.random.randn(6))
```

```
In [95]: s[::2] = np.nan
```

```
In [96]: s
```

```
Out[96]: 0      NaN
         1      2.955035
         2      NaN
         3     -0.693838
         4      NaN
         5      0.019771
dtype: float64
```

- Dizide 0. indexten başlayıp ikişer ikişer artarak değerleri NaN yapır.

```
In [97]: s.fillna(s.mean())
```

```
out[97]: 0    0.760323
          1    2.955035
          2    0.760323
          3   -0.693838
          4    0.760323
          5    0.019771
dtype: float64
```

- Doldurma değerinin gruba göre değişmesine ihtiyacınız olduğunu varsayıalım. Bunu yapmanın bir yolu, verileri grüplamak ve her veriobreğinde fillna'yı çağırın bir işlevle uygulama kullanmaktır.

```
In [98]: states = ['Ohio', 'New York', 'Vermont', 'Florida',
                 'Oregon', 'Nevada', 'California', 'Idaho']
```

```
In [99]: group_key = ['East'] * 4 + ['West'] * 4
```

```
In [100]: data = pd.Series(np.random.randn(8), index=states)
```

```
In [101]: data
```

```
Out[101]: Ohio        0.845023
           New York    0.501454
           Vermont     -0.810567
           Florida      1.032399
           Oregon      -0.042195
           Nevada       0.620024
           California   -0.550067
           Idaho        0.512447
dtype: float64
```

- ['East'] \* 4 sözdiziminin, ['East'] içindeki öğelerin dört kopyasını içeren bir liste oluşturduğunu unutmayın. Listelerin birbirine eklenmesi onları birleştirir.

```
In [102]: data[['Vermont', 'Nevada', 'Idaho']] = np.nan
```

```
In [103]: data
```

```
Out[103]: Ohio        0.845023
           New York    0.501454
           Vermont     NaN
           Florida      1.032399
           Oregon      -0.042195
           Nevada       NaN
           California   -0.550067
           Idaho        NaN
dtype: float64
```

```
In [104]: data.groupby(group_key).mean()
```

```
Out[104]: East    0.792959  
West   -0.296131  
dtype: float64
```

```
In [105]: fill_mean = lambda g: g.fillna(g.mean())
```

```
In [106]: data.groupby(group_key).apply(fill_mean)
```

```
Out[106]: East    Ohio        0.845023  
              New York     0.501454  
              Vermont      0.792959  
              Florida       1.032399  
             West    Oregon     -0.042195  
              Nevada      -0.296131  
              California   -0.550067  
              Idaho        -0.296131  
dtype: float64
```

```
In [107]: fill_values = {'East': 0.5, 'West': -1}
```

```
In [108]: fill_func = lambda g: g.fillna(fill_values[g.name])
```

```
In [109]: data.groupby(group_key).apply(fill_func)
```

```
Out[109]: East    Ohio        0.845023  
              New York     0.501454  
              Vermont      0.500000  
              Florida       1.032399  
             West    Oregon     -0.042195  
              Nevada      -1.000000  
              California   -0.550067  
              Idaho        -1.000000  
dtype: float64
```

## Örnek: Rastgele Örneklemme ve Permütasyon

- Büyük bir veri kümesinden rastgele bir örnek (değiştirerek veya değiştirmeden) çizmek istediğiniz varsayılmı. Orada “draws” gerçekleştirmenin çeşitli yolları vardır; burada Seriler için örnek yöntemi kullanıyoruz.
- Bunu göstermek için, İngiliz tarzı oyun kağıdı destesi oluşturmanın bir yolunu burada bulabilirsiniz:

```
In [113]: # Kupa, Maça, Sinek, Karo
suits = ['H', 'S', 'C', 'D']
card_val = (list(range(1, 11)) + [10] * 3) * 4
base_names = ['A'] + list(range(2, 11)) + ['J', 'K', 'Q']
cards = []
for suit in ['H', 'S', 'C', 'D']:
    cards.extend(str(num) + suit for num in base_names)
```

```
In [111]: deck = pd.Series(card_val, index=cards)
```

```
In [112]: deck[:13]
```

```
Out[112]: AH      1
2H      2
3H      3
4H      4
5H      5
6H      6
7H      7
8H      8
9H      9
10H     10
JH      10
KH      10
QH      10
dtype: int64
```

- Artık dizini Blackjack ve diğer oyunlarda kullanılan kart adlarını ve değerlerini içeren 52 uzunlığında bir Serimiz var.

```
In [114]: deck[:13]
```

```
Out[114]: AH      1
2H      2
3H      3
4H      4
5H      5
6H      6
7H      7
8H      8
9H      9
10H     10
JH      10
KH      10
QH      10
dtype: int64
```

- Daha önce söylenilenlere dayanarak desteden beş kartlık bir el çekmek şu şekilde yazılabılır:

```
In [115]: def draw(deck, n=5):
    return deck.sample(n)
```

```
In [116]: draw(deck)
```

```
Out[116]: 3C      3
          4S      4
          QD     10
          4C      4
          4H      4
          dtype: int64
```

- Her türden iki rastgele kart istedığınızı varsayalım. Renk, her kart adının son karakteri olduğundan, buna göre grupperlabilir ve uygulamayı kullanabiliriz:

```
In [118]: get_suit = lambda card: card[-1] # son harf uygun
```

```
In [119]: deck.groupby(get_suit).apply(draw, n=2)
```

```
Out[119]: C   3C    3
          4C    4
          D   9D    9
          8D    8
          H   8H    8
          4H    4
          S   6S    6
          5S    5
          dtype: int64
```

- Alternatif olarak bu şekilde de yazabiliriz.

```
In [120]: deck.groupby(get_suit, group_keys=False).apply(draw, n=2)
```

```
Out[120]: KC    10
          9C     9
          4D     4
          AD     1
          KH    10
          5H     5
          JS    10
          8S     8
          dtype: int64
```

## Örnek: Grup Ağırlıklı Ortalama ve Korelasyon

- Groupby'nin böl-uygula-birleştir paradigmı altında, grup ağırlıklı ortalama gibi bir DataFrame veya iki Serideki sütunlar arasındaki işlemler mümkündür.

```
In [121]: df = pd.DataFrame({'category': ['a', 'a', 'a', 'a',
                                         'b', 'b', 'b', 'b'],
                           'data': np.random.randn(8),
                           'weights': np.random.rand(8)})
```

```
In [122]: df
```

```
Out[122]:
```

	category	data	weights
0	a	-0.554530	0.573170
1	a	-0.382435	0.033382
2	a	0.693145	0.198885
3	a	0.003351	0.256744
4	b	-2.138820	0.401886
5	b	-1.171410	0.017226
6	b	-0.316815	0.680556
7	b	0.944048	0.260641

- Kategoriye göre grup ağırlıklı ortalaması şu şekilde olacaktır:

```
In [123]: grouped = df.groupby('category')
```

```
In [124]: get_wavg = lambda g: np.average(g['data'], weights=g['weights'])
```

```
In [125]: grouped.apply(get_wavg)
```

```
Out[125]: category
a    -0.180656
b    -0.624338
dtype: float64
```

```
In [127]: close_px = pd.read_csv('stock_px_2.csv', parse_dates=True,
                               index_col=0)
```

```
In [128]: close_px.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2214 entries, 2003-01-02 to 2011-10-14
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  -- 
 0   AAPL    2214 non-null   float64 
 1   MSFT    2214 non-null   float64 
 2   XOM    2214 non-null   float64 
 3   SPX     2214 non-null   float64 
dtypes: float64(4)
memory usage: 86.5 KB
```

```
In [129]: close_px[-4:]
```

```
Out[129]:
```

	AAPL	MSFT	XOM	SPX
2011-10-11	400.29	27.00	76.27	1195.54
2011-10-12	402.19	26.96	77.16	1207.25
2011-10-13	408.43	27.18	76.37	1203.66
2011-10-14	422.00	27.27	78.11	1224.58

- Günlük getirilerin (yüzde değişimlerden hesaplanan) SPX ile yıllık korelasyonlarından oluşan bir DataFrame'i hesaplama yapmanın bir yolu olarak öncelikle 'SPX' sütunuyla her sütunun ikili korelasyonunu hesaplayan bir fonksiyon yaratırız.

```
In [130]: spx_corr = lambda x: x.corrwith(x['SPX'])
```

- Daha sonra, pct\_change'i kullanarak close\_px'teki değişim yüzdesini hesaplıyoruz:

```
In [131]: rets = close_px.pct_change().dropna()
```

- Son olarak, bu yüzde değişikliklerini yıla göre grupperyoruz; bu, her tarih saat etiketinin yıl özelliğini döndüren tek satırlık bir işlevle her satır etiketinden çıkarılabilir:

```
In [132]: get_year = lambda x: x.year
```

```
In [133]: by_year = rets.groupby(get_year)
```

```
In [134]: by_year.apply(spx_corr)
```

Out[134]:

	AAPL	MSFT	XOM	SPX
2003	0.541124	0.745174	0.661265	1.0
2004	0.374283	0.588531	0.557742	1.0
2005	0.467540	0.562374	0.631010	1.0
2006	0.428267	0.406126	0.518514	1.0
2007	0.508118	0.658770	0.786264	1.0
2008	0.681434	0.804626	0.828303	1.0
2009	0.707103	0.654902	0.797921	1.0
2010	0.710105	0.730118	0.839057	1.0
2011	0.691931	0.800996	0.859975	1.0

- Sütunlar arası korelasyonları da hesaplayabilirsiniz. Burada Apple ile Microsoft arasındaki yıllık korelasyonu hesaplıyoruz:

```
In [135]: by_year.apply(lambda g: g['AAPL'].corr(g['MSFT']))
```

Out[135]:

2003	0.480868
2004	0.259024
2005	0.300093
2006	0.161735
2007	0.417738
2008	0.611901
2009	0.432738
2010	0.571946
2011	0.581987

dtype: float64

## Örnek: Grup Bazında Doğrusal Regresyon

- Onceki örnekle aynı temada, işlev bir pandas nesnesi veya skaler değer döndürdüğü sürece groupby'yi daha karmaşık grup bazında istatistiksel analiz gerçekleştirmek için kullanabilirsiniz. Örneğin aşağıdaki regresyon fonksiyonu her bir veri yiğini üzerinde sıradan bir en küçük kareler (OLS) regresyonu yürütür.

```
In [14]: import statsmodels.api as sm
def regress(data, yvar, xvars):
    Y = data[yvar]
    X = data[xvars]
    X['intercept'] = 1.
    result = sm.OLS(Y, X).fit()
    return result.params
```

```
In [15]: by_year.apply(regress, 'AAPL', ['SPX'])
```

Out[15]:

	SPX	intercept
2003	1.195406	0.000710
2004	1.363463	0.004201
2005	1.766415	0.003246
2006	1.645496	0.000080
2007	1.198761	0.003438
2008	0.968016	-0.001110
2009	0.879103	0.002954
2010	1.052608	0.001261
2011	0.806605	0.001514

## Pivot Tablolar ve Çapraz Tablolama

- Pivot tablo, elektronik tablo programlarında ve diğer veri analiz yazılımlarında sıkılıkla bulunan bir veri özetleme aracıdır. Bir veri tablosunu bir veya daha fazla anahtara göre toplar, Verileri, grup anahtarlarından bazıları satırlar boyunca ve bazıları sütunlar boyunca olacak şekilde bir dikdörtgen içinde düzenler.
- Tıps veri kümesine dönersek, güne ve sigara içen kişiye göre düzenlenmiş bir grup ortalamaları tablosu (varsayılan pivot\_tablo toplama türü) hesaplamak istediğiniz varsayılmış satırlar:

```
In [19]: tips.pivot_table(index=['day', 'smoker'])
```

		size	tip	tip_pct	total_bill
day	smoker				
Fri	No	2.250000	2.812500	0.151650	18.420000
	Yes	2.066667	2.714000	0.174783	16.813333
Sat	No	2.555556	3.102889	0.158048	19.661778
	Yes	2.476190	2.875476	0.147906	21.276667
Sun	No	2.929825	3.167895	0.160113	20.506667
	Yes	2.578947	3.516842	0.187250	24.120000
Thur	No	2.488889	2.673778	0.160298	17.113111
	Yes	2.352941	3.030000	0.163863	19.190588

- Bu doğrudan groupby ile üretilebilirdi. Şimdi, yalnızca tip\_pct ve size'yi toplamak ve ayrıca zamana göre grupperlemek istediğimizi varsayıyalım. Sigara içenler tablo sütunlarında, gün ise satırlardadır.

```
In [20]: tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
columns='smoker')
```

		size		tip_pct	
smoker		No	Yes	No	Yes
time	day				
Dinner	Fri	2.000000	2.222222	0.139622	0.165347
	Sat	2.555556	2.476190	0.158048	0.147906
	Sun	2.929825	2.578947	0.160113	0.187250
	Thur	2.000000	NaN	0.159744	NaN
Lunch	Fri	3.000000	1.833333	0.187735	0.188937
	Thur	2.500000	2.352941	0.160311	0.163863

Margins = True ileterek bu tabloyu kısmi toplamları içerecek şekilde genişletebiliriz. Bu, Tüm satır ve sütun etiketlerini ekleme etkisine sahiptir; karşılık gelen değerler, tek bir katmandaki tüm veriler için grup istatistikleridir:

```
In [22]: tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
columns='smoker', margins=True)
```

		size		tip_pct			
smoker		No	Yes	All	No	Yes	All
time	day						
Dinner	Fri	2.000000	2.222222	2.166667	0.139622	0.165347	0.158916
	Sat	2.555556	2.476190	2.517241	0.158048	0.147906	0.153152
	Sun	2.929825	2.578947	2.842105	0.160113	0.187250	0.166897
	Thur	2.000000	NaN	2.000000	0.159744	NaN	0.159744
Lunch	Fri	3.000000	1.833333	2.000000	0.187735	0.188937	0.188765
	Thur	2.500000	2.352941	2.459016	0.160311	0.163863	0.161301
All		2.668874	2.408602	2.569672	0.159328	0.163196	0.160803

- Farklı bir toplama işlevi kullanmak için onu aggfunc'a iletin. Örneğin 'count' veya len, size grup boyutlarının çapraz tablosunu (sayım (count) veya sıklık (frequency)) verecektir.

```
In [23]: tips.pivot_table('tip_pct', index=['time', 'smoker'], columns='day',
                           aggfunc=len, margins=True)
```

	day	Fri	Sat	Sun	Thur	All
time	smoker					
Dinner	No	3.0	45.0	57.0	1.0	106.0
	Yes	9.0	42.0	19.0	NaN	70.0
Lunch	No	1.0	NaN	NaN	44.0	45.0
	Yes	6.0	NaN	NaN	17.0	23.0
All		19.0	87.0	76.0	62.0	244.0

- Bazı kombinasyonlar boşsa (veya başka türlü NA), bir fill\_value iletmek isteyebilirsiniz:

```
In [24]: tips.pivot_table('tip_pct', index=['time', 'size', 'smoker'],
                           columns='day', aggfunc='mean', fill_value=0)
```

	day		Fri	Sat	Sun	Thur
time	size	smoker				
Dinner	1	No	0.000000	0.137931	0.000000	0.000000
		Yes	0.000000	0.325733	0.000000	0.000000
	2	No	0.139622	0.162705	0.168859	0.159744
		Yes	0.171297	0.148668	0.207893	0.000000
	3	No	0.000000	0.154661	0.152663	0.000000
		Yes	0.000000	0.144995	0.152660	0.000000
	4	No	0.000000	0.150096	0.148143	0.000000
		Yes	0.117750	0.124515	0.193370	0.000000
	5	No	0.000000	0.000000	0.206928	0.000000
		Yes	0.000000	0.106572	0.065660	0.000000
...						
Lunch	1	No	0.000000	0.000000	0.000000	0.181728
		Yes	0.223776	0.000000	0.000000	0.000000
	2	No	0.000000	0.000000	0.000000	0.166005
		Yes	0.181969	0.000000	0.000000	0.158843
	3	No	0.187735	0.000000	0.000000	0.084246
		Yes	0.000000	0.000000	0.000000	0.204952
	4	No	0.000000	0.000000	0.000000	0.138919
		Yes	0.000000	0.000000	0.000000	0.155410
	5	No	0.000000	0.000000	0.000000	0.121389
		Yes	0.000000	0.000000	0.000000	0.173706

[21 rows x 4 columns]

## pivot\_table Seçenekleri

Function name	Description
values	Column name or names to aggregate; by default aggregates all numeric columns
index	Column names or other group keys to group on the rows of the resulting pivot table
columns	Column names or other group keys to group on the columns of the resulting pivot table
aggfunc	Aggregation function or list of functions ('mean' by default); can be any function valid in a groupby context
fill_value	Replace missing values in result table
dropna	If True, do not include columns whose entries are all NA
margins	Add row/column subtotals and grand total (False by default)

## Çapraz Tablolar: Crosstab

```
In [138]: data
Out[138]:
   Sample Nationality   Handedness
0      1        USA  Right-handed
1      2        Japan  Left-handed
2      3        USA  Right-handed
3      4        Japan  Right-handed
4      5        Japan  Left-handed
5      6        Japan  Right-handed
6      7        USA  Right-handed
7      8        USA  Left-handed
8      9        Japan  Right-handed
9     10        USA  Right-handed
```

- Bazı anket analizlerinin bir parçası olarak, bu verileri uyruğa (nationality) ve el tercihine (handedness) göre özetlemek isteyebiliriz. Bunu yapmak için pivot\_table'i kullanabilirsiniz, ancak pandas.crosstab işlevi daha kullanışlı olabilir.

```
In [139]: pd.crosstab(data.Nationality, data.Handedness, margins=True)
Out[139]:
Handedness  Left-handed  Right-handed  All
Nationality
Japan          2            3      5
USA            1            4      5
All            3            7     10
```

- Çapraz tablonun ilk iki bağımsız değişkeninin her biri bir dizi, Seri veya dizi listesi olabilir.

```
In [140]: pd.crosstab([tips.time, tips.day], tips.smoker, margins=True)
Out[140]:
smoker      No  Yes  All
time   day
Dinner  Fri    3    9   12
        Sat   45   42   87
        Sun   57   19   76
        Thur   1    0    1
Lunch   Fri    1    6    7
        Thur  44   17   61
All       151   93  244
```

## Zaman Serileri

- Zaman serisi verileri, finans, ekonomi, ekoloji, sinir bilimi ve fizik gibi birçok farklı alanda yapılandırılmış verilerin önemli bir biçimidir. Zamanın birçok noktasında gözlemlenen veya ölçülen her şey bir zaman serisi oluşturur. Çoğu zaman serisi sabit frekanslıdır; yani veri noktaları belirli kurallara göre düzenli aralıklarla, örneğin her 15 saniyede bir, her 5 dakikada bir veya ayda bir kez meydana gelir. Zaman serileri ayrıca sabit bir zaman birimi veya birimler arasında kayma olmaksızın düzensizdir.
- Pandalar birçok yerleşik zaman serisi aracı ve veri algoritması sağlar. Çok büyük zaman serileriyle verimli bir şekilde çalışabilir ve düzensiz ve sabit frekanslı zaman serilerini kolayca dilimleyebilir, toplayabilir ve yeniden örnekleyebilirsiniz. Bu araçlardan bazıları özellikle finans ve ekonomi uygulamaları için kullanışlıdır.

## Tarih ve Saat Veri Türleri ve Araçları

- Python standart kütüphanesi tarih ve saat verilerinin yanı sıra takvimle ilgili işlevsellik için veri türlerini içerir.

```
In [1]: from datetime import datetime  
  
In [2]: now = datetime.now()  
  
In [3]: now  
  
Out[3]: datetime.datetime(2023, 12, 13, 23, 19, 3, 271468)  
  
In [4]: now.year, now.month, now.day  
  
Out[4]: (2023, 12, 13)
```

- datetime hem tarihi hem de saati mikrosaniyeye kadar saklar. timedelta, iki datetime nesnesi arasındaki zamansal farkı temsil eder.

```
In [5]: delta = datetime(2011, 1, 7) - datetime(2008, 6, 24, 8, 15)  
  
In [6]: delta  
  
Out[6]: datetime.timedelta(days=926, seconds=56700)  
  
In [7]: delta.days  
  
Out[7]: 926  
  
In [8]: delta.seconds  
  
Out[8]: 56700
```

- Yeni bir kaydırılmış nesne elde etmek için bir datetime nesnesine bir timedelta veya bunun katlarını ekleyebilir (veya çıkarabilirsiniz).

```
In [9]: from datetime import timedelta
In [10]: start = datetime(2011, 1, 7)
In [11]: start + timedelta(12)
Out[11]: datetime.datetime(2011, 1, 19, 0, 0)
In [12]: start - 2 * timedelta(12)
Out[12]: datetime.datetime(2010, 12, 14, 0, 0)
```

## Datetime Modülündeki Türler

Type	Description
date	Store calendar date (year, month, day) using the Gregorian calendar
time	Store time of day as hours, minutes, seconds, and microseconds
datetime	Stores both date and time
timedelta	Represents the difference between two datetime values (as days, seconds, and microseconds)
tzinfo	Base type for storing time zone information

## String ve Datetime Arasında Dönüşüm

- Pandas Timestamp nesnelerini, str veya strftime yöntemini kullanarak ve bir biçim belirterek dizeler olarak biçimlendirebilirsiniz.

```
In [13]: stamp = datetime(2011, 1, 3)
In [14]: str(stamp)
Out[14]: '2011-01-03 00:00:00'
In [15]: stamp.strftime('%Y-%m-%d')
Out[15]: '2011-01-03'
```

## Datetime Biçimi Belirtimi (ISO C89 Uyumlu)

Type	Description
%Y	Four-digit year
%y	Two-digit year
%m	Two-digit month [01, 12]
%d	Two-digit day [01, 31]
%H	Hour (24-hour clock) [00, 23]
%I	Hour (12-hour clock) [01, 12]
%M	Two-digit minute [00, 59]
%S	Second [00, 61] (seconds 60, 61 account for leap seconds)
%w	Weekday as integer [0 (Sunday), 6]
%U	Week number of the year [00, 53]; Sunday is considered the first day of the week, and days before the first Sunday of the year are "week 0"
%W	Week number of the year [00, 53]; Monday is considered the first day of the week, and days before the first Monday of the year are "week 0"
%z	UTC time zone offset as +HHMM or -HHMM; empty if time zone naive
%F	Shortcut for %Y-%m-%d (e.g., 2012-4-18)
%D	Shortcut for %m/%d/%y (e.g., 04/18/12)

- Datetime.strptime kullanarak dizeleri tarihlere dönüştürmek için aynı biçim kodlarını kullanabilirsiniz.

```
In [16]: value = '2011-01-03'

In [17]: datetime.strptime(value, '%Y-%m-%d')
Out[17]: datetime.datetime(2011, 1, 3, 0, 0)

In [18]: datestrs = ['7/6/2011', '8/6/2011']

In [19]: [datetime.strptime(x, '%m/%d/%Y') for x in datestrs]
Out[19]: [datetime.datetime(2011, 7, 6, 0, 0), datetime.datetime(2011, 8, 6, 0, 0)]
```

- datetime.strptime, bilinen bir formattaki bir tarihi ayırtmanın iyi bir yoludur. Parser.parse yöntemi kullanılarak otomatik yüklenir.

```
In [20]: from dateutil.parser import parse

In [21]: parse('2011-01-03')
Out[21]: datetime.datetime(2011, 1, 3, 0, 0)
```

- dateutil, insanların anlayabileceği tarih gösterimlerinin çoğunu ayrıştırma yeteneğine sahiptir.

```
In [22]: parse('Jan 31, 1997 10:45 PM')
```

```
Out[22]: datetime.datetime(1997, 1, 31, 22, 45)
```

- Uluslararası yerel ayarlarda, günün aydan önce görünmesi çok yaygındır, bu nedenle bunu belirtmek için dayfirst=True kullanılır.

```
In [23]: parse('6/12/2011', dayfirst=True)
```

```
Out[23]: datetime.datetime(2011, 12, 6, 0, 0)
```

- pandalar genellikle ister eksen dizini olarak ister DataFrame'de bir sütun olarak kullanılsın, tarih dizileriyle çalışmaya yönelikdir. to\_datetime yöntemi birçok farklı tarih gösterimini ayırtırır.

```
In [24]: datestrs = ['2011-07-06 12:00:00', '2011-08-06 00:00:00']
```

```
In [27]: pd.to_datetime(datestrs)
```

```
Out[27]: DatetimeIndex(['2011-07-06 12:00:00', '2011-08-06 00:00:00'], dtype='datetime64[ns]', freq=None)
```

- Ayrıca eksik sayılması gereken değerleri de işler (None, empty, string vb.).

```
In [28]: idx = pd.to_datetime(datestrs + [None])
```

```
In [29]: idx
```

```
Out[29]: DatetimeIndex(['2011-07-06 12:00:00', '2011-08-06 00:00:00', 'NaT'], dtype='datetime64[ns]', freq=None)
```

```
In [30]: idx[2]
```

```
Out[30]: NaT
```

```
In [31]: pd.isnull(idx)
```

```
Out[31]: array([False, False, True])
```

- NaT (Bir Zaman Değil), pandaların timestamp verileri için boş değeridir.

## Yerel Ayara Özgü Tarih Biçimlendirmesi

Type	Description
%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%c	Full date and time (e.g., 'Tue 01 May 2012 04:20:57 PM')
%p	Locale equivalent of AM or PM
%x	Locale-appropriate formatted date (e.g., in the United States, May 1, 2012 yields '05/01/2012')
%X	Locale-appropriate time (e.g., '04:24:12 PM')

## Zaman Serisi Temelleri

- Pandalardaki temel bir zaman serisi nesnesi türü, timestamps ile indekslenen bir Seridir ve genellikle pandaların dışında Python dizeleri veya datetime nesneleri olarak temsil edilir.

```
In [1]: from datetime import datetime  
  
In [2]: dates = [datetime(2011, 1, 2), datetime(2011, 1, 5),  
              datetime(2011, 1, 7), datetime(2011, 1, 8),  
              datetime(2011, 1, 10), datetime(2011, 1, 12)]  
  
In [9]: ts = pd.Series(np.random.randn(6), index=dates)  
  
In [10]: ts  
out[10]: 2011-01-02    -0.269283  
          2011-01-05    -0.688027  
          2011-01-07   -1.434519  
          2011-01-08    0.105629  
          2011-01-10   -0.998575  
          2011-01-12   -1.088790  
          dtype: float64
```

- Diğer Seriler gibi, farklı indekslenmiş zaman serileri arasındaki aritmetik işlemler otomatik olarak tarihlere göre hizalanır.

```
In [11]: ts + ts[::-2]  
out[11]: 2011-01-02    -0.538566  
          2011-01-05      NaN  
          2011-01-07   -2.869037  
          2011-01-08      NaN  
          2011-01-10   -1.997149  
          2011-01-12      NaN  
          dtype: float64
```

- `ts[::-2]`'nin `ts`'deki her ikinci öğeyi seçtiğini hatırlayın.
- `DatetimeIndex`'teki skaler değerler pandaların `Timestamp` nesneleridir.

## İndeksleme, Seçim, Alt Kümeleme

- Zaman serisi diğer pandalar gibi davranır. Etikete göre verileri indekslerken ve seçerken seri:

```
In [16]: stamp = ts.index[2]
```

```
In [17]: ts[stamp]
```

```
Out[17]: -1.4345187152125936
```

- Kolaylık sağlamak için tarih olarak yorumlanabilecek bir dize de iletebilirsiniz.

```
In [18]: ts['1/10/2011']
```

```
Out[18]: -0.9985746147532291
```

```
In [19]: ts['20110110']
```

```
Out[19]: -0.9985746147532291
```

- Daha uzun zaman serileri için bir yıl veya yalnızca bir yıl ve ay geçebilir veri dilimlerini kolayca seçmek için:

```
In [20]: longer_ts = pd.Series(np.random.randn(1000),
                           index=pd.date_range('1/1/2000', periods=1000))
```

```
In [21]: longer_ts
```

```
Out[21]:
2000-01-01    0.910034
2000-01-02   -2.731488
2000-01-03    1.942395
2000-01-04    1.525293
2000-01-05    0.989042
...
2002-09-22    0.108107
2002-09-23    2.308425
2002-09-24    2.641329
2002-09-25    1.636798
2002-09-26    0.459391
Freq: D, Length: 1000, dtype: float64
```

- Burada '2001' dizisi bir yıl olarak yorumlanır ve o zaman dilimini seçer. Bu, ayı belirtirseniz de işe yarar:

```
In [22]: longer_ts['2001-05']

Out[22]: 2001-05-01    -1.276762
          2001-05-02     1.989561
          2001-05-03    -0.887315
          2001-05-04    -0.848858
          2001-05-05     2.142190
          2001-05-06    -1.543825
          2001-05-07    -2.519996
          2001-05-08     0.500282
          2001-05-09     1.368691
          2001-05-10     0.952258
          2001-05-11    -0.099646
          2001-05-12    -0.586162
          2001-05-13     0.773846
          2001-05-14    -1.089723
          2001-05-15     0.548954
          2001-05-16     1.028654
          2001-05-17    -1.106401
          2001-05-18     0.797552
          2001-05-19     0.786008
          2001-05-20     0.182205
          2001-05-21     0.648641
          2001-05-22     0.719695
          2001-05-23    -0.586982
          2001-05-24    -0.065358
          2001-05-25     0.349775
          2001-05-26     0.473094
          2001-05-27    -0.670251
          2001-05-28    -0.614197
          2001-05-29     0.212153
          2001-05-30    -0.774203
          2001-05-31    -0.684294
          Freq: D, dtype: float64
```

- Datetime nesneleriyle dilimleme de işe yarar.

```
In [23]: ts[datetime(2011, 1, 7):]

Out[23]: 2011-01-07    -1.434519
          2011-01-08     0.105629
          2011-01-10    -0.998575
          2011-01-12    -1.088790
          dtype: float64
```

- Zaman serisi verilerinin çoğu kronolojik olarak sıralandığından, aralık sorgusu gerçekleştirmek için zaman serisinde yer almayan zaman damgalarını dilimleyebilirsiniz.

```
In [24]: ts['1/6/2011':'1/11/2011']

Out[24]: 2011-01-07    -1.434519
          2011-01-08     0.105629
          2011-01-10    -0.998575
          dtype: float64
```

- Bir Seriyi iki tarih arasında dilimleyen eşdeğer bir örnek yöntem olan truncate vardır:

```
In [25]: ts.truncate(after='1/9/2011')
```

```
Out[25]: 2011-01-02    -0.269283
          2011-01-05    -0.688027
          2011-01-07   -1.434519
          2011-01-08    0.105629
          dtype: float64
```

```
In [26]: dates = pd.date_range('1/1/2000', periods=100, freq='W-WED')
```

```
In [27]: long_df = pd.DataFrame(np.random.randn(100, 4),
                           index=dates,
                           columns=['Colorado', 'Texas',
                                     'New York', 'Ohio'])
```

```
In [28]: long_df.loc['5-2001']
```

```
Out[28]:
          Colorado    Texas  New York    Ohio
2001-05-02  0.190972  0.417419 -1.461011  0.283704
2001-05-09 -0.479271  1.055721  2.373928  1.099725
2001-05-16  0.093997  0.274434  0.134923 -0.224197
2001-05-23  0.957193  1.195712 -0.225109  0.546649
2001-05-30 -0.290118  0.505177  1.407976  0.729841
```

## Yinelenen İndekslere Sahip Zaman Serileri

- Bazı uygulamalarda belirli bir timestamp denk gelen birden fazla veri gözleme olabilir.

```
In [29]: dates = pd.DatetimeIndex(['1/1/2000', '1/2/2000', '1/2/2000',
                               '1/2/2000', '1/3/2000'])
```

```
In [30]: dup_ts = pd.Series(np.arange(5), index=dates)
```

```
In [31]: dup_ts
```

```
Out[31]: 2000-01-01    0
          2000-01-02    1
          2000-01-02    2
          2000-01-02    3
          2000-01-03    4
          dtype: int32
```

- Dizinin benzersiz olmadığını `is_unique` özelliğini kontrol ederek anlayabiliriz.

```
In [32]: dup_ts.index.is_unique
```

```
Out[32]: False
```

- Bu zaman serisine indeksleme artık zaman damgasının kopyalanıp kopyalanmadığını bağlı olarak ya skaler değerler ya da dilimler üretecektir.

```
In [33]: dup_ts['1/3/2000'] # kopyalanmadı
```

```
Out[33]: 4
```

```
In [34]: dup_ts['1/2/2000'] #kopyalandı
```

```
Out[34]: 2000-01-02    1
          2000-01-02    2
          2000-01-02    3
          dtype: int32
```

- Benzersiz olmayan zaman damgalarına sahip verileri toplamak istediğiniz varsayılm. Bunu yapmanın bir yolu `groupby`'yi kullanmak ve `seviye=0`'a geçmektir.

```
In [35]: grouped = dup_ts.groupby(level=0)
```

```
In [36]: grouped.mean()
```

```
Out[36]: 2000-01-01    0.0
          2000-01-02    2.0
          2000-01-03    4.0
          dtype: float64
```

```
In [37]: grouped.count()
```

```
Out[37]: 2000-01-01    1
          2000-01-02    3
          2000-01-03    1
          dtype: int64
```

## Tarih Aralıkları, Frekanslar ve Kaydırma

- Pandalardaki genel zaman serilerinin düzensiz olduğu varsayıılır; yani sabit bir frekansları yoktur. Birçok uygulama için bu yeterlidir. Bu, bir zaman serisine eksik değerlerin dahil edilmesi anlamına gelse bile, günlük, aylık veya 15 dakikada bir gibi sabit bir sıklığa göre çalışır. Neyse ki pandalar, yeniden örnekleme, frekans çıkarımı ve sabit frekanslı tarih aralıkları oluşturmak için eksiksiz bir standart zaman serisi frekansları ve araçlarına sahiptir.

```
In [38]: ts
```

```
Out[38]: 2011-01-02    -0.269283
          2011-01-05   -0.688027
          2011-01-07  -1.434519
          2011-01-08   0.105629
          2011-01-10  -0.998575
          2011-01-12  -1.088790
          dtype: float64
```

```
In [39]: resampler = ts.resample('D')
```

- 'D' dizisi günlük frekans olarak yorumlanır.

## Tarih Aralıkları Oluşturma

- pandas.date\_range, belirli bir tarihe göre belirtilen uzunlukta bir DatetimeIndex oluşturur.

```
In [40]: index = pd.date_range('2012-04-01', '2012-06-01')
```

```
In [41]: index
```

```
Out[41]: DatetimeIndex(['2012-04-01', '2012-04-02', '2012-04-03', '2012-04-04',
       '2012-04-05', '2012-04-06', '2012-04-07', '2012-04-08',
       '2012-04-09', '2012-04-10', '2012-04-11', '2012-04-12',
       '2012-04-13', '2012-04-14', '2012-04-15', '2012-04-16',
       '2012-04-17', '2012-04-18', '2012-04-19', '2012-04-20',
       '2012-04-21', '2012-04-22', '2012-04-23', '2012-04-24',
       '2012-04-25', '2012-04-26', '2012-04-27', '2012-04-28',
       '2012-04-29', '2012-04-30', '2012-05-01', '2012-05-02',
       '2012-05-03', '2012-05-04', '2012-05-05', '2012-05-06',
       '2012-05-07', '2012-05-08', '2012-05-09', '2012-05-10',
       '2012-05-11', '2012-05-12', '2012-05-13', '2012-05-14',
       '2012-05-15', '2012-05-16', '2012-05-17', '2012-05-18',
       '2012-05-19', '2012-05-20', '2012-05-21', '2012-05-22',
       '2012-05-23', '2012-05-24', '2012-05-25', '2012-05-26',
       '2012-05-27', '2012-05-28', '2012-05-29', '2012-05-30',
       '2012-05-31', '2012-06-01'],
      dtype='datetime64[ns]', freq='D')
```

```
In [42]: pd.date_range(start='2012-04-01', periods=20)
```

```
Out[42]: DatetimeIndex(['2012-04-01', '2012-04-02', '2012-04-03', '2012-04-04',
       '2012-04-05', '2012-04-06', '2012-04-07', '2012-04-08',
       '2012-04-09', '2012-04-10', '2012-04-11', '2012-04-12',
       '2012-04-13', '2012-04-14', '2012-04-15', '2012-04-16',
       '2012-04-17', '2012-04-18', '2012-04-19', '2012-04-20'],
      dtype='datetime64[ns]', freq='D')
```

```
In [43]: pd.date_range(end='2012-06-01', periods=20)
```

```
Out[43]: DatetimeIndex(['2012-05-13', '2012-05-14', '2012-05-15', '2012-05-16',
   '2012-05-17', '2012-05-18', '2012-05-19', '2012-05-20',
   '2012-05-21', '2012-05-22', '2012-05-23', '2012-05-24',
   '2012-05-25', '2012-05-26', '2012-05-27', '2012-05-28',
   '2012-05-29', '2012-05-30', '2012-05-31', '2012-06-01'],
  dtype='datetime64[ns]', freq='D')
```

- Başlangıç ve bitiş tarihleri, oluşturulan tarih dizini için kesin sınırları tanımlar. Örneğin, her ayın son iş gününü içeren bir tarih dizini istiyorsanız, 'BM' frekansını geçersiniz ve yalnızca tarih aralığına denk gelen veya bu tarih aralığının içinde olan tarihler dahil edilecek.

```
In [44]: pd.date_range('2000-01-01', '2000-12-01', freq='BM')
```

```
Out[44]: DatetimeIndex(['2000-01-31', '2000-02-29', '2000-03-31', '2000-04-28',
   '2000-05-31', '2000-06-30', '2000-07-31', '2000-08-31',
   '2000-09-29', '2000-10-31', '2000-11-30'],
  dtype='datetime64[ns]', freq='BM')
```

## Temel Zaman Serisi Frekansları (Kapsamlı Değil)

Alias	Offset type	Description
D	Day	Calendar daily
B	BusinessDay	Business daily
H	Hour	Hourly
T or min	Minute	Minutely
S	Second	Secondly
L or ms	Milli	Millisecond (1/1,000 of 1 second)
U	Micro	Microsecond (1/1,000,000 of 1 second)
M	MonthEnd	Last calendar day of month
BM	BusinessMonthEnd	Last business day (weekday) of month
MS	MonthBegin	First calendar day of month
BMS	BusinessMonthBegin	First weekday of month
W-MON, W-TUE, ...	Week	Weekly on given day of week (MON, TUE, WED, THU, FRI, SAT, or SUN)
WOM-1MON, WOM-2MON, ...	WeekOfMonth	Generate weekly dates in the first, second, third, or fourth week of the month (e.g., WOM-3FRI for the third Friday of each month)
Q-JAN, Q-FEB, ...	QuarterEnd	Quarterly dates anchored on last calendar day of each month, for year ending in indicated month (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)
BQ-JAN, BQ-FEB, ...	BusinessQuarterEnd	Quarterly dates anchored on last weekday day of each month, for year ending in indicated month
QS-JAN, QS-FEB, ...	QuarterBegin	Quarterly dates anchored on first calendar day of each month, for year ending in indicated month
BQS-JAN, BQS-FEB, ...	BusinessQuarterBegin	Quarterly dates anchored on first weekday day of each month, for year ending in indicated month
A-JAN, A-FEB, ...	YearEnd	Annual dates anchored on last calendar day of given month (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)
BA-JAN, BA-FEB, ...	BusinessYearEnd	Annual dates anchored on last weekday of given month
AS-JAN, AS-FEB, ...	YearBegin	Annual dates anchored on first day of given month
BAS-JAN, BAS-FEB, ...	BusinessYearBegin	Annual dates anchored on first weekday of given month

```
In [45]: pd.date_range('2012-05-02 12:56:31', periods=5)
Out[45]: DatetimeIndex(['2012-05-02 12:56:31', '2012-05-03 12:56:31',
                       '2012-05-04 12:56:31', '2012-05-05 12:56:31',
                       '2012-05-06 12:56:31'],
                      dtype='datetime64[ns]', freq='D')

In [46]: pd.date_range('2012-05-02 12:56:31', periods=5, normalize=True)
Out[46]: DatetimeIndex(['2012-05-02', '2012-05-03', '2012-05-04', '2012-05-05',
                       '2012-05-06'],
                      dtype='datetime64[ns]', freq='D')
```

- Bazen zaman bilgileriyle birlikte başlangıç veya bitiş tarihleriniz olur ancak bir kural olarak gece yarısına normalleştirilmiş bir dizi timestamps oluşturmak istersiniz. Bunu yapmak için normalize seçeneği vardır.

## Frekanslar ve Tarih Farkları

- Pandalardaki frekanslar bir temel frekans ve bir çarptan oluşan. Temel frekanslar genellikle aylık için 'M' veya saatlik için 'H' gibi bir dize takma adıyla anılır. Her temel frekans için, genel olarak tarih farkı olarak adlandırılan tanımlanmış bir nesne vardır.

```
In [47]: from pandas.tseries.offsets import Hour, Minute
In [48]: hour = Hour()
In [49]: hour
Out[49]: <Hour>
In [50]: four_hours = Hour(4)
In [51]: four_hours
Out[51]: <4 * Hours>
```

- Bir tamsayı ileterek bir uzaklığın katını tanımlayabilirsiniz.
- Çoğu uygulamada, 'H' veya '4H' gibi bir dize takma adı kullanmak yerine bu nesnelerden birini açıkça oluşturmanıza asla gerek kalmaz. Temel frekansın önüne bir tamsayı koymak bir kat oluşturur.

```
In [52]: pd.date_range('2000-01-01', '2000-01-03 23:59', freq='4h')

Out[52]: DatetimeIndex(['2000-01-01 00:00:00', '2000-01-01 04:00:00',
                       '2000-01-01 08:00:00', '2000-01-01 12:00:00',
                       '2000-01-01 16:00:00', '2000-01-01 20:00:00',
                       '2000-01-02 00:00:00', '2000-01-02 04:00:00',
                       '2000-01-02 08:00:00', '2000-01-02 12:00:00',
                       '2000-01-02 16:00:00', '2000-01-02 20:00:00',
                       '2000-01-03 00:00:00', '2000-01-03 04:00:00',
                       '2000-01-03 08:00:00', '2000-01-03 12:00:00',
                       '2000-01-03 16:00:00', '2000-01-03 20:00:00'],
                      dtype='datetime64[ns]', freq='4H')
```

- Birçok ofset eklenecek bir araya getirilebilir.

```
In [53]: Hour(2) + Minute(30)

Out[53]: <150 * Minutes>
```

- Benzer şekilde, aynı ifadeye etkili bir şekilde ayırtılacak '1h30min' gibi frekans dizelerini iletebilirsiniz.

```
In [54]: pd.date_range('2000-01-01', periods=10, freq='1h30min')

Out[54]: DatetimeIndex(['2000-01-01 00:00:00', '2000-01-01 01:30:00',
                       '2000-01-01 03:00:00', '2000-01-01 04:30:00',
                       '2000-01-01 06:00:00', '2000-01-01 07:30:00',
                       '2000-01-01 09:00:00', '2000-01-01 10:30:00',
                       '2000-01-01 12:00:00', '2000-01-01 13:30:00'],
                      dtype='datetime64[ns]', freq='90T')
```

## Ayın Haftası Tarihleri

- Yararlı bir sıklık sınıfı, WOM ile başlayan "week of month"dir. Bu, her ayın üçüncü cuma günü gibi tarihleri almanızı sağlar.

```
In [55]: rng = pd.date_range('2012-01-01', '2012-09-01', freq='WOM-3FRI')

In [56]: list(rng)

Out[56]: [Timestamp('2012-01-20 00:00:00'),
           Timestamp('2012-02-17 00:00:00'),
           Timestamp('2012-03-16 00:00:00'),
           Timestamp('2012-04-20 00:00:00'),
           Timestamp('2012-05-18 00:00:00'),
           Timestamp('2012-06-15 00:00:00'),
           Timestamp('2012-07-20 00:00:00'),
           Timestamp('2012-08-17 00:00:00')]
```

## Verileri Kaydırma (Önde ve Gecikmeli)

- “Kaydırma”, verilerin zaman içinde geriye ve ileriye taşınmasını ifade eder. Hem Series hem de DataFrame, dizini değiştirilmeden bırakarak ileri veya geri basit kaydırma yapmak için bir kaydırma yöntemine sahiptir.

```
In [4]: ts = pd.Series(np.random.randn(4),
                     index=pd.date_range('1/1/2000', periods=4, freq='M'))
```

```
In [5]: ts
```

```
Out[5]: 2000-01-31    -1.025152
2000-02-29     1.381067
2000-03-31     0.191788
2000-04-30    -2.290531
Freq: M, dtype: float64
```

```
In [6]: ts.shift(2)
```

```
Out[6]: 2000-01-31      NaN
2000-02-29      NaN
2000-03-31    -1.025152
2000-04-30     1.381067
Freq: M, dtype: float64
```

- Bu şekilde kayma yaptığımızda, zaman serisinin ya başında ya da sonunda eksik veriler ortaya çıkıyor.
- Basit değişiklikler dizini değiştirilmeden bıraktığı için bazı veriler atılır. Dolayısıyla, eğer frekans biliniyorsa, yalnızca veriler yerine timestamps'i ilerletmek için kaydırma geçilebilir.

```
In [7]: ts.shift(2, freq='M')
```

```
Out[7]: 2000-03-31    -1.025152
2000-04-30     1.381067
2000-05-31     0.191788
2000-06-30    -2.290531
Freq: M, dtype: float64
```

```
In [8]: ts.shift(3, freq='D')
```

```
Out[8]: 2000-02-03    -1.025152
2000-03-03     1.381067
2000-04-03     0.191788
2000-05-03    -2.290531
dtype: float64
```

```
In [9]: ts.shift(1, freq='90T')
```

```
Out[9]: 2000-01-31 01:30:00    -1.025152
2000-02-29 01:30:00     1.381067
2000-03-31 01:30:00     0.191788
2000-04-30 01:30:00    -2.290531
dtype: float64
```

- Buradaki T dakika anlamına gelir.

## Tarihleri Ofsetlerle Kaydırma

- Pandaların tarih farkları aynı zamanda datetime veya Timestamp nesneleriyle de kullanılabilir.

```
In [13]: from pandas.tseries.offsets import Day, MonthEnd
```

```
In [16]: now = datetime(2023, 12, 15)
```

```
In [17]: now + 3 * Day()
```

```
Out[17]: Timestamp('2023-12-18 00:00:00')
```

- MonthEnd gibi bağlılık eklerseniz, ilk artış, sıklık kuralına göre bir tarihi bir sonraki tarihe "ileri alacaktır".

```
In [18]: now + MonthEnd()
```

```
Out[18]: Timestamp('2023-12-31 00:00:00')
```

```
In [19]: now + MonthEnd(2)
```

```
Out[19]: Timestamp('2024-01-31 00:00:00')
```

- Bağlılık uzaklıklar, sırasıyla ileri veya geri alma yöntemlerini kullanarak tarihleri açık bir şekilde ileriye veya geriye doğru "döndürebilir".

```
In [20]: offset = MonthEnd()
```

```
In [21]: offset.rollforward(now)
```

```
Out[21]: Timestamp('2023-12-31 00:00:00')
```

```
In [22]: offset.rollback(now)
```

```
Out[22]: Timestamp('2023-11-30 00:00:00')
```

```
In [23]: ts = pd.Series(np.random.randn(20),
                     index=pd.date_range('1/15/2000', periods=20, freq='4d'))
```

```
In [24]: ts
```

```
Out[24]: 2000-01-15    -1.407450
          2000-01-19    -0.927888
          2000-01-23     0.691302
          2000-01-27     0.023513
          2000-01-31    -1.180537
          2000-02-04     0.036557
          2000-02-08    -1.007959
          2000-02-12     0.539584
          2000-02-16     0.410774
          2000-02-20     0.255703
          2000-02-24     0.818441
          2000-02-28    -0.702599
          2000-03-03    -0.479204
          2000-03-07     2.574249
          2000-03-11    -0.540458
          2000-03-15     2.485668
          2000-03-19    -0.187928
          2000-03-23    -0.403963
          2000-03-27     0.668122
          2000-03-31    -1.263707
          Freq: 4D, dtype: float64
```

```
In [25]: ts.groupby(offset.rollforward).mean()
```

```
Out[25]: 2000-01-31    -0.560212
          2000-02-29     0.050072
          2000-03-31     0.356597
          dtype: float64
```

```
In [26]: ts.resample('M').mean()
```

```
Out[26]: 2000-01-31    -0.560212
          2000-02-29     0.050072
          2000-03-31     0.356597
          Freq: M, dtype: float64
```

## Saat Dilimi İşleme

```
In [27]: import pytz
```

```
In [28]: pytz.common_timezones[-5:]
```

```
Out[28]: ['US/Eastern', 'US/Hawaii', 'US/Mountain', 'US/Pacific', 'UTC']
```

- Pytz'den bir saat dilimi nesnesi almak için pytz.timezone'u kullanın.

```
In [29]: tz = pytz.timezone('America/New_York')
```

```
In [30]: tz
```

```
Out[30]: <DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>
```

## Saat Dilimi Yerelleştirmesi ve Dönüşüm

```
In [31]: rng = pd.date_range('3/9/2012 9:30', periods=6, freq='D')
```

```
In [32]: ts = pd.Series(np.random.randn(len(rng)), index=rng)
```

```
In [33]: ts
```

```
Out[33]: 2012-03-09 09:30:00    -0.608087  
2012-03-10 09:30:00     0.027970  
2012-03-11 09:30:00    -0.748634  
2012-03-12 09:30:00    -0.033192  
2012-03-13 09:30:00    -0.950202  
2012-03-14 09:30:00     0.800325  
Freq: D, dtype: float64
```

- İndeksin tz alanı yoktur.

```
In [34]: print(ts.index.tz)
```

```
None
```

- Tarih aralıkları, bir saat dilimi ayarıyla oluşturulabilir.

```
In [35]: pd.date_range('3/9/2012 9:30', periods=10, freq='D', tz='UTC')
```

```
Out[35]: DatetimeIndex(['2012-03-09 09:30:00+00:00', '2012-03-10 09:30:00+00:00',  
                       '2012-03-11 09:30:00+00:00', '2012-03-12 09:30:00+00:00',  
                       '2012-03-13 09:30:00+00:00', '2012-03-14 09:30:00+00:00',  
                       '2012-03-15 09:30:00+00:00', '2012-03-16 09:30:00+00:00',  
                       '2012-03-17 09:30:00+00:00', '2012-03-18 09:30:00+00:00'],  
                      dtype='datetime64[ns, UTC]', freq='D')
```

- Saftan yerelleştirilmiş dönüşüm tz\_localize yöntemiyle gerçekleştirilir.

```
In [36]: ts
```

```
Out[36]: 2012-03-09 09:30:00    -0.608087  
2012-03-10 09:30:00     0.027970  
2012-03-11 09:30:00    -0.748634  
2012-03-12 09:30:00    -0.033192  
2012-03-13 09:30:00    -0.950202  
2012-03-14 09:30:00     0.800325  
Freq: D, dtype: float64
```

```
In [37]: ts_utc = ts.tz_localize('UTC')
```

```
In [38]: ts_utc
```

```
Out[38]: 2012-03-09 09:30:00+00:00    -0.608087  
2012-03-10 09:30:00+00:00     0.027970  
2012-03-11 09:30:00+00:00    -0.748634  
2012-03-12 09:30:00+00:00    -0.033192  
2012-03-13 09:30:00+00:00    -0.950202  
2012-03-14 09:30:00+00:00     0.800325  
Freq: D, dtype: float64
```

```
In [39]: ts_utc.index  
Out[39]: DatetimeIndex(['2012-03-09 09:30:00+00:00', '2012-03-10 09:30:00+00:00',  
                           '2012-03-11 09:30:00+00:00', '2012-03-12 09:30:00+00:00',  
                           '2012-03-13 09:30:00+00:00', '2012-03-14 09:30:00+00:00'],  
                           dtype='datetime64[ns, UTC]', freq='D')
```

- Bir zaman serisi belirli bir zaman dilimine yerelleştirildiğinde, tz\_convert ile başka bir zaman dilimine dönüştürülebilir.

```
In [40]: ts_utc.tz_convert('America/New_York')  
Out[40]: 2012-03-09 04:30:00-05:00    -0.608087  
          2012-03-10 04:30:00-05:00     0.027970  
          2012-03-11 05:30:00-04:00    -0.748634  
          2012-03-12 05:30:00-04:00    -0.033192  
          2012-03-13 05:30:00-04:00    -0.950202  
          2012-03-14 05:30:00-04:00     0.800325  
          Freq: D, dtype: float64
```

- Amerika/New\_York zaman diliminde bir DST geçişini kapsayan önceki zaman serisi durumunda, EST'ye yerelleştirebilir ve örneğin UTC veya Berlin'e dönüştürebiliriz.

```
In [41]: ts_eastern = ts.tz_localize('America/New_York')
```

```
In [42]: ts_eastern.tz_convert('UTC')
```

```
Out[42]: 2012-03-09 14:30:00+00:00    -0.608087  
          2012-03-10 14:30:00+00:00     0.027970  
          2012-03-11 13:30:00+00:00    -0.748634  
          2012-03-12 13:30:00+00:00    -0.033192  
          2012-03-13 13:30:00+00:00    -0.950202  
          2012-03-14 13:30:00+00:00     0.800325  
          dtype: float64
```

```
In [43]: ts_eastern.tz_convert('Europe/Berlin')
```

```
Out[43]: 2012-03-09 15:30:00+01:00    -0.608087  
          2012-03-10 15:30:00+01:00     0.027970  
          2012-03-11 14:30:00+01:00    -0.748634  
          2012-03-12 14:30:00+01:00    -0.033192  
          2012-03-13 14:30:00+01:00    -0.950202  
          2012-03-14 14:30:00+01:00     0.800325  
          dtype: float64
```

- tz\_localize ve tz\_convert aynı zamanda DatetimeIndex'teki örnek yöntemlerdir.

```
In [44]: ts.index.tz_localize('Asia/Shanghai')

Out[44]: DatetimeIndex(['2012-03-09 09:30:00+08:00', '2012-03-10 09:30:00+08:00',
   '2012-03-11 09:30:00+08:00', '2012-03-12 09:30:00+08:00',
   '2012-03-13 09:30:00+08:00', '2012-03-14 09:30:00+08:00'],
  dtype='datetime64[ns, Asia/Shanghai]', freq=None)
```

## Zaman Dilimi Duyarlı Timestamp Nesneleriyle İşlemler

- Zaman serileri ve tarih aralıklarına benzer şekilde, ayrı timestamp nesneleri de benzer şekilde orijinalden saat dilimine duyarlı olarak yerelleştirilebilir ve bir saat diliminden diğerine dönüştürülebilir.

```
In [45]: stamp = pd.Timestamp('2011-03-12 04:00')

In [46]: stamp_utc = stamp.tz_localize('utc')

In [47]: stamp_utc.tz_convert('America/New_York')

Out[47]: Timestamp('2011-03-11 23:00:00-0500', tz='America/New_York')
```

- Timestamp'i oluştururken bir saat dilimini de iletebilirsiniz.

```
In [48]: stamp_moscow = pd.Timestamp('2011-03-12 04:00', tz='Europe/Moscow')

In [49]: stamp_moscow

Out[49]: Timestamp('2011-03-12 04:00:00+0300', tz='Europe/Moscow')
```

- Zaman dilimi duyarlı timestamp nesneleri, Unix döneminden (1 Ocak 1970) bu yana bir UTC zaman damgası değerini nanosaniye cinsinden dahili olarak saklar; bu UTC değeri saat dilimi dönüşümleri arasında değişmez.

```
In [50]: stamp_utc.value

Out[50]: 12999024000000000000

In [51]: stamp_utc.tz_convert('America/New_York').value

Out[51]: 12999024000000000000
```

```
In [52]: from pandas.tseries.offsets import Hour

In [53]: stamp = pd.Timestamp('2012-03-12 01:30', tz='US/Eastern')
```

```
In [54]: stamp
Out[54]: Timestamp('2012-03-12 01:30:00-0400', tz='US/Eastern')

In [55]: stamp + Hour()
Out[55]: Timestamp('2012-03-12 02:30:00-0400', tz='US/Eastern')
```

- Ardından, DST'den çıkmadan 90 dakika önce.

```
In [56]: stamp = pd.Timestamp('2012-11-04 00:30', tz='US/Eastern')
In [57]: stamp
Out[57]: Timestamp('2012-11-04 00:30:00-0400', tz='US/Eastern')

In [58]: stamp + 2 * Hour()
Out[58]: Timestamp('2012-11-04 01:30:00-0500', tz='US/Eastern')
```

## Farklı Saat Dilimleri Arasındaki İşlemler

- Farklı zaman dilimlerine sahip iki zaman serisi birleştirilirse sonuç UTC olacaktır. Timestampsler UTC'de gizli olarak saklandığından, bu basit bir işlemidir ve herhangi bir dönüşümün gerçekleşmesini gerektirmez.

```
In [59]: rng = pd.date_range('3/7/2012 9:30', periods=10, freq='B')
In [60]: ts = pd.Series(np.random.randn(len(rng)), index=rng)
In [61]: ts
Out[61]:
2012-03-07 09:30:00    -0.322267
2012-03-08 09:30:00    -1.820649
2012-03-09 09:30:00    -0.415552
2012-03-12 09:30:00    -0.466860
2012-03-13 09:30:00     0.290765
2012-03-14 09:30:00     0.546943
2012-03-15 09:30:00     1.600851
2012-03-16 09:30:00    -0.705396
2012-03-19 09:30:00     0.928889
2012-03-20 09:30:00    -1.322869
Freq: B, dtype: float64

In [62]: ts1 = ts[:7].tz_localize('Europe/London')
In [63]: ts2 = ts1[2:].tz_convert('Europe/Moscow')
In [64]: result = ts1 + ts2
In [65]: result.index
Out[65]: DatetimeIndex(['2012-03-07 09:30:00+00:00', '2012-03-08 09:30:00+00:00',
                           '2012-03-09 09:30:00+00:00', '2012-03-12 09:30:00+00:00',
                           '2012-03-13 09:30:00+00:00', '2012-03-14 09:30:00+00:00',
                           '2012-03-15 09:30:00+00:00'],
                           dtype='datetime64[ns, UTC]', freq=None)
```

## Dönemler ve Dönem Aritmetiği

- Dönemler günler, aylar, üç aylık dönemler veya yıllar gibi zaman aralıklarını temsil eder. Periyod sınıfı, bir dize veya tam sayı ve bir frekans gerektiren bu veri türünü temsil eder.

```
In [66]: p = pd.Period('2007', freq='A-DEC')
```

```
In [67]: p
```

```
Out[67]: Period('2007', 'A-DEC')
```

- Bu durumda, Dönem nesnesi 1 Ocak 2007'den 31 Aralık 2007'ye kadar olan tam zaman aralığını temsil eder. Uygun bir şekilde, tamsayıları periyotlara eklemek ve çıkarmak, frekanslarına göre kayma etkisine sahiptir.

```
In [68]: p + 5
```

```
Out[68]: Period('2012', 'A-DEC')
```

```
In [69]: p - 2
```

```
Out[69]: Period('2005', 'A-DEC')
```

- Eğer iki periyot aynı frekansa sahipse aralarındaki fark aralarındaki birim sayısı kadardır.

```
In [70]: pd.Period('2014', freq='A-DEC') - p
```

```
Out[70]: <7 * YearEnds: month=12>
```

```
In [71]: rng = pd.period_range('2000-01-01', '2000-06-30', freq='M')
```

```
In [72]: rng
```

```
Out[72]: PeriodIndex(['2000-01', '2000-02', '2000-03', '2000-04', '2000-05', '2000-06'], dtype='period[M]')
```

- period\_range işleviyle düzenli dönem aralıkları oluşturulabilir.
- PeriodIndex sınıfı bir dönem dizisini saklar.

```
In [73]: pd.Series(np.random.randn(6), index=rng)
```

```
Out[73]: 2000-01    0.446509  
2000-02   -0.672756  
2000-03   -0.563652  
2000-04    1.079574  
2000-05    0.680696  
2000-06    0.684361  
Freq: M, dtype: float64
```

- Bir dize diziniz varsa PeriodIndex sınıfını da kullanabilirsiniz.

```
In [74]: values = ['2001Q3', '2002Q2', '2003Q1']

In [75]: index = pd.PeriodIndex(values, freq='Q-DEC')

In [76]: index

Out[76]: PeriodIndex(['2001Q3', '2002Q2', '2003Q1'], dtype='period[Q-DEC]')
```

## Dönem Frekans Dönüşümü

- Periods ve PeriodIndex nesneleri asfreq yöntemiyle başka bir frekansa dönüştürülebilir. Örnek olarak, yıllık bir dönemimiz olduğunu ve bunu yılın başında veya sonunda aylık bir döneme dönüştürmek istediğimizi varsayalım.

```
In [77]: p = pd.Period('2007', freq='A-DEC')

In [78]: p

Out[78]: Period('2007', 'A-DEC')

In [79]: p.asfreq('M', how='start')

Out[79]: Period('2007-01', 'M')

In [80]: p.asfreq('M', how='end')

Out[80]: Period('2007-12', 'M')
```

- Period('2007', 'ADEC') öğesini, aylık dönemlere bölünmüş bir zaman aralığını işaret eden bir tür imleç olarak düşünebilirsiniz.

```
In [81]: p = pd.Period('2007', freq='A-JUN')

In [82]: p

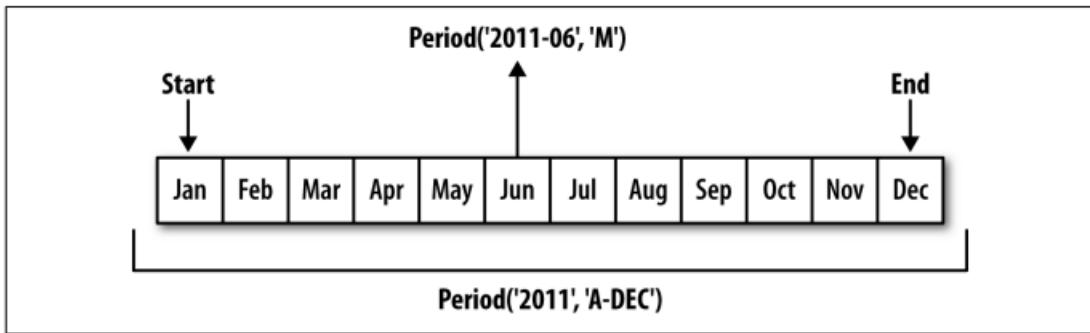
Out[82]: Period('2007', 'A-JUN')

In [83]: p.asfreq('M', 'start')

Out[83]: Period('2006-07', 'M')

In [84]: p.asfreq('M', 'end')

Out[84]: Period('2007-06', 'M')
```



dönem frekans dönüşüm illüstrasyonu

- Yüksek frekanstan düşük frekansa geçiş yaptığınızda pandalar, alt periyodun “ait olduğu yere” bağlı olarak süper periyodu belirler. Örneğin, A-HAZİRAN frekansında Ağustos-2007 ayı aslında 2008 döneminin bir parçasıdır.

```
In [85]: p = pd.Period('Aug-2007', 'M')
```

```
In [86]: p.asfreq('A-JUN')
```

```
Out[86]: Period('2008', 'A-JUN')
```

- Burada yıllık dönemler, her yıllık dönem içinde yer alan ilk aya karşılık gelen aylık dönemlerle değiştirilmektedir. Bunun yerine her yılın son iş gününü istiyorsak 'B' sıklığını kullanabilir ve yılın sonunu istediğimizi belirtebiliriz.

```
In [87]: ts.asfreq('B', how='end')
```

```
Out[87]: 2012-03-07 09:30:00    -0.322267
2012-03-08 09:30:00    -1.820649
2012-03-09 09:30:00    -0.415552
2012-03-12 09:30:00    -0.466860
2012-03-13 09:30:00    0.290765
2012-03-14 09:30:00    0.546943
2012-03-15 09:30:00    1.600851
2012-03-16 09:30:00    -0.705396
2012-03-19 09:30:00    0.928889
2012-03-20 09:30:00    -1.322869
Freq: B, dtype: float64
```

## Üç Aylık Dönem Frekansları

- Üç aylık veriler muhasebe, finans ve diğer alanlarda standarttır. Üç aylık verilerin çoğu, genellikle yılın 12 ayından birinin son takvimi veya iş günü olan mali yıl sonuna göre raporlanır. Dolayısıyla 2012Ç4 dönemi, mali yılsonuna göre farklı bir anlam taşımaktadır. pandalar, Q-JAN'dan Q-DEC'e kadar olası 12 üç aylık frekansın tümünü destekler.

```
In [6]: p = pd.Period('2012Q4', freq='Q-JAN')  
  
In [7]: p  
  
Out[7]: Period('2012Q4', 'Q-JAN')
```

- Mali yılın ocak ayında sona ermesi durumunda, 2012Q4 Kasım'dan Ocak'a kadar sürer ve bunu günlük sıklığa dönüştürerek kontrol edebilirsiniz.

Year 2012												
M	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
Q-DEC	2012Q1		2012Q2		2012Q3		2012Q4					
Q-SEP	2012Q2		2012Q3		2012Q4		2013Q1					
Q-FEB	2012Q4		2013Q1		2013Q2		2013Q3		Q4			

farklı üç aylık sıklık kuralları

```
In [8]: p.asfreq('D', 'start')  
  
Out[8]: Period('2011-11-01', 'D')  
  
In [9]: p.asfreq('D', 'end')  
  
Out[9]: Period('2012-01-31', 'D')
```

- Böylece kolay dönem aritmetiği yapmak mümkündür; örneğin, timestamp'i çeyreğin ikinciden son iş günü saat 16.00'ya almak için şunları yapabilirsiniz:

```
In [10]: p4pm = (p.asfreq('B', 'e') - 1).asfreq('T', 's') + 16 * 60  
  
In [11]: p4pm  
  
Out[11]: Period('2012-01-30 16:00', 'T')  
  
In [12]: p4pm.to_timestamp()  
  
Out[12]: Timestamp('2012-01-30 16:00:00')
```

- period\_range'ı kullanarak üç aylık aralıklar oluşturabilirsiniz.

```
In [13]: rng = pd.period_range('2011Q3', '2012Q4', freq='Q-JAN')

In [14]: ts = pd.Series(np.arange(len(rng)), index=rng)

In [15]: ts

Out[15]: 2011Q3    0
          2011Q4    1
          2012Q1    2
          2012Q2    3
          2012Q3    4
          2012Q4    5
Freq: Q-JAN, dtype: int32

In [16]: new_rng = (rng.asfreq('B', 'e') - 1).asfreq('T', 's') + 16 * 60

In [17]: ts.index = new_rng.to_timestamp()

In [18]: ts

Out[18]: 2010-10-28 16:00:00    0
          2011-01-28 16:00:00    1
          2011-04-28 16:00:00    2
          2011-07-28 16:00:00    3
          2011-10-28 16:00:00    4
          2012-01-30 16:00:00    5
dtype: int32
```

## Timestampleri Dönemlere (ve Geriye) Dönüştürme

```
In [19]: rng = pd.date_range('2000-01-01', periods=3, freq='M')

In [20]: ts = pd.Series(np.random.randn(3), index=rng)

In [21]: ts

Out[21]: 2000-01-31    0.054984
          2000-02-29    0.686255
          2000-03-31    0.018817
Freq: M, dtype: float64

In [22]: pts = ts.to_period()

In [23]: pts

Out[23]: 2000-01    0.054984
          2000-02    0.686255
          2000-03    0.018817
Freq: M, dtype: float64
```

- Dönemler örtüşmeyen zaman aralıklarını ifade ettiğinden, bir timestamp yalnızca belirli bir zamana ait olabilir. Belirli bir frekans için tek bir periyot. Yeni PeriodIndex'in sıklığı varsayılan olarak timestamp'ten çıkarılsa da istediğiniz sıklığı belirtebilirsiniz. Sonuçta yinelenen dönemlerin bulunmasında da bir sorun yoktur.

```
In [24]: rng = pd.date_range('1/29/2000', periods=6, freq='D')
```

```
In [25]: ts2 = pd.Series(np.random.randn(6), index=rng)
```

```
In [26]: ts2
```

```
Out[26]: 2000-01-29    2.459577
          2000-01-30    0.867810
          2000-01-31   -0.495219
          2000-02-01    0.637518
          2000-02-02    0.622589
          2000-02-03   -0.498953
Freq: D, dtype: float64
```

```
In [27]: ts2.to_period('M')
```

```
Out[27]: 2000-01    2.459577
          2000-01    0.867810
          2000-01   -0.495219
          2000-02    0.637518
          2000-02    0.622589
          2000-02   -0.498953
Freq: M, dtype: float64
```

- Burada ay sonu periyotlarını verir.
- Timestamp'e geri dönmek için to\_timestamp komutunu kullanın.

```
In [30]: pts = ts2.to_period()
```

```
In [31]: pts
```

```
Out[31]: 2000-01-29    2.459577
          2000-01-30    0.867810
          2000-01-31   -0.495219
          2000-02-01    0.637518
          2000-02-02    0.622589
          2000-02-03   -0.498953
Freq: D, dtype: float64
```

```
In [29]: pts.to_timestamp(how='end')
```

```
Out[29]: 2000-01-29 23:59:59.999999999    2.459577
          2000-01-30 23:59:59.999999999    0.867810
          2000-01-31 23:59:59.999999999   -0.495219
          2000-02-01 23:59:59.999999999    0.637518
          2000-02-02 23:59:59.999999999    0.622589
          2000-02-03 23:59:59.999999999   -0.498953
Freq: D, dtype: float64
```

## Dizilerden PeriodIndex Oluşturma

- Sabit frekanslı veri kümeleri bazen birden fazla sütuna yayılmış zaman aralığı bilgileri ile depolanır. Örneğin, bu makroekonomik veri setinde yıl ve çeyrek farklı sütunlardır:

```
In [32]: data = pd.read_csv('macrodata.csv')

In [33]: data.head(5)

Out[33]:
   year  quarter  realgdp  realcons  realinv  realgovt  realdpi  cpi  m1  tbilrate  unemp  pop  infl  realint
0  1959.0      1.0  2710.349  1707.4  286.898  470.045  1886.9  28.98  139.7  2.82  5.8  177.146  0.00  0.00
1  1959.0      2.0  2778.801  1733.7  310.859  481.301  1919.7  29.15  141.7  3.08  5.1  177.830  2.34  0.74
2  1959.0      3.0  2775.488  1751.8  289.226  491.260  1916.4  29.35  140.5  3.82  5.3  178.657  2.74  1.09
3  1959.0      4.0  2785.204  1753.7  299.356  484.052  1931.3  29.37  140.0  4.33  5.6  179.386  0.27  4.06
4  1960.0      1.0  2847.699  1770.5  331.722  462.199  1955.5  29.54  139.6  3.50  5.2  180.007  2.31  1.19
```

```
In [34]: data.year

Out[34]:
0    1959.0
1    1959.0
2    1959.0
3    1959.0
4    1960.0
...
198   2008.0
199   2008.0
200   2009.0
201   2009.0
202   2009.0
Name: year, Length: 203, dtype: float64
```

```
In [35]: data.quarter

Out[35]:
0    1.0
1    2.0
2    3.0
3    4.0
4    1.0
...
198   3.0
199   4.0
200   1.0
201   2.0
202   3.0
Name: quarter, Length: 203, dtype: float64
```

- Bu dizileri PeriodIndex'e bir frekansla ileterek, bunları DataFrame için bir dizin oluşturmak üzere birllestirebilirsiniz.

```
In [36]: index = pd.PeriodIndex(year=data.year, quarter=data.quarter,
                           freq='Q-DEC')

In [37]: index

Out[37]: PeriodIndex(['1959Q1', '1959Q2', '1959Q3', '1959Q4', '1960Q1', '1960Q2',
                     '1960Q3', '1960Q4', '1961Q1', '1961Q2',
                     ...
                     '2007Q2', '2007Q3', '2007Q4', '2008Q1', '2008Q2', '2008Q3',
                     '2008Q4', '2009Q1', '2009Q2', '2009Q3'],
                           dtype='period[Q-DEC]', length=203)
```

```
In [38]: data.index = index
```

```
In [39]: data.infl
```

```
Out[39]: 1959Q1    0.00
1959Q2    2.34
1959Q3    2.74
1959Q4    0.27
1960Q1    2.31
...
2008Q3   -3.16
2008Q4   -8.79
2009Q1    0.94
2009Q2    3.37
2009Q3    3.56
Freq: Q-DEC, Name: infl, Length: 203, dtype: float64
```

## Yeniden Örnekleme ve Frekans Dönüşümü

- Yeniden örneklemeye, bir zaman serisinin bir frekanstan diğerine dönüştürülmesi işlemini ifade eder. Yüksek frekanslı verileri daha düşük frekansa toplamaya alt örneklemeye, düşük frekansı daha yüksek frekansa dönüştürmeye ise üst örneklemeye adı verilir. Yeniden örneklemenin tamamı bu kategorilerden herhangi birine gitmez; örneğin, WWED'i (haftalık çarşamba günü) W-FRI'ye dönüştürmek ne yukarı örneklemeye ne de alt örneklemeyidir.
- pandasın nesneleri, tüm frekans dönüşümleri için en güçlü işlev olan yeniden örneklemeye yöntemiyle donatılmıştır. resample'in groupby'ye benzer bir API'si vardır; verileri gruplandırmak için yeniden örneklemeyi çağırırsınız, ardından bir toplama işlevini çağırırsınız.

```
In [40]: rng = pd.date_range('2000-01-01', periods=100, freq='D')
```

```
In [41]: ts = pd.Series(np.random.randn(len(rng)), index=rng)
```

```
In [42]: ts
```

```
Out[42]: 2000-01-01    -0.047889
2000-01-02     1.074117
2000-01-03    -2.934927
2000-01-04    -1.107591
2000-01-05    -1.469198
...
2000-04-05     0.552689
2000-04-06    -0.513269
2000-04-07     0.852428
2000-04-08    -1.019905
2000-04-09     1.276577
Freq: D, Length: 100, dtype: float64
```

```
In [43]: ts.resample('M').mean()
```

```
Out[43]: 2000-01-31    -0.182394  
2000-02-29     0.250603  
2000-03-31    -0.086329  
2000-04-30     0.341524  
Freq: M, dtype: float64
```

```
In [44]: ts.resample('M', kind='period').mean()
```

```
Out[44]: 2000-01    -0.182394  
2000-02     0.250603  
2000-03    -0.086329  
2000-04     0.341524  
Freq: M, dtype: float64
```

## Resample Yöntem Argümanları

Argument	Description
freq	String or DateOffset indicating desired resampled frequency (e.g., 'M', '5min', or Second(15))
axis	Axis to resample on; default axis=0
fill_method	How to interpolate when upsampling, as in 'ffill' or 'bfill'; by default does no interpolation
closed	In downsampling, which end of each interval is closed (inclusive), 'right' or 'left'
label	In downsampling, how to label the aggregated result, with the 'right' or 'left' bin edge (e.g., the 9:30 to 9:35 five-minute interval could be labeled 9:30 or 9:35)
loffset	Time adjustment to the bin labels, such as '-1s' / Second(-1) to shift the aggregate labels one second earlier
limit	When forward or backward filling, the maximum number of periods to fill
kind	Aggregate to periods ('period') or timestamps ('timestamp'); defaults to the type of index the time series has
convention	When resampling periods, the convention ('start' or 'end') for converting the low-frequency period to high frequency; defaults to 'end'

## Altörnekleme

Verileri düzenli, daha düşük bir frekansta toplamak oldukça normal bir zaman serisi görevidir. Topladığınız verilerin sık sık düzeltmesine gerek yoktur; istenilen frekans zaman serisini toplanacak parçalara bölmek için kullanılan bin kenarlarını tanımlar. Örneğin, aylık 'M' veya 'BM'ye dönüştürmek için verileri birer aylık aralıklarla bölmek gereklidir. Her aralığın yarı açık olduğu söylenir; bir veri noktası yalnızca bir aralığa ait olabilir ve aralıkların birleşimi tüm zaman dilimini oluşturmmalıdır. Verileri alt örnekleme için yeniden örnekleme kullanırken dikkate alınması gereken birkaç nokta vardır:

- Her aralığın hangi tarafı kapalıdır
- Her birleştirilmiş bölmenin aralığın başlangıcı veya sonu ile nasıl etiketleneceği

```
In [45]: rng = pd.date_range('2000-01-01', periods=12, freq='T')
```

```
In [46]: ts = pd.Series(np.arange(12), index=rng)
```

```
In [47]: ts
```

```
Out[47]: 2000-01-01 00:00:00    0  
2000-01-01 00:01:00    1  
2000-01-01 00:02:00    2  
2000-01-01 00:03:00    3  
2000-01-01 00:04:00    4  
2000-01-01 00:05:00    5  
2000-01-01 00:06:00    6  
2000-01-01 00:07:00    7  
2000-01-01 00:08:00    8  
2000-01-01 00:09:00    9  
2000-01-01 00:10:00   10  
2000-01-01 00:11:00   11  
Freq: T, dtype: int32
```

- Bir dakikalık periyotlar.
- Her grubun toplamını alarak bu verileri beş dakikalık parçalar veya çubuklar halinde toplamak istediğiniz varsayıalım.

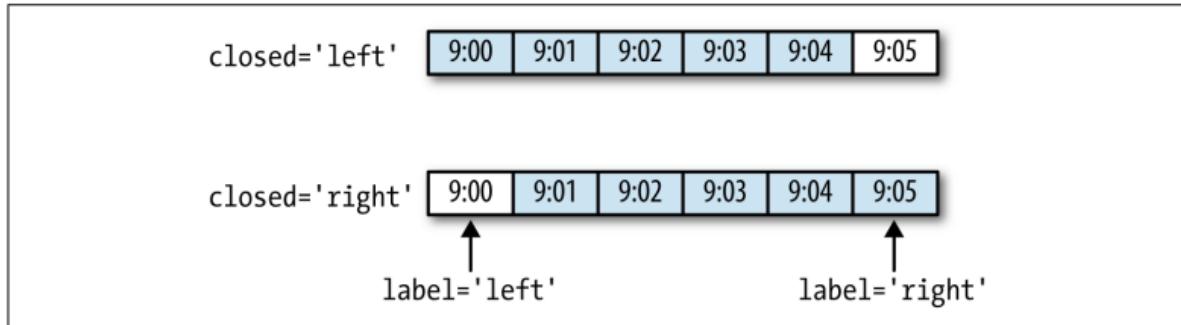
```
In [48]: ts.resample('5min', closed='right').sum()
```

```
Out[48]: 1999-12-31 23:55:00    0  
2000-01-01 00:00:00    15  
2000-01-01 00:05:00    40  
2000-01-01 00:10:00   11  
Freq: 5T, dtype: int32
```

- Aktardığınız sıklık, bölme kenarlarını beş dakikalık artışlarla tanımlar. Varsayılan olarak sol bölge kenarı dahildir, dolayısıyla 00:00 değeri 00:00 ila 00:05 aralığına dahil edilir. Bir Closed='right' değerini geçmek sağda kapatılacak aralığı değiştirir.
- Ortaya çıkan zaman serisi, her bölmenin sol tarafındaki timestampler ile etiketlenir. label='right' komutunu ileterek bunları sağ bölge kenarıyla etiketleyebilirsiniz.

```
In [49]: ts.resample('5min', closed='right', label='right').sum()
```

```
Out[49]: 2000-01-01 00:00:00    0  
2000-01-01 00:05:00    15  
2000-01-01 00:10:00    40  
2000-01-01 00:15:00   11  
Freq: 5T, dtype: int32
```



kapalı etiket kurallarının beş dakikalık yeniden örnekleme çizimi

Son olarak, zaman damgasının hangi aralığa atıfta bulunduğu daha net hale getirmek için sonuç dizinini bir miktar kaydırın, örneğin sağ kenardan bir saniye çıkarmak isteyebilirsiniz. Bunu yapmak için `loffset`'e bir dize veya tarih farkı iletin.

```
In [52]: ts.resample('5min', closed='right',
                  label='right', offset='-1s').sum()
```

## Açık-Yüksek-Düşük-Kapanış (OHLC) Yeniden Örnekleme

Finansta, bir zaman serisini toplamanın popüler bir yolu, her biri için dört değer hesaplamaktır. paket: ilk (açık), son (kapanış), maksimum (yüksek) ve minimum (düşük) değerler. Ohlc toplama işlevini kullanarak, verilerin tek bir taramasında verimli bir şekilde hesaplanan bu dört toplamayı içeren sütunlara sahip bir DataFrame elde edekeksiniz.

```
In [53]: ts.resample('5min').ohlc()
```

Out[53]:

	open	high	low	close
2000-01-01 00:00:00	0	4	0	4
2000-01-01 00:05:00	5	9	5	9
2000-01-01 00:10:00	10	11	10	11

## Örnekleme ve Enterpolasyon

- Düşük frekanstan yüksek frekansa geçiş yaparken toplamaya gerek yoktur. Bazı haftalık verileri içeren bir DataFrame'i ele alalım:

```
In [54]: frame = pd.DataFrame(np.random.randn(2, 4),
                           index=pd.date_range('1/1/2000', periods=2,
                           freq='W-WED'),
                           columns=['Colorado', 'Texas', 'New York', 'Ohio'])
```

```
In [55]: frame
```

```
Out[55]:
```

	Colorado	Texas	New York	Ohio
2000-01-05	-1.571668	-0.928343	-0.346228	-0.406763
2000-01-12	-0.017172	0.969793	0.129631	-0.052012

- Bu verilerle bir toplama işlevi kullandığınızda grup başına yalnızca bir değer vardır ve eksik değerler boşluklara neden olur. Herhangi bir toplama olmadan daha yüksek frekansa dönüştürmek için asfreq yöntemini kullanıyoruz.

```
In [56]: df_daily = frame.resample('D').asfreq()
```

```
In [57]: df_daily
```

```
Out[57]:
```

	Colorado	Texas	New York	Ohio
2000-01-05	-1.571668	-0.928343	-0.346228	-0.406763
2000-01-06	NaN	NaN	NaN	NaN
2000-01-07	NaN	NaN	NaN	NaN
2000-01-08	NaN	NaN	NaN	NaN
2000-01-09	NaN	NaN	NaN	NaN
2000-01-10	NaN	NaN	NaN	NaN
2000-01-11	NaN	NaN	NaN	NaN
2000-01-12	-0.017172	0.969793	0.129631	-0.052012

- Her haftalık değeri Çarşamba dışındaki günlerde doldurmak istediğiniz varsayılm. Fillna ve reindex yöntemlerinde aynı doldurma veya enterpolasyon yöntemleri mevcuttur.

```
In [58]: frame.resample('D').ffill()
```

```
Out[58]:
```

	Colorado	Texas	New York	Ohio
2000-01-05	-1.571668	-0.928343	-0.346228	-0.406763
2000-01-06	-1.571668	-0.928343	-0.346228	-0.406763
2000-01-07	-1.571668	-0.928343	-0.346228	-0.406763
2000-01-08	-1.571668	-0.928343	-0.346228	-0.406763
2000-01-09	-1.571668	-0.928343	-0.346228	-0.406763
2000-01-10	-1.571668	-0.928343	-0.346228	-0.406763
2000-01-11	-1.571668	-0.928343	-0.346228	-0.406763
2000-01-12	-0.017172	0.969793	0.129631	-0.052012

- Benzer şekilde, gözlemlenen bir değeri kullanmaya ne kadar devam edeceğini sınırılamak için yalnızca belirli sayıda dönemi doldurmayı seçebilirsiniz.

```
In [59]: frame.resample('D').ffill(limit=2)
```

Out[59]:

	Colorado	Texas	New York	Ohio
2000-01-05	-1.571668	-0.928343	-0.346228	-0.406763
2000-01-06	-1.571668	-0.928343	-0.346228	-0.406763
2000-01-07	-1.571668	-0.928343	-0.346228	-0.406763
2000-01-08	NaN	NaN	NaN	NaN
2000-01-09	NaN	NaN	NaN	NaN
2000-01-10	NaN	NaN	NaN	NaN
2000-01-11	NaN	NaN	NaN	NaN
2000-01-12	-0.017172	0.969793	0.129631	-0.052012

## Dönemlerle Yeniden Örnekleme

- Dönemlere göre indekslenen verileri yeniden örneklemek timestampler'e benzer.

```
In [60]: frame = pd.DataFrame(np.random.randn(24, 4),
                           index=pd.period_range('1-2000', '12-2001',
                           freq='M'),
                           columns=['Colorado', 'Texas', 'New York', 'Ohio'])
```

```
In [61]: frame[:5]
```

Out[61]:

	Colorado	Texas	New York	Ohio
2000-01	-0.128473	-0.512441	-0.653036	-0.596222
2000-02	2.529926	-1.742397	0.853715	-0.794932
2000-03	0.727674	-1.667102	-0.019327	1.292522
2000-04	0.063650	0.787204	0.981168	-0.022600
2000-05	0.127667	1.741935	0.036204	-0.945992

```
In [62]: annual_frame = frame.resample('A-DEC').mean()
```

```
In [63]: annual_frame
```

Out[63]:

	Colorado	Texas	New York	Ohio
2000	0.520066	-0.279507	0.092027	-0.100978
2001	-0.127082	-0.281556	-0.028576	0.549539

- Yukarı örnekleme daha incelikli bir iştir; tıpkı asfreq yönteminde olduğu gibi, yeniden örneklemeden önce değerleri yeni frekansta zaman aralığının hangi ucuna yerlestireceğinize karar vermeniz gereklidir. Kural argümanı varsayılan olarak 'başlangıç'tır ancak 'bitiş' de olabilir.

```
In [64]: annual_frame.resample('Q-DEC').ffill()
```

Out[64]:

	Colorado	Texas	New York	Ohio
2000Q1	0.520066	-0.279507	0.092027	-0.100978
2000Q2	0.520066	-0.279507	0.092027	-0.100978
2000Q3	0.520066	-0.279507	0.092027	-0.100978
2000Q4	0.520066	-0.279507	0.092027	-0.100978
2001Q1	-0.127082	-0.281556	-0.028576	0.549539
2001Q2	-0.127082	-0.281556	-0.028576	0.549539
2001Q3	-0.127082	-0.281556	-0.028576	0.549539
2001Q4	-0.127082	-0.281556	-0.028576	0.549539

```
In [65]: annual_frame.resample('Q-DEC', convention='end').ffill()
```

Out[65]:

	Colorado	Texas	New York	Ohio
2000Q4	0.520066	-0.279507	0.092027	-0.100978
2001Q1	0.520066	-0.279507	0.092027	-0.100978
2001Q2	0.520066	-0.279507	0.092027	-0.100978
2001Q3	0.520066	-0.279507	0.092027	-0.100978
2001Q4	-0.127082	-0.281556	-0.028576	0.549539

Dönemler zaman aralıklarını ifade ettiğinden, üst örnekleme ve alt örneklemeyle ilgili kurallar daha katıdır:

- Alt örneklemede hedef frekans, kaynak frekansın bir alt periyodu olmalıdır.
- Yukarı örneklemede hedef frekans, kaynak frekansının bir üst periyodu olmalıdır.
- Bu kurallara uyulmaması durumunda bir istisna gündeme gelecektir. Bu esas olarak üç aylık, yıllık ve haftalık frekansları etkiler; örneğin, QMAR tarafından tanımlanan zaman aralıkları yalnızca A-MAR, A-JUN, A-SEP ve A-DEC ile uyumludur.

```
In [66]: annual_frame.resample('Q-MAR').ffill()
```

Out[66]:

	Colorado	Texas	New York	Ohio
2000Q4	0.520066	-0.279507	0.092027	-0.100978
2001Q1	0.520066	-0.279507	0.092027	-0.100978
2001Q2	0.520066	-0.279507	0.092027	-0.100978
2001Q3	0.520066	-0.279507	0.092027	-0.100978
2001Q4	-0.127082	-0.281556	-0.028576	0.549539
2002Q1	-0.127082	-0.281556	-0.028576	0.549539
2002Q2	-0.127082	-0.281556	-0.028576	0.549539
2002Q3	-0.127082	-0.281556	-0.028576	0.549539

## Hareketli Pencere İşlevleri

Zaman serisi işlemleri için kullanılan dizi dönüşümlerinin önemli bir sınıfı, kayan bir pencere üzerinde veya üstel olarak azalan ağırlıklarla değerlendirilen istatistikler ve diğer işlevlerdir. Bu, gürültülü veya boşluklu verileri düzeltmek için yararlı olabilir. Üstel ağırlıklı hareketli ortalama gibi sabit uzunlukta bir pencereye sahip olmayan işlevler içermesine rağmen, bunlara hareketli pencere işlevleri diyorum. Diğer istatistiksel işlevler gibi bunlar da eksik verileri otomatik olarak hariç tutar.

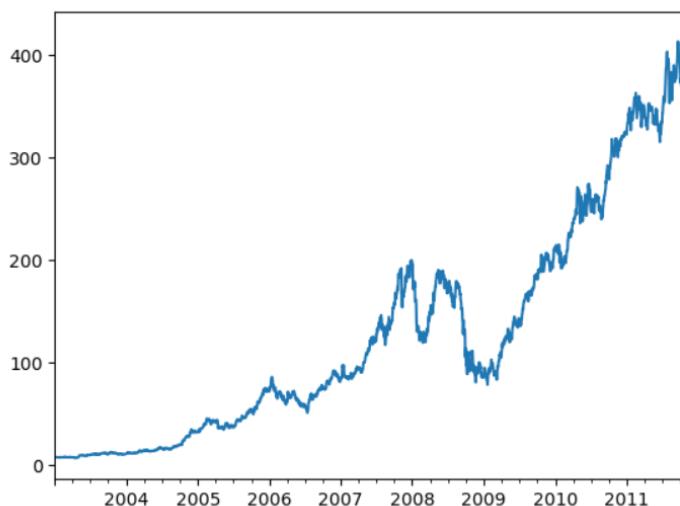
```
In [73]: close_px_all = pd.read_csv('stock_px_2.csv',
                                 parse_dates=True, index_col=0)
```

```
In [74]: close_px = close_px_all[['AAPL', 'MSFT', 'XOM']]
```

```
In [75]: close_px = close_px.resample('B').ffill()
```

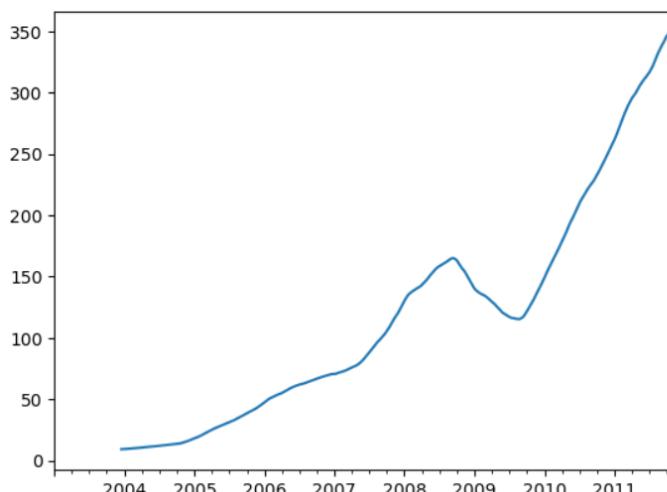
```
In [76]: close_px.AAPL.plot()
```

```
Out[76]: <Axes: >
```



```
In [77]: close_px.AAPL.rolling(250).mean().plot()
```

```
Out[77]: <Axes: >
```



- Varsayılan olarak yuvarlama işlevleri, penceredeki tüm değerlerin NA olmamasını gerektirir. Bu davranış, eksik verileri ve özellikle de gerçeği hesaba katacak şekilde değiştirebilir.

```
In [78]: appl_std250 = close_px.AAPL.rolling(250, min_periods=10).std()
```

```
In [79]: appl_std250[5:12]
```

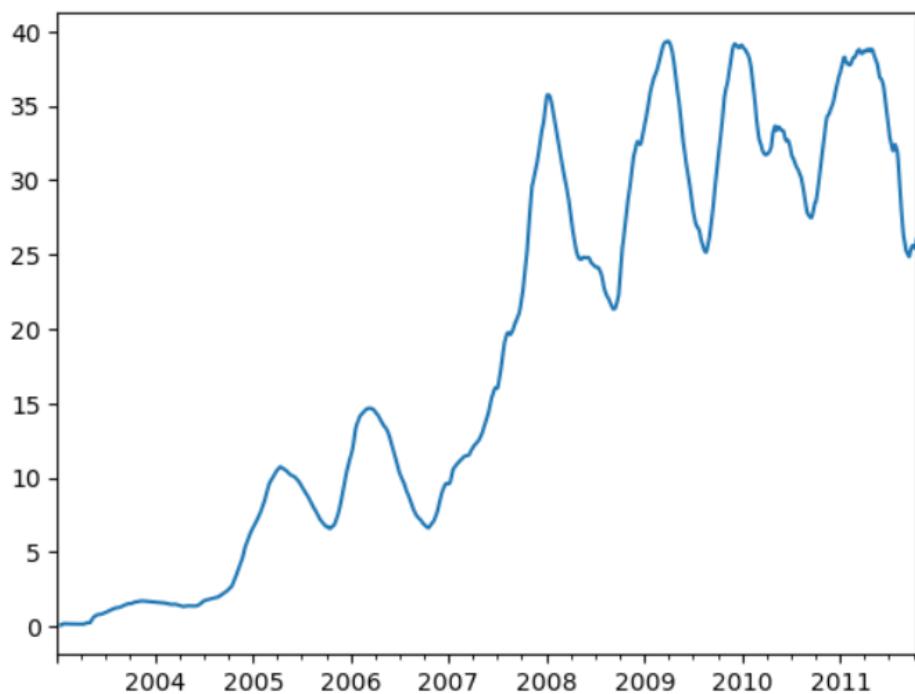
```
Out[79]:
```

2003-01-09	NaN
2003-01-10	NaN
2003-01-13	NaN
2003-01-14	NaN
2003-01-15	0.077496
2003-01-16	0.074760
2003-01-17	0.112368

Freq: B, Name: AAPL, dtype: float64

```
In [80]: appl_std250.plot()
```

```
Out[80]: <Axes: >
```

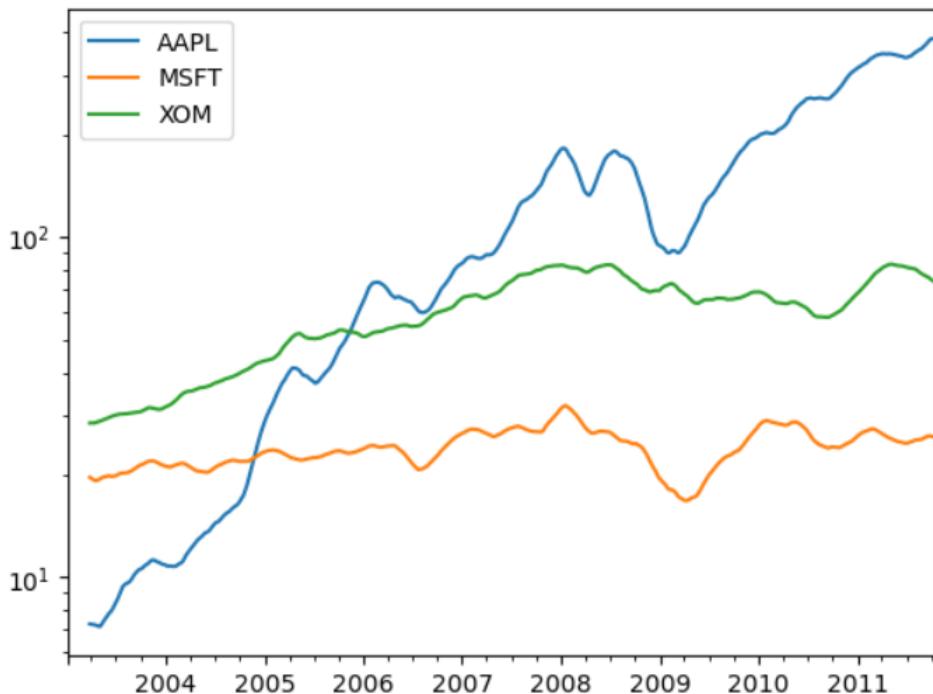


- Genişleyen pencere ortalamasını hesaplamak için yuvarlama yerine genişletme operatörünü kullanın. Genişleyen ortalama, zaman penceresini zaman serisini genişletir ve tüm seriyi kapsayana kadar pencerenin boyutunu artırır.

```
In [81]: expanding_mean = appl_std250.expanding().mean()
```

```
In [82]: close_px.rolling(60).mean().plot(logy=True)
```

```
Out[82]: <Axes: >
```



- Yuvarlama işlevi aynı zamanda belirli sayıda dönem yerine sabit boyutlu bir zaman farkını belirten bir dizeyi de kabul eder. Bu gösterimin kullanılması düzensiz zaman serileri için yararlı olabilir. Bunlar, yeniden örneklemeye iletebileceğiniz dizelerin aynısıdır. Örneğin, 20 günlük hareketli ortalamayı şu şekilde hesaplayabiliriz:

```
In [83]: close_px.rolling('20D').mean()
```

```
Out[83]:
```

	AAPL	MSFT	XOM
2003-01-02	7.400000	21.110000	29.220000
2003-01-03	7.425000	21.125000	29.230000
2003-01-06	7.433333	21.256667	29.473333
2003-01-07	7.432500	21.425000	29.342500
2003-01-08	7.402000	21.402000	29.240000
...	...	...	...
2011-10-10	389.351429	25.602143	72.527857
2011-10-11	388.505000	25.674286	72.835000
2011-10-12	388.531429	25.810000	73.400714
2011-10-13	388.826429	25.961429	73.905000
2011-10-14	391.038000	26.048667	74.185333

2292 rows × 3 columns

## Üstel Ağırlıklı Fonksiyonlar

Eşit ağırlıklı gözlemlerle statik bir pencere boyutu kullanmanın bir alternatif, daha yeni gözlemlere daha fazla ağırlık vermek için sabit bir azalma faktörü belirlemektir. Bozunma faktörünü belirlemenin birkaç yolu vardır. Popüler olanı, sonucu, pencere boyutu açılığa eşit olan basit bir hareketli pencere işleviyle karşılaştırılabilir hale getiren bir yayılma alanı kullanmaktadır.

Üstel ağırlıklı istatistik, daha yeni gözlemlere daha fazla ağırlık verdiginden, eşit ağırlıklı versiyona kıyasla değişikliklere daha hızlı "adapte olur".

- pandaların yuvarlanması ve genişleme işlemlerini gerçekleştirecek ewm operatörü vardır. İşte karşılaştırılan bir örnek: Span=60 olan EW hareketli ortalaması ile Apple hisse senedi fiyatının 60 günlük hareketli ortalaması:

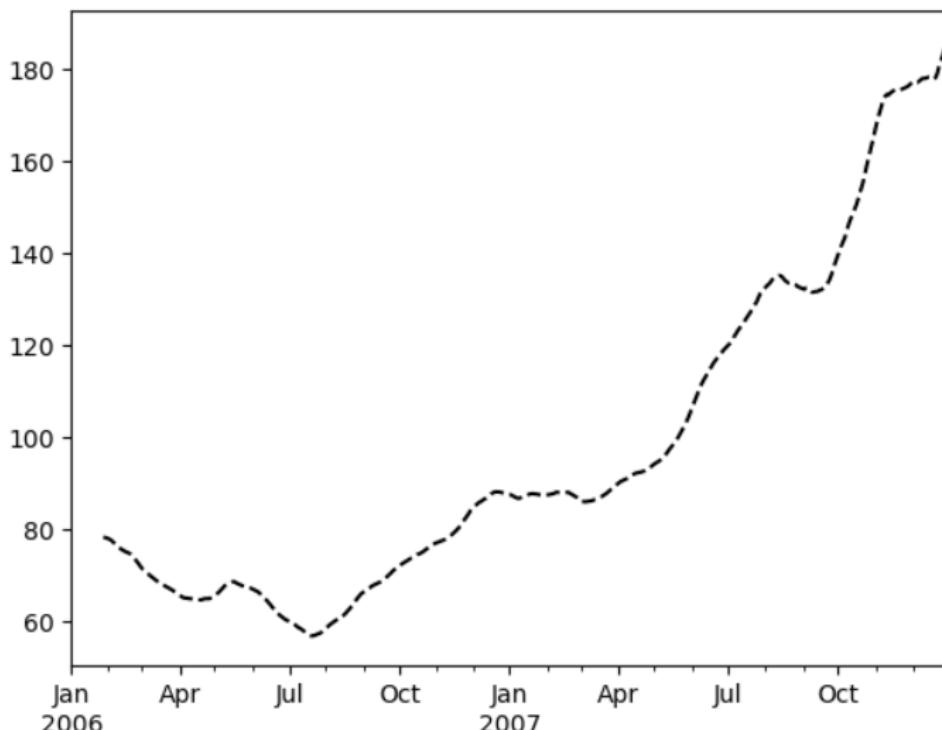
```
In [86]: aapl_px = close_px.AAPL['2006':'2007']
```

```
In [87]: ma60 = aapl_px.rolling(30, min_periods=20).mean()
```

```
In [88]: ewma60 = aapl_px.ewm(span=30).mean()
```

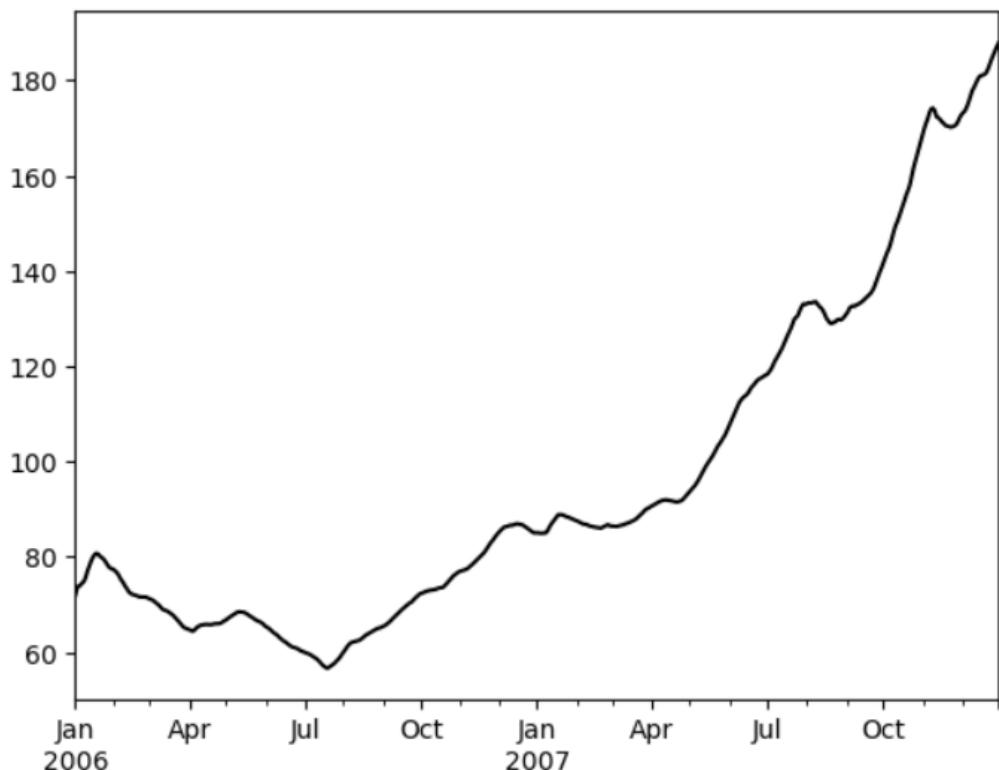
```
In [94]: ma60.plot(style='k--', label='Simple MA')
```

```
Out[94]: <Axes: >
```



```
In [95]: ewma60.plot(style='k-', label='EW MA')
```

```
Out[95]: <Axes: >
```



## İkili Hareketli Pencere İşlevleri

- Korelasyon ve kovaryans gibi bazı istatistiksel operatörlerin iki zaman serisi üzerinde çalışması gereklidir. Örnek olarak, finansal analistler genellikle bir hisse senedinin korelasyonunuyla ilgilenirler. S&P 500 gibi bir kıyaslama endeksi için. Buna bakmak için öncelikle ilgilendiğimiz tüm zaman serileri için yüzde değişimi hesaplıyoruz:

```
In [99]: spx_px = close_px_all['SPX']
```

```
In [100]: spx_rets = spx_px.pct_change()
```

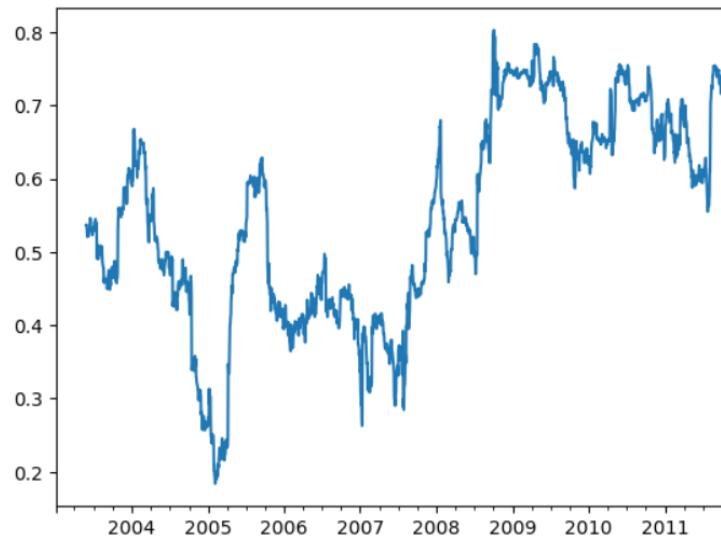
```
In [101]: returns = close_px.pct_change()
```

- Rolling'i çağrıdıktan sonra düzeltme toplama işlevi, spx\_rets ile yuvarlanma korelasyonunu hesaplayabilir.

```
In [103]: corr = returns.AAPL.rolling(125, min_periods=100).corr(spx_rets)
```

```
In [104]: corr.plot()
```

```
Out[104]: <Axes: >
```

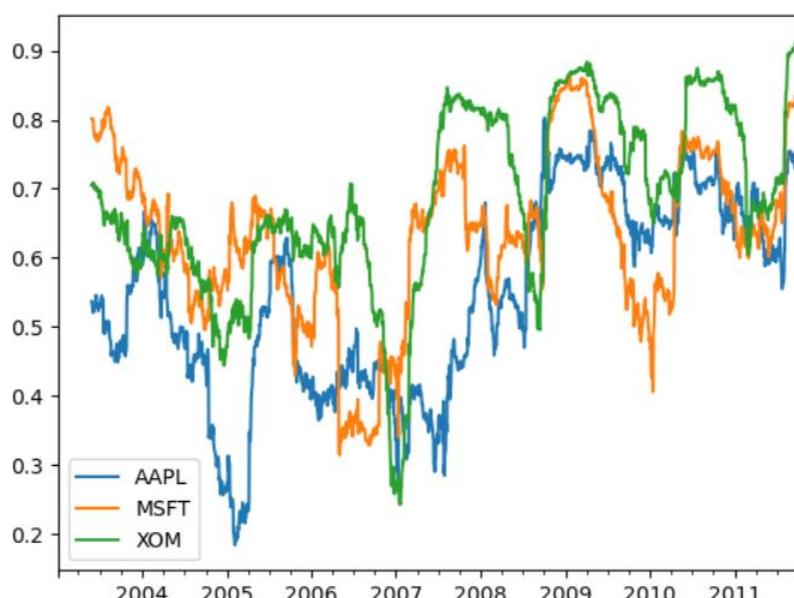


- Diyelim ki S&P 500 endeksinin birçok hisse senedi ile korelasyonunu aynı anda hesaplamak istiyorsunuz. Bir döngü yazmak ve yeni bir DataFrame oluşturmak kolay olabilir ancak tekrarlanabilir olabilir; bu nedenle, bir Seri ve bir DataFrame ileterseniz, roll\_corr gibi bir işlev, Serinin (bu durumda spx\_rets) DataFrame'deki her sütunla korelasyonunu hesaplar.

```
In [105]: corr = returns.rolling(125, min_periods=100).corr(spx_rets)
```

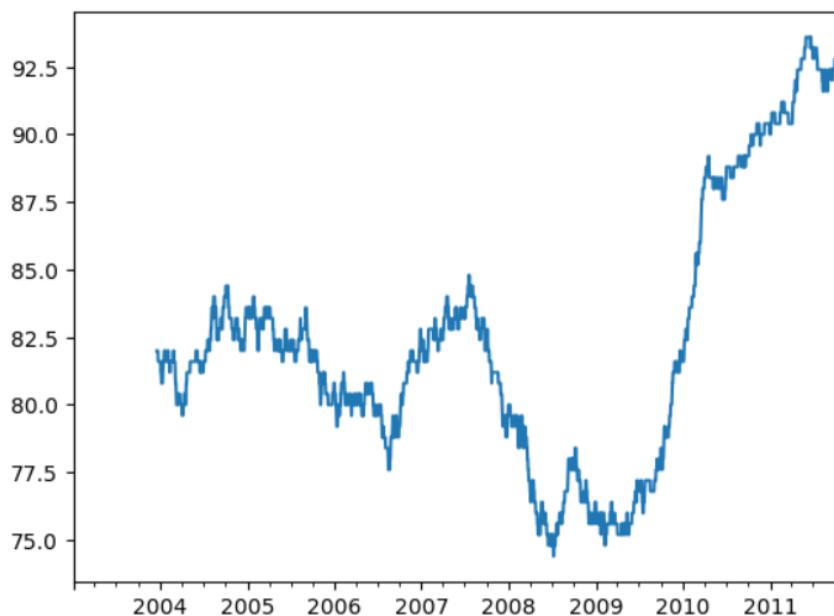
```
In [106]: corr.plot()
```

```
Out[106]: <Axes: >
```



## Kullanıcı Tanımlı Hareketli Pencere Fonksiyonları

```
In [107]: from scipy.stats import percentileofscore  
In [108]: score_at_2percent = lambda x: percentileofscore(x, 0.02)  
In [109]: result = returns.AAPL.rolling(250).apply(score_at_2percent)  
In [110]: result.plot()  
  
Out[110]: <Axes: >
```



## Gelişmiş Pandalar

### Kategorik Veriler

#### Arka Plan ve Motivasyon

- Sıklıkla, bir tablodaki bir sütun, daha küçük bir dizi farklı değerin tekrarlanan örneklerini içerebilir. Unique ve value\_counts gibi fonksiyonlar sırasıyla bir diziden farklı değerleri çıkarmamızı ve frekanslarını hesaplamamızı sağlar.

```
In [1]: import numpy as np; import pandas as pd

In [2]: values = pd.Series(['apple', 'orange', 'apple',
                           'apple'] * 2)

In [3]: values

Out[3]: 0      apple
        1    orange
        2      apple
        3      apple
        4      apple
        5    orange
        6      apple
        7      apple
       dtype: object

In [4]: pd.unique(values)

Out[4]: array(['apple', 'orange'], dtype=object)

In [5]: pd.value_counts(values)

Out[5]: apple    6
        orange   2
       Name: count, dtype: int64
```

- Birçok veri sistemi (veri ambarı, istatistiksel hesaplama veya diğer kullanıcılar için), daha verimli depolama ve hesaplama amacıyla verileri tekrarlanan değerlerle temsil etmek için özel yaklaşımlar geliştirmiştir. Veri ambarlamada en iyi uygulama, farklı değerleri içeren ve birincil gözlemleri boyut tablosuna referans veren tamsayı anahtarları olarak saklayan boyut tabloları adı verilen tabloları kullanmaktadır.

```
In [6]: values = pd.Series([0, 1, 0, 0] * 2)

In [7]: dim = pd.Series(['apple', 'orange'])

In [8]: values

Out[8]: 0    0
        1    1
        2    0
        3    0
        4    0
        5    1
        6    0
        7    0
       dtype: int64

In [9]: dim

Out[9]: 0      apple
        1    orange
       dtype: object
```

- Orijinal dizi dizisini geri yüklemek için take yöntemini kullanabiliriz.

```
In [10]: dim.take(values)
```

```
Out[10]: 0    apple
         1    orange
         0    apple
         0    apple
         0    apple
         1    orange
         0    apple
         0    apple
dtype: object
```

## Pandalarda Kategorik Tip

- pandalar, tam sayıya dayalı kategorik gösterimi veya kodlamayı kullanan verileri tutmak için özel bir Kategorik türe sahiptir.

```
In [11]: fruits = ['apple', 'orange', 'apple', 'apple'] * 2
```

```
In [12]: N = len(fruits)
```

```
In [13]: df = pd.DataFrame({'fruit': fruits,
                           'basket_id': np.arange(N),
                           'count': np.random.randint(3, 15, size=N),
                           'weight': np.random.uniform(0, 4, size=N)},
                           columns=['basket_id', 'fruit', 'count', 'weight'])
```

```
In [14]: df
```

```
Out[14]:
```

	basket_id	fruit	count	weight
0	0	apple	14	0.678482
1	1	orange	11	3.916972
2	2	apple	5	0.098849
3	3	apple	12	0.755215
4	4	apple	6	0.985321
5	5	orange	4	0.848468
6	6	apple	10	0.824659
7	7	apple	8	3.003058

- Burada df['fruit'] Python dize nesnelerinin bir dizisidir. Bunu arayarak kategorik hale getirebiliriz:

```
In [19]: fruit_cat = df['fruit'].astype('category')

In [20]: fruit_cat

Out[20]: 0    apple
1    orange
2    apple
3    apple
4    apple
5    orange
6    apple
7    apple
Name: fruit, dtype: category
Categories (2, object): ['apple', 'orange']
```

- Fruit\_cat'in değerleri bir NumPy dizisi değil, pandas.Categorical'ın bir örneğidir.

```
In [21]: c = fruit_cat.values

In [22]: type(c)

Out[22]: pandas.core.arrays.categorical.Categorical
```

- Ayrıca pandas.Categorical'ı doğrudan diğer Python dizilerinden de oluşturabilirsiniz.

```
In [23]: my_categories = pd.Categorical(['foo', 'bar', 'baz', 'foo', 'bar'])

In [24]: my_categories

Out[24]: ['foo', 'bar', 'baz', 'foo', 'bar']
Categories (3, object): ['bar', 'baz', 'foo']
```

- Başka bir kaynaktan kategorik kodlanmış veri aldıysanız alternatif from\_codes yapıcısını kullanabilirsiniz.

```
In [25]: categories = ['foo', 'bar', 'baz']

In [26]: codes = [0, 1, 2, 0, 0, 1]

In [27]: my_cats_2 = pd.Categorical.from_codes(codes, categories)

In [28]: my_cats_2

Out[28]: ['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo', 'bar', 'baz']
```

- Açıkça belirtilmediği sürece kategorik dönüşümler, kategorilerin belirli bir sıralamasını varsaymaz.

- Dolayısıyla kategoriler dizisi, giriş verilerinin sırasına bağlı olarak farklı bir sırada olabilir. `from_codes` veya diğer yapıclardan herhangi birini kullanırken, kategorilerin anlamlı bir sıralamaya sahip olduğunu belirtebilirsiniz:

```
In [29]: ordered_cat = pd.Categorical.from_codes(codes, categories,
                                                ordered=True)
```

```
In [30]: ordered_cat
```

```
Out[30]: ['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo' < 'bar' < 'baz']
```

- [`foo < bar < baz`] çıktısı, sıralamada 'foo'nun 'bar'dan önce geldiğini belirtir, vb. Sırasız bir kategorik örnek `as_ordered` ile tanımlanabilir.

```
In [31]: my_cats_2.as_ordered()
```

```
Out[31]: ['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo' < 'bar' < 'baz']
```

## Kategoriklerle Hesaplamalar

```
In [32]: np.random.seed(12345)
```

```
In [33]: draws = np.random.randn(1000)
```

```
In [34]: draws[:5]
```

```
Out[34]: array([-0.20470766,  0.47894334, -0.51943872, -0.5557303 ,  1.96578057])
```

- Bu verilerin dörtte birlik grupperlemesini hesaplayalım ve bazı istatistikler çıkaralım:

```
In [35]: bins = pd.qcut(draws, 4)
```

```
In [36]: bins
```

```
Out[36]: [(-0.684, -0.0101], (-0.0101, 0.63], (-0.684, -0.0101], (-0.684, -0.0101],
          (0.63, 3.928], ..., (-0.0101, 0.63], (-0.684, -0.0101], (-2.9499999999999999
          7, -0.684], (-0.0101, 0.63], (0.63, 3.928]]
Length: 1000
Categories (4, interval[float64, right]): [(-2.9499999999999997, -0.684] <
          (-0.684, -0.0101] < (-0.0101, 0.63] < (0.63, 3.928]]
```

```
In [37]: bins = pd.qcut(draws, 4, labels=['Q1', 'Q2', 'Q3', 'Q4'])

In [38]: bins
Out[38]: ['Q2', 'Q3', 'Q2', 'Q2', 'Q4', ..., 'Q3', 'Q2', 'Q1', 'Q3', 'Q4']
Length: 1000
Categories (4, object): ['Q1' < 'Q2' < 'Q3' < 'Q4']

In [39]: bins.codes[:10]
Out[39]: array([1, 2, 1, 1, 3, 3, 2, 2, 3, 3], dtype=int8)
```

- Kategorik etiketli bin'ler, verilerdeki bin kenarları hakkında bilgi içermez, dolayısıyla bazı özet istatistikleri çıkarmak için groupby'yi kullanabiliriz:

```
In [40]: bins = pd.Series(bins, name='quartile')

In [41]: results = (pd.Series(draws)
                    .groupby(bins)
                    .agg(['count', 'min', 'max'])
                    .reset_index())

In [42]: results
Out[42]:
   quartile  count      min      max
0         Q1    250 -2.949343 -0.685484
1         Q2    250 -0.683066 -0.010115
2         Q3    250 -0.010032  0.628894
3         Q4    250  0.634238  3.927528

In [43]: results['quartile']
Out[43]: 0    Q1
1    Q2
2    Q3
3    Q4
Name: quartile, dtype: category
Categories (4, object): ['Q1' < 'Q2' < 'Q3' < 'Q4']
```

## Kategoriklerle Daha İyi Performans

- Belirli bir veri kümesi üzerinde çok sayıda analiz yapıyorsanız kategorik veri kümesine dönüştürmek önemli genel performans kazanımları sağlayabilir. DataFrame sütununun kategorik bir sürümü de genellikle ölçüde daha az bellek kullanır. 10 milyon öğeli ve az sayıda farklı kategoriye sahip bazı Serileri ele alalım:

```
In [44]: N = 10000000
In [45]: draws = pd.Series(np.random.randn(N))
In [46]: labels = pd.Series(['foo', 'bar', 'baz', 'qux'] * (N // 4))
In [47]: categories = labels.astype('category')
In [48]: labels.memory_usage()
Out[48]: 80000132
In [49]: categories.memory_usage()
Out[49]: 10000336
```

- Etiketlerin kategorilere göre çok daha fazla bellek kullandığını görüyoruz.

## Kategorik Yöntemler

- Kategorik verileri içeren seriler, Series.str özel dize yöntemlerine benzer birkaç özel yönteme sahiptir. Bu aynı zamanda kategorilere ve kodlara kolay erişim sağlar.

```
In [51]: s = pd.Series(['a', 'b', 'c', 'd'] * 2)
In [52]: cat_s = s.astype('category')
In [53]: cat_s
Out[53]: 0    a
          1    b
          2    c
          3    d
          4    a
          5    b
          6    c
          7    d
dtype: category
Categories (4, object): ['a', 'b', 'c', 'd']
```

- cat özel niteliği kategorik yöntemlere erişim sağlar.

```
In [54]: cat_s.cat.codes
Out[54]: 0    0
          1    1
          2    2
          3    3
          4    0
          5    1
          6    2
          7    3
dtype: int8
```

```
In [55]: cat_s.cat.categories
```

```
Out[55]: Index(['a', 'b', 'c', 'd'], dtype='object')
```

```
In [56]: actual_categories = ['a', 'b', 'c', 'd', 'e']
```

```
In [57]: cat_s2 = cat_s.cat.set_categories(actual_categories)
```

```
In [58]: cat_s2
```

```
Out[58]: 0    a
          1    b
          2    c
          3    d
          4    a
          5    b
          6    c
          7    d
          dtype: category
          Categories (5, object): ['a', 'b', 'c', 'd', 'e']
```

- Veriler değişmemiş gibi görünse de yeni kategoriler, bunları kullanan operasyonlara yansıyacak. Örneğin, `value_counts`, eğer varsa kategorileri gösterir.

```
In [59]: cat_s.value_counts()
```

```
Out[59]: a    2
          b    2
          c    2
          d    2
          Name: count, dtype: int64
```

```
In [60]: cat_s2.value_counts()
```

```
Out[60]: a    2
          b    2
          c    2
          d    2
          e    0
          Name: count, dtype: int64
```

- Büyük veri kümelerinde kategoriler genellikle bellek tasarrufu ve daha iyi performans için uygun bir araç olarak kullanılır. Büyük bir DataFrame veya Seriyi filtreledikten sonra, çoğu kategoriler verilerde görünmeyebilir. Bu konuda yardımcı olmak için, gözlemlenmeyen kategorileri kırpmak amacıyla `remove_unused_categories` yöntemini kullanabiliriz:

```
In [61]: cat_s3 = cat_s[cat_s.isin(['a', 'b'])]

In [62]: cat_s3
Out[62]: 0    a
1    b
4    a
5    b
dtype: category
Categories (4, object): ['a', 'b', 'c', 'd']

In [63]: cat_s3.cat.remove_unused_categories()
Out[63]: 0    a
1    b
4    a
5    b
dtype: category
Categories (2, object): ['a', 'b']
```

## Pandalardaki Seriler İçin Kategorik Yöntemler

Method	Description
add_categories	Append new (unused) categories at end of existing categories
as_ordered	Make categories ordered
as_unordered	Make categories unordered
remove_categories	Remove categories, setting any removed values to null
remove_unused_categories	Remove any category values which do not appear in the data
rename_categories	Replace categories with indicated set of new category names; cannot change the number of categories
reorder_categories	Behaves like rename_categories, but can also change the result to have ordered categories
set_categories	Replace the categories with the indicated set of new categories; can add or remove categories

## Modelleme İçin Kukla Değişkenler Oluşturma

- İstatistikleri veya makine öğrenimi araçlarını kullanırken, genellikle kategorik verileri tek-etkin kodlama olarak da bilinen kukla değişkenlere dönüştürürsünüz. Bu, her farklı kategori için bir sütuna sahip bir DataFrame oluşturmayı içerir; bu sütunlar belirli bir kategorinin oluşumları için 1'leri, aksi takdirde 0'ları içerir.

```
In [64]: cat_s = pd.Series(['a', 'b', 'c', 'd'] * 2, dtype='category')
```

```
In [65]: pd.get_dummies(cat_s)
```

```
Out[65]:
```

	a	b	c	d
0	True	False	False	False
1	False	True	False	False
2	False	False	True	False
3	False	False	False	True
4	True	False	False	False
5	False	True	False	False
6	False	False	True	False
7	False	False	False	True

## Gelişmiş GroupBy Kullanımı

### Grup Dönüşümleri ve “Sarmalanmamış” GroupBys

- Grubun şekline yayınlanacak skaler bir değer üretebilir
- Giriş grubuyla aynı şekilde sahip bir nesne üretebilir
- Girişini değiştirmemelidir

```
In [66]: df = pd.DataFrame({'key': ['a', 'b', 'c'] * 4,  
                           'value': np.arange(12.)})
```

```
In [67]: df
```

```
Out[67]:
```

	key	value
0	a	0.0
1	b	1.0
2	c	2.0
3	a	3.0
4	b	4.0
5	c	5.0
6	a	6.0
7	b	7.0
8	c	8.0
9	a	9.0
10	b	10.0
11	c	11.0

```
In [68]: g = df.groupby('key').value
```

```
In [69]: g.mean()
```

```
Out[69]: key
          a    4.5
          b    5.5
          c    6.5
Name: value, dtype: float64
```

- Bunun yerine `df['value']` ile aynı şekle sahip ancak değerlerin 'anahtar' ile gruplandırılmış ortalamayla değiştirildiği bir Seri üretmek istediğimizi varsayıyalım. Dönüşürmek için lambda `x: x.mean()` fonksiyonunu aktarabiliriz:

```
In [70]: g.transform(lambda x: x.mean())
```

```
Out[70]: 0    4.5
         1    5.5
         2    6.5
         3    4.5
         4    5.5
         5    6.5
         6    4.5
         7    5.5
         8    6.5
         9    4.5
        10   5.5
        11   6.5
Name: value, dtype: float64
```

```
In [71]: g.transform('mean')
```

```
Out[71]: 0    4.5
         1    5.5
         2    6.5
         3    4.5
         4    5.5
         5    6.5
         6    4.5
         7    5.5
         8    6.5
         9    4.5
        10   5.5
        11   6.5
Name: value, dtype: float64
```

- `Apply` gibi, `transform` da Seri döndüren işlevlerle çalışır ancak sonuç, girdiyle aynı boyutta olmalıdır. Örneğin, lambda fonksiyonunu kullanarak her grubu 2 ile çarpabiliriz:

```
In [72]: g.transform(lambda x: x * 2)
```

```
out[72]: 0    0.0
         1    2.0
         2    4.0
         3    6.0
         4    8.0
         5   10.0
         6   12.0
         7   14.0
         8   16.0
         9   18.0
        10   20.0
        11   22.0
Name: value, dtype: float64
```

```
In [73]: g.transform(lambda x: x.rank(ascending=False))
```

```
out[73]: 0    4.0
         1    4.0
         2    4.0
         3    3.0
         4    3.0
         5    3.0
         6    2.0
         7    2.0
         8    2.0
         9    1.0
        10   1.0
        11   1.0
Name: value, dtype: float64
```

- 'Ortalama' veya 'toplam' gibi yerleşik toplama işlevleri genellikle genel uygulama işlevinden çok daha hızlıdır. Bunların aynı zamanda dönüşümle birlikte kullanıldığında "hızlı bir geçmişi" vardır. Bu sarmalanmamış grup işlemi olarak adlandırılan işlemi gerçekleştirmemizi sağlar.

```
In [75]: g.transform('mean')
```

```
out[75]: 0    4.5
         1    5.5
         2    6.5
         3    4.5
         4    5.5
         5    6.5
         6    4.5
         7    5.5
         8    6.5
         9    4.5
        10   5.5
        11   6.5
Name: value, dtype: float64
```

```
In [76]: normalized = (df['value'] - g.transform('mean')) / g.transform('std')
```

```
In [77]: normalized
```

```
Out[77]: 0    -1.161895
         1    -1.161895
         2    -1.161895
         3    -0.387298
         4    -0.387298
         5    -0.387298
         6     0.387298
         7     0.387298
         8     0.387298
         9     1.161895
        10    1.161895
        11    1.161895
Name: value, dtype: float64
```

## Gruplandırılmış Zaman Yeniden Örnekleme

- Zaman serisi verileri için yeniden örnekleme yöntemi anlamsal olarak zaman aralığına dayalı bir grup işlemidir.

```
In [78]: N = 15
```

```
In [79]: times = pd.date_range('2017-05-20 00:00', freq='1min', periods=N)
```

```
In [80]: df = pd.DataFrame({'time': times,
                           'value': np.arange(N)})
```

```
In [81]: df
```

```
Out[81]:
```

	time	value
0	2017-05-20 00:00:00	0
1	2017-05-20 00:01:00	1
2	2017-05-20 00:02:00	2
3	2017-05-20 00:03:00	3
4	2017-05-20 00:04:00	4
5	2017-05-20 00:05:00	5
6	2017-05-20 00:06:00	6
7	2017-05-20 00:07:00	7
8	2017-05-20 00:08:00	8
9	2017-05-20 00:09:00	9
10	2017-05-20 00:10:00	10
11	2017-05-20 00:11:00	11
12	2017-05-20 00:12:00	12
13	2017-05-20 00:13:00	13
14	2017-05-20 00:14:00	14

```
In [82]: df.set_index('time').resample('5min').count()
```

out[82]:

	value
time	
2017-05-20 00:00:00	5
2017-05-20 00:05:00	5
2017-05-20 00:10:00	5

- Bir DataFrame'in, ek bir grup anahtar sütunu ile işaretlenmiş birden çok zaman serisi içerdigini varsayalim:

```
In [83]: df2 = pd.DataFrame({'time': times.repeat(3),
                           'key': np.tile(['a', 'b', 'c'], N),
                           'value': np.arange(N * 3.)})
```

```
In [84]: df2[:7]
```

out[84]:

	time	key	value
0	2017-05-20 00:00:00	a	0.0
1	2017-05-20 00:00:00	b	1.0
2	2017-05-20 00:00:00	c	2.0
3	2017-05-20 00:01:00	a	3.0
4	2017-05-20 00:01:00	b	4.0
5	2017-05-20 00:01:00	c	5.0
6	2017-05-20 00:02:00	a	6.0

- Anahtar'in her değeri için aynı yeniden örneklemeyi yapmak amacıyla pandas.Time Grouper nesnesini tanıtıyoruz:

```
In [85]: time_key = pd.TimeGrouper('5min')
```

## Yöntem Zincirleme Teknikleri

- Bir veri kümesine bir dizi dönüşüm uygularken, kendinizi analizinizde hiç kullanılmayan çok sayıda geçici değişken oluştururken bulabilirsiniz.

```
In [88]: df = load_data()
df2 = df[df['col2'] < 0]
df2['col1_demeaned'] = df2['col1'] - df2['col1'].mean()
result = df2.groupby('key').col1_demeaned.std()
```

- Burada herhangi bir gerçek veri kullanmıyor olsak da bu örnek bazı yeni yöntemleri vurgulamaktadır. Birincisi, DataFrame.sign yöntemi,  $df[k] = v$  biçimindeki sütun atamalarına işlevsel bir alternatifdir. Nesneyi yerinde değiştirmek yerine, belirtilen değişikliklerle yeni bir DataFrame döndürür.

```
In [ ]: # Her zamanki işlevsel olmayan yol
df2 = df.copy()
df2['k'] = v
```

```
In [ ]: # İşlevsel atama yolu
df2 = df.assign(k=v)
```

- Yerinde atama, atamayı kullanmaktan daha hızlı yürütülebilir, ancak atama, daha kolay yöntem zincirleme sağlar.

```
In [ ]: result = (df2.assign(col1_demeaned=df2.col1 - df2.col2.mean())
                 .groupby('key')
                 .col1_demeaned.std())
```

- Yöntem zincirleme yaparken akılda tutulması gereken bir şey, geçici nesnelere başvurmanız gerekebileceğidir.

```
In [ ]: df = load_data()
df2 = df[df['col2'] < 0]
```

```
In [ ]: df = (load_data()
            [lambda x: x['col2'] < 0])
```

- Çağrılabilir öğeleri çalışırken göstermek için şu şekilde yeniden yazılabilir.
- Burada load\_data'nın sonucu bir değişkene atanmaz, bu nedenle []'ye iletilen işlev daha sonra yöntem zincirinin o aşamasındaki nesneye bağlanır.
- O halde devam edebilir ve tüm diziyi tek bir zincirleme ifade olarak yazabiliz:

```
In [ ]: result = (load_data()
                  [lambda x: x.col2 < 0]
                  .assign(col1_demeaned=lambda x: x.col1 - x.col1.mean())
                  .groupby('key')
                  .col1_demeaned.std())
```

## Pipe Yöntemi

- Bazen kendi işlevlerinizi veya üçüncü taraf kitaplıklardaki işlevleri kullanmanız gereklidir. Pipe yönteminin devreye girdiği yer burasıdır.
- Bir dizi işlev çağrısını düşünün:

```
In [90]: a = f(df, arg1=v1)
          b = g(a, v2, arg3=v3)
          c = h(b, arg4=v4)
```

- Series veya DataFrame nesnelerini kabul eden ve döndüren işlevleri kullanırken, bunu pipe çağrılarını kullanarak yeniden yazabilirsiniz.

```
In [ ]: result = (df.pipe(f, arg1=v1)
                  .pipe(g, v2, arg3=v3)
                  .pipe(h, arg4=v4))
```

- `f(df)` ve `df.pipe(f)` ifadesi eşdeğerdir ancak pipe zincirleme çağrıyı kolaylaştırır.
- Pipe için potansiyel olarak yararlı bir model, işlem dizilerini yeniden kullanılabılır işlevlere genelleştirmektir. Örnek olarak, grup ortalamalarını bir değerden çıkarmayı düşünelim.

```
In [ ]: g = df.groupby(['key1', 'key2'])
          df['col1'] = df['col1'] - g.transform('mean')
```

- Birden fazla sütunu küçültmek ve grup anahtarlarını kolayca değiştirmek istediğiniz varsayıyalım.

```
In [ ]: def group_demean(df, by, cols):
          result = df.copy()
          g = df.groupby(by)
          for c in cols:
              result[c] = df[c] - g[c].transform('mean')
          return result
```

- O zaman şunu yazmak mümkündür:

```
In [ ]: result = (df[df.col1 < 0]
                  .pipe(group_demean, ['key1', 'key2'], ['col1']))
```

# Python'da Modelleme Kütüphanelerine Giriş

## Pandalar ve Model Kodu Arasında Arayüz Oluşturma

- Model geliştirme için yaygın bir iş akışı, modelin kendisini oluşturmak için bir modelleme kitaplığına geçmeden önce veri yükleme ve temizleme için pandaları kullanmaktadır.
- Pandalar ve diğer analiz kütüphaneleri arasındaki temas noktası genellikle NumPy dizileridir. DataFrame'i NumPy dizisine dönüştürmek için .values özelliğini kullanın.

```
In [2]: import pandas as pd
```

```
In [3]: import numpy as np
```

```
In [4]: data = pd.DataFrame({  
    'x0': [1, 2, 3, 4, 5],  
    'x1': [0.01, -0.01, 0.25, -4.1, 0.],  
    'y': [-1.5, 0., 3.6, 1.3, -2.]})
```

```
In [5]: data
```

```
Out[5]:
```

	x0	x1	y
0	1	0.01	-1.5
1	2	-0.01	0.0
2	3	0.25	3.6
3	4	-4.10	1.3
4	5	0.00	-2.0

```
In [6]: data.columns
```

```
Out[6]: Index(['x0', 'x1', 'y'], dtype='object')
```

```
In [7]: data.values
```

```
Out[7]: array([[ 1. ,  0.01, -1.5 ],  
   [ 2. , -0.01,  0. ],  
   [ 3. ,  0.25,  3.6 ],  
   [ 4. , -4.1 ,  1.3 ],  
   [ 5. ,  0. , -2. ]])
```

- Önceki bölümlerden hatırlayabileceğiniz gibi, tekrar DataFrame'e dönüştürmek için, isteğe bağlı sütun adlarıyla iki boyutlu bir ndarray iletebilirsiniz:

```
In [8]: df2 = pd.DataFrame(data.values, columns=['one', 'two', 'three'])
```

```
In [9]: df2
```

```
out[9]:
```

	one	two	three
0	1.0	0.01	-1.5
1	2.0	-0.01	0.0
2	3.0	0.25	3.6
3	4.0	-4.10	1.3
4	5.0	0.00	-2.0

```
In [10]: df3 = data.copy()
```

```
In [11]: df3['strings'] = ['a', 'b', 'c', 'd', 'e']
```

```
In [12]: df3
```

```
out[12]:
```

	x0	x1	y	strings
0	1	0.01	-1.5	a
1	2	-0.01	0.0	b
2	3	0.25	3.6	c
3	4	-4.10	1.3	d
4	5	0.00	-2.0	e

```
In [13]: df3.values
```

```
out[13]: array([[1, 0.01, -1.5, 'a'],
 [2, -0.01, 0.0, 'b'],
 [3, 0.25, 3.6, 'c'],
 [4, -4.1, 1.3, 'd'],
 [5, 0.0, -2.0, 'e']], dtype=object)
```

- Bazı modellerde sütunların yalnızca bir alt kümesini kullanmak isteyebilirsiniz. Loc indekslemeyi değerlerle kullanmanız önerilir.

```
In [14]: model_cols = ['x0', 'x1']
```

```
In [15]: data.loc[:, model_cols].values
```

```
out[15]: array([[ 1. ,  0.01],
 [ 2. , -0.01],
 [ 3. ,  0.25],
 [ 4. , -4.1 ],
 [ 5. ,  0. ]])
```

```
In [16]: data['category'] = pd.Categorical(['a', 'b', 'a', 'a', 'b'],
                                         categories=['a', 'b'])
```

```
In [17]: data
```

```
Out[17]:
```

	x0	x1	y	category
0	1	0.01	-1.5	a
1	2	-0.01	0.0	b
2	3	0.25	3.6	a
3	4	-4.10	1.3	a
4	5	0.00	-2.0	b

- 'Kategori' sütununu kukla değişkenlerle değiştirmek istersek, kukla değişkenler yaratır, 'kategori' sütununu bırakır ve ardından sonucu birleştiririz:

```
In [18]: dummies = pd.get_dummies(data.category, prefix='category')
```

```
In [19]: data_with_dummies = data.drop('category', axis=1).join(dummies)
```

```
In [20]: data_with_dummies
```

```
Out[20]:
```

	x0	x1	y	category_a	category_b
0	1	0.01	-1.5	True	False
1	2	-0.01	0.0	False	True
2	3	0.25	3.6	True	False
3	4	-4.10	1.3	True	False
4	5	0.00	-2.0	False	True

## Patsy ile Model Açıklamaları Oluşturma

- Patsy, R ve S istatistiksel programlama dilleri tarafından kullanılan formül sözdiziminden ilham alan (ancak tam olarak aynı olmayan) küçük dize tabanlı "formül sözdizimi" ile istatistiksel modelleri (özellikle doğrusal modelleri) tanımlamak için kullanılan bir Python kütüphanesidir.
- Patsy'nin formülleri şuna benzeyen özel bir dize sözdizimidir:

$$y \sim x_0 + x_1$$

- $a + b$  sözdizimi  $a$ 'nın  $b$ 'ye eklenmesi anlamına gelmez, bunun yerine bunların model için oluşturulan tasarım matrisindeki terimler olduğu anlamına gelir. `patsy.dmatrices` işlevi, bir veri kümesiyle (bir DataFrame veya dizi dizisi olabilir) birlikte bir formül dizesi alır ve doğrusal bir model için tasarım matrisleri üretir.

```
In [24]: data = pd.DataFrame({
    'x0': [1, 2, 3, 4, 5],
    'x1': [0.01, -0.01, 0.25, -4.1, 0.],
    'y': [-1.5, 0., 3.6, 1.3, -2.]})
```

```
In [25]: data
```

```
Out[25]:
      x0    x1    y
0   1   0.01  -1.5
1   2   -0.01   0.0
2   3    0.25   3.6
3   4   -4.10   1.3
4   5    0.00  -2.0
```

```
In [26]: import patsy
```

```
In [27]: y, X = patsy.dmatrices('y ~ x0 + x1', data)
```

```
In [28]: y
```

```
Out[28]: DesignMatrix with shape (5, 1)
      y
-1.5
0.0
3.6
1.3
-2.0
Terms:
'y' (column 0)
```

```
In [30]: X
```

```
Out[30]: DesignMatrix with shape (5, 3)
      Intercept  x0    x1
1   1   0.01
1   2   -0.01
1   3    0.25
1   4   -4.10
1   5    0.00
Terms:
'Intercept' (column 0)
'x0' (column 1)
'x1' (column 2)
```

- Bu Patsy DesignMatrix örnekleri, ek meta verilere sahip NumPy ndarray'leridir.

```
In [31]: np.asarray(y)
```

```
Out[31]: array([[-1.5],  
                 [ 0. ],  
                 [ 3.6],  
                 [ 1.3],  
                 [-2. ]])
```

```
In [32]: np.asarray(x)
```

```
Out[32]: array([[ 1. ,  1. ,  0.01],  
                 [ 1. ,  2. , -0.01],  
                 [ 1. ,  3. ,  0.25],  
                 [ 1. ,  4. , -4.1 ],  
                 [ 1. ,  5. ,  0. ]])
```

- Intercept teriminin nereden geldiğini merak edebilirsiniz. Bu, sıradan en küçük kareler (OLS) regresyonu gibi doğrusal modeller için bir kurallıdır. Modele + 0 terimini ekleyerek kesişmeyi bastırabilirsiniz.

```
In [33]: patsy.dmatrices('y ~ x0 + x1 + 0', data)[1]
```

```
Out[33]: DesignMatrix with shape (5, 2)
```

```
x0      x1  
1  0.01  
2 -0.01  
3  0.25  
4 -4.10  
5  0.00
```

```
Terms:
```

```
'x0' (column 0)  
'x1' (column 1)
```

- Patsy nesneleri, sıradan bir en küçük kareler regresyonu gerçekleştiren numpy.linalg.lstsq gibi algoritmalarla doğrudan aktarılabilir.

```
In [34]: coef, resid, _, _ = np.linalg.lstsq(x, y)
```

- Model meta verileri design\_info özelliğinde tutulur, böylece bir Seri elde etmek için model sütun adlarını uygun katsayılaraya yeniden bağlayabilirsiniz, örneğin:

```
In [35]: coef
```

```
Out[35]: array([[ 0.31290976],  
                 [-0.07910564],  
                 [-0.26546384]])
```

```
In [36]: coef = pd.Series(coef.squeeze(), index=X.design_info.column_names)

In [37]: coef

Out[37]: Intercept    0.312910
          x0        -0.079106
          x1       -0.265464
          dtype: float64
```

## Patsy Formüllerinde Veri Dönüşümleri

```
In [38]: y, X = patsy.dmatrices('y ~ x0 + np.log(np.abs(x1) + 1)', data)

In [39]: X

Out[39]: DesignMatrix with shape (5, 3)
          Intercept  x0  np.log(np.abs(x1) + 1)
          1           1      0.00995
          1           2      0.00995
          1           3      0.22314
          1           4      1.62924
          1           5      0.00000

          Terms:
          'Intercept' (column 0)
          'x0' (column 1)
          'np.log(np.abs(x1) + 1)' (column 2)
```

- Yaygın olarak kullanılan bazı değişken dönüşümleri arasında standartlaştırma (0 ve varyans 1) ve merkezleme (ortalamanın çıkarılması). Patsy'nin bunun için yerleşik işlevleri vardır.

```
In [40]: y, X = patsy.dmatrices('y ~ standardize(x0) + center(x1)', data)

In [41]: X

Out[41]: DesignMatrix with shape (5, 3)
          Intercept  standardize(x0)  center(x1)
          1           -1.41421     0.78
          1            -0.70711    0.76
          1            0.00000    1.02
          1            0.70711   -3.33
          1            1.41421     0.77

          Terms:
          'Intercept' (column 0)
          'standardize(x0)' (column 1)
          'center(x1)' (column 2)
```

- Modelleme sürecinin bir parçası olarak, bir modeli bir veri kümesine sığdırabilir, ardından modeli diğerine göre değerlendirebilirsiniz. Bu bir uzatma kısmı veya daha sonra gözlemlenen yeni bir veri olabilir. Merkezileştirme ve standartlaştırma gibi dönüşümleri uygularken, modeli yeni verilere dayalı tahminler oluşturmak için kullanırken dikkatli olmalısınız. Bunlara durum bilgisi olan dönüşümler denir çünkü yeni bir veri kümesini dönüştürken orijinal veri kümesinin ortalaması veya standart sapması gibi istatistikleri kullanmanız gereklidir.
- `patsy.build_design_matrices` işlevi, orijinal örnek içi veri kümesinden kaydedilen bilgileri kullanarak yeni örnek dışı verilere dönüşümler uygulayabilir:

```
In [42]: new_data = pd.DataFrame({
    'x0': [6, 7, 8, 9],
    'x1': [3.1, -0.5, 0, 2.3],
    'y': [1, 2, 3, 4]})

In [43]: new_X = patsy.build_design_matrices([X.design_info], new_data)

In [44]: new_X

Out[44]: [DesignMatrix with shape (4, 3)
   Intercept standardize(x0) center(x1)
       1           2.12132      3.87
       1           2.82843      0.27
       1           3.53553      0.77
       1           4.24264      3.07
   Terms:
   'Intercept' (column 0)
   'standardize(x0)' (column 1)
   'center(x1)' (column 2)]
```

Patsy formülleri bağlamındaki artı simgesi (+) toplama anlamına gelmediğinden, bir veri kümesinden sütunları ada göre eklemek istediğinizde bunları özel I işlevi:

```
In [45]: y, X = patsy.dmatrices('y ~ I(x0 + x1)', data)

In [46]: X

Out[46]: DesignMatrix with shape (5, 2)
   Intercept I(x0 + x1)
       1           1.01
       1           1.99
       1           3.25
       1          -0.10
       1           5.00
   Terms:
   'Intercept' (column 0)
   'I(x0 + x1)' (column 1)
```

## Kategorik Veriler ve Patsy

- Sayısal olmayan veriler, bir model tasarım matrisi için birçok farklı yolla dönüştürülebilir.
- Patsy formülünde sayısal olmayan terimler kullandığınızda, bunlar varsayılan olarak yapay değişkenlere dönüştürülür. Bir kesinti varsa, önlemek için seviyelerden biri dışında bırakılacaktır.

```
In [47]: data = pd.DataFrame({
    'key1': ['a', 'a', 'b', 'b', 'a', 'b', 'a', 'b'],
    'key2': [0, 1, 0, 1, 0, 1, 0, 0],
    'v1': [1, 2, 3, 4, 5, 6, 7, 8],
    'v2': [-1, 0, 2.5, -0.5, 4.0, -1.2, 0.2, -1.7]
})
```

```
In [48]: y, X = patsy.dmatrices('v2 ~ key1', data)
```

```
In [49]: X
```

```
Out[49]: DesignMatrix with shape (8, 2)
Intercept  key1[T.b]
1          0
1          0
1          1
1          1
1          0
1          1
1          0
1          1

Terms:
'Intercept' (column 0)
'key1' (column 1)
```

- Modelden kesme noktasını çıkarırsanız, her kategori değerine ilişkin sütunlar model tasarım matrisine dahil edilecektir:

```
In [50]: y, X = patsy.dmatrices('v2 ~ key1 + 0', data)
```

```
In [51]: X
```

```
Out[51]: DesignMatrix with shape (8, 2)
key1[a]  key1[b]
1          0
1          0
0          1
0          1
1          0
0          1
1          0
0          1

Terms:
'key1' (columns 0:2)
```

- Sayısal sütunlar, C işleviyle kategorik olarak yorumlanabilir.

```
In [52]: y, X = patsy.dmatrices('v2 ~ C(key2)', data)
```

```
In [53]: X
```

```
Out[53]: DesignMatrix with shape (8, 2)
```

	Intercept	C(key2)[T.1]
1	1	0
1	1	1
1	1	0
1	1	1
1	1	0
1	1	1
1	1	0
1	1	0

Terms:

- 'Intercept' (column 0)
- 'C(key2)' (column 1)

- Bir modelde birden çok kategorik terim kullandığınızda, örneğin varyans analizi (ANOVA) modellerinde kullanılabilen anahtar1:anahtar2 biçimindeki etkileşim terimlerini dahil edebileceğiniz için işler daha karmaşık olabilir.

```
In [54]: data['key2'] = data['key2'].map({0: 'zero', 1: 'one'})
```

```
In [55]: data
```

```
Out[55]:
```

	key1	key2	v1	v2
0	a	zero	1	-1.0
1	a	one	2	0.0
2	b	zero	3	2.5
3	b	one	4	-0.5
4	a	zero	5	4.0
5	b	one	6	-1.2
6	a	zero	7	0.2
7	b	zero	8	-1.7

```
In [56]: y, X = patsy.dmatrices('v2 ~ key1 + key2', data)
```

```
In [57]: X
```

```
Out[57]: DesignMatrix with shape (8, 3)
```

	Intercept	key1[T.b]	key2[T.zero]
1	1	0	1
1	1	0	0
1	1	1	1
1	1	1	0
1	1	0	1
1	1	1	0
1	1	0	1
1	1	1	1

Terms:

- 'Intercept' (column 0)
- 'key1' (column 1)
- 'key2' (column 2)

```
In [58]: y, X = patsy.dmatrices('v2 ~ key1 + key2 + key1:key2', data)
```

```
In [59]: X
```

```
Out[59]: DesignMatrix with shape (8, 4)
Intercept  key1[T.b]  key2[T.zero]  key1[T.b]:key2[T.zero]
1          0           1             0
1          0           0             0
1          1           1             1
1          1           0             0
1          0           1             0
1          1           0             0
1          0           1             0
1          1           1             1

Terms:
'Intercept' (column 0)
'key1' (column 1)
'key2' (column 2)
'key1:key2' (column 3)
```

# İstatistik Modellerine Giriş

## Doğrusal Modelleri Tahmin Etme

- İstatistik modellerinde, daha temelden (örneğin, sıradan en küçük kareler) daha karmaşık olana (örneğin, yinelemeli olarak yeniden ağırlıklandırılmış en küçük kareler) kadar çeşitli türde doğrusal regresyon modelleri vardır.
- İstatistik modellerindeki doğrusal modellerin iki farklı ana arayüzü vardır: dizi tabanlı ve formül tabanlı. Bunlara şu API modülü içe aktarmaları aracılığıyla erişilir:

```
In [*]: import statsmodels.api as sm
import statsmodels.formula.api as smf
```

- Bunların nasıl kullanılacağını göstermek için bazı rastgele verilerden doğrusal bir model oluşturuyoruz:

```
In [2]: def dnorm(mean, variance, size=1):
    if isinstance(size, int):
        size = size,
    return mean + np.sqrt(variance) * np.random.randn(*size)
```

```
In [6]: np.random.seed(12345)
```

```
In [7]: N = 100
X = np.c_[dnorm(0, 0.4, size=N),
          dnorm(0, 0.6, size=N),
          dnorm(0, 0.2, size=N)]
eps = dnorm(0, 0.1, size=N)
beta = [0.1, 0.3, 0.5]
y = np.dot(X, beta) + eps
```

- Burada bilinen parametrelere sahip “true” model beta olarak yazıldı. Bu durumda dnorm, belirli bir ortalama ve varyansa sahip normal dağılmış veriler üretmek için yardımcı bir fonksiyondur.

```
In [8]: X[:5]
```

```
Out[8]: array([[-0.12946849, -1.21275292,  0.50422488],
   [ 0.30291036, -0.43574176, -0.25417986],
   [-0.32852189, -0.02530153,  0.13835097],
   [-0.35147471, -0.71960511, -0.25821463],
   [ 1.2432688 , -0.37379916, -0.52262905]])
```

```
In [9]: y[:5]
```

```
Out[9]: array([ 0.42786349, -0.67348041, -0.09087764, -0.48949442, -0.12894109])
```

- Doğrusal bir model genellikle daha önce Patsy'de gördüğümüz gibi bir kesme terimiyle donatılır. sm.add\_constant işlevi mevcut bir matrise bir kesme sütunu ekleyebilir:

```
In [10]: X_model = sm.add_constant(X)
```

```
In [11]: X_model[:5]
```

```
Out[11]: array([[ 1.        , -0.12946849, -1.21275292,  0.50422488],
   [ 1.        ,  0.30291036, -0.43574176, -0.25417986],
   [ 1.        , -0.32852189, -0.02530153,  0.13835097],
   [ 1.        , -0.35147471, -0.71960511, -0.25821463],
   [ 1.        ,  1.2432688 , -0.37379916, -0.52262905]])
```

- sm.OLS sınıfı sıradan bir en küçük kareler doğrusal regresyonuna uyabilir:

```
In [13]: model = sm.OLS(y, X)
```

- Modelin uyum yöntemi, tahmini model parametrelerini ve diğer teşhisleri içeren bir regresyon sonuçları nesnesini döndürür:

```
In [14]: results = model.fit()
```

```
In [15]: results.params
```

```
Out[15]: array([0.17826108, 0.22303962, 0.50095093])
```

- Sonuçlara ilişkin özet yöntemi, modelin tanılama çıktısını ayrıntılandıran bir model yazdırabilir:

```
In [16]: print(results.summary())
```

```
OLS Regression Results
=====
Dep. Variable: y R-squared (uncentered): 0.430
Model: OLS Adj. R-squared (uncentered): 0.413
Method: Least Squares F-statistic: 24.42
Date: Wed, 20 Dec 2023 Prob (F-statistic): 7.44e-12
Time: 21:47:18 Log-Likelihood: -34.305
No. Observations: 100 AIC: 74.61
Df Residuals: 97 BIC: 82.42
Df Model: 3
Covariance Type: nonrobust
=====

=
      coef  std err      t      P>|t|      [0.025      0.97
5]
-----
x1      0.1783  0.053   3.364    0.001     0.073     0.28
x2      0.2230  0.046   4.818    0.000     0.131     0.31
x3      0.5010  0.080   6.237    0.000     0.342     0.66
0
=====
=
Omnibus: 4.662 Durbin-Watson: 2.20
1
Prob(Omnibus): 0.097 Jarque-Bera (JB): 4.09
8
Skew: 0.481 Prob(JB): 0.12
9
Kurtosis: 3.243 Cond. No. 1.7
4
=====
=
Notes:
[1] R2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

- Buradaki parametre adlarına x1, x2 vb. genel adlar verilmiştir. Bunun yerine tüm model parametrelerinin bir DataFrame'de olduğunu varsayıalım:

```
In [17]: data = pd.DataFrame(x, columns=['col0', 'col1', 'col2'])
```

```
In [18]: data['y'] = y
```

```
In [19]: data[:5]
```

```
Out[19]:
```

	col0	col1	col2	y
0	-0.129468	-1.212753	0.504225	0.427863
1	0.302910	-0.435742	-0.254180	-0.673480
2	-0.328522	-0.025302	0.138351	-0.090878
3	-0.351475	-0.719605	-0.258215	-0.489494
4	1.243269	-0.373799	-0.522629	-0.128941

- Artık statsmodels formül API'sini ve Patsy formül dizelerini kullanabiliriz:

```
In [20]: results = smf.ols('y ~ col0 + col1 + col2', data=data).fit()
```

```
In [21]: results.params
```

```
Out[21]: Intercept      0.033559
          col0          0.176149
          col1          0.224826
          col2          0.514808
          dtype: float64
```

```
In [22]: results.tvalues
```

```
Out[22]: Intercept      0.952188
          col0          3.319754
          col1          4.850730
          col2          6.303971
          dtype: float64
```

## Zaman Serisi Süreçlerinin Tahmin Edilmesi

- İstatistik modellerindeki diğer bir model sınıfı, zaman serisi analizi içindir. Bunlar arasında otoregresif süreçler, Kalman filtreleme ve diğer durum uzayı modelleri ve çok değişkenli otoregresif modeller yer almaktadır.
- Bazı zaman serisi verilerini otoregresif yapı ve gürültü ile simüle edelim:

```
In [24]: init_x = 4

import random
values = [init_x, init_x]
N = 1000

b0 = 0.8
b1 = -0.4
noise = dnorm(0, 0.1, N)
for i in range(N):
    new_x = values[-1] * b0 + values[-2] * b1 + noise[i]
    values.append(new_x)
```

- Bu veri, 0.8 ve -0.4 parametreleriyle bir AR(2) yapısına (iki gecikme) sahiptir. Bir AR modeli taktığınızda, dahil edilecek gecikmeli terimlerin sayısını bilemeyebilirsiniz, bu nedenle modele daha fazla sayıda gecikme sığdırabilirsiniz.

```
In [27]: MAXLAGS = 5
```

```
In [ ]: model = sm.tsa.AR(values)
```

```
In [ ]: results = model.fit(MAXLAGS)
```

- Sonuçlardaki tahmin edilen parametreler ilk önce kesişmeyi ve ardından ilk iki gecikmeye ilişkin tahminleri içerir.

```
In [32]: results.params
```

```
out[32]: Intercept      0.033559
          col0          0.176149
          col1          0.224826
          col2          0.514808
          dtype: float64
```

## Scikit-Learn'e Giriş

- scikit-learn, en yaygın kullanılan ve güvenilir genel amaçlı Python makine öğrenimi araç takımlarından biridir. Model seçimi ve değerlendirmesi, veri dönüşümü, veri yükleme ve model kalıcılığı için araçlarla birlikte geniş bir standart denetimli ve denetimsiz makine öğrenimi yöntemleri yelpazesi içerir. Bu modeller sınıflandırma, kümeleme, tahmin ve diğer ortak görevler için kullanılabilir.

- Örnek olarak, 1912'de batan Titanik'teki yolcuların hayatı kalma oranlarıyla ilgili bir Kaggle yarışmasından artık klasik olan bir veri kümesini kullanıyor.

```
In [33]: train = pd.read_csv('train.csv')
In [34]: test = pd.read_csv('test.csv')
In [35]: train[:4]
Out[35]:
   PassengerId  Survived  Pclass   Name     Sex   Age  SibSp  Parch   Ticket  Fare Cabin Embarked
0      1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5 21111,    NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
1      2,1,1,"Cumings, Mrs. John Bradley (Florence Br...     NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
2      3,1,3,"Heikkinen, Miss. Laina",female,26,0,0,S...     NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
3      4,1,1,"Futrelle, Mrs. Jacques Heath (Lily May ...     NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
```

- statsmodels ve scikit-learn gibi kütüphaneler genellikle eksik verilerle beslenemez, bu nedenle eksik veri içeren herhangi bir şey olup olmadığını görmek için sütunlara bakarız:

```
In [36]: train.isnull().sum()
```

```
Out[36]: PassengerId      0
Survived        891
Pclass          891
Name           891
Sex            891
Age            891
SibSp          891
Parch          891
Ticket         891
Fare           891
Cabin          891
Embarked       891
dtype: int64
```

```
In [37]: test.isnull().sum()
```

```
Out[37]: PassengerId      0
Pclass          418
Name           418
Sex            418
Age            418
SibSp          418
Parch          418
Ticket         418
Fare           418
Cabin          418
Embarked       418
dtype: int64
```

- Bunun gibi istatistiklerde ve makine öğrenimi örneklerinde tipik bir görev, verilerdeki özelliklere dayanarak bir yolcunun hayatı kalıp kalamayacağını tahmin etmektir. Bir model, bir eğitim veri kümesine yerleştirilir ve ardından örnek dışı bir test veri kümesi üzerinde değerlendirilir.

- Yaş bir tahmin aracı olarak kullanılmak isteniyor ancak eksik verileri var. Eksik veri atamasını yapmanın birkaç yolu vardır, ancak basit bir yol kullanılır ve her iki tablodaki boş değerleri doldurmak için eğitim veri kümesinin ortancası kullanılır.

```
In [38]: impute_value = train['Age'].median()
```

```
In [39]: train['Age'] = train['Age'].fillna(impute_value)
```

```
In [40]: test['Age'] = test['Age'].fillna(impute_value)
```

- Daha sonra bazı model değişkenlerine karar verip NumPy dizileri oluşturuyoruz.

```
In [45]: predictors = ['Pclass', 'IsFemale', 'Age']
```

```
In [46]: X_train = train[predictors].values
```

```
In [47]: X_test = test[predictors].values
```

```
In [48]: y_train = train['Survived'].values
```

```
In [49]: X_train[:5]
```

```
Out[49]: array([[nan, 0., nan],
 [nan, 0., nan],
 [nan, 0., nan],
 [nan, 0., nan],
 [nan, 0., nan]])
```

```
In [50]: y_train[:5]
```

```
Out[50]: array([nan, nan, nan, nan, nan])
```

- Scikit-learn'deki LogisticRegression modelini kullanıyoruz ve bir model örneği oluşturuyoruz.

```
In [51]: from sklearn.linear_model import LogisticRegression
```

```
In [52]: model = LogisticRegression()
```

- İstatistik modellerine benzer şekilde, bu modeli, modelin uyum yöntemini kullanarak eğitim verilerine sığdırılabilir.

```
In [ ]: model.fit(X_train, y_train)
```

```
Out[104]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False)
```

- Artık `model.predict` kullanarak test veri seti için tahminler oluşturulabilir.

```
In [ ]: y_predict = model.predict(X_test)
```

```
In [ ]: y_predict[:10]
```

```
Out[106]: array([0, 0, 0, 0, 1, 0, 1, 0, 1, 0])
```

- Test veri kümesi için doğru değerlere sahipseniz doğruluk yüzdesini veya başka bir hata ölçüsünü hesaplayabilirsiniz:

```
In [56]: (y_true == y_predict).mean()
```

- Çapraz doğrulama, örnek dışı tahmini simüle etmek için eğitim verilerini bölgerek çalışır. Ortalama karesel hata gibi bir model doğruluk puanına dayanarak, model parametreleri üzerinde bir ızgara araması yapılabilir. Lojistik regresyon gibi bazı modellerde yerleşik çapraz doğrulamaya sahip tahminci sınıfları bulunur. Örneğin, `LogisticRegressionCV` sınıfı, model düzenleme parametresi `C`'de bir ızgara aramasının ne kadar ayrıntılı bir şekilde yapılacağını gösteren bir parametreyle kullanılabilir.

```
In [107]: from sklearn.linear_model import LogisticRegressionCV
In [108]: model_cv = LogisticRegressionCV(10)
In [109]: model_cv.fit(X_train, y_train)
Out[109]:
LogisticRegressionCV(Cs=10, class_weight=None, cv=None, dual=False,
fit_intercept=True, intercept_scaling=1.0, max_iter=100,
multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
refit=True, scoring=None, solver='lbfgs', tol=0.0001, verbose=0)
```

- Çapraz doğrulamayı elle yapmak için veri bölme işlemini gerçekleştiren `cross_val_score` yardımcı işlevini kullanabilirsiniz.

Örneğin, modeli çapraz doğrulamak için eğitim verilerini birbirine örtüşmeyen dört bölüme ayırlırsa şunlar yapılabilir.

```
In [110]: from sklearn.model_selection import cross_val_score
In [111]: model = LogisticRegression(C=10)
In [112]: scores = cross_val_score(model, X_train, y_train, cv=4)
In [113]: scores
Out[113]: array([ 0.7723,  0.8027,  0.7703,  0.7883])
```

# VERİ ANALİZİ ÖRNEKLERİ

## Bitly'den USA.gov Verileri

- 2011 yılında URL kısaltma hizmeti Bitly, .gov veya .mil ile biten bağlantıları kısaltan kullanıcılarından toplanan anonim verilerin akışını sağlamak için ABD hükümetinin web sitesi USA.gov ile ortaklık kurdu.
- Saatlik anlık görüntüler söz konusu olduğunda, her dosyadaki her satır, JSON olarak bilinen ve JavaScript Nesne Gösterimi anlamına gelen ortak bir web verisi biçimini içerir. Örneğin, bir dosyanın yalnızca ilk satırını okursak şöyle bir şey görebiliriz:

```
In [1]: path = 'example.txt'

In [2]: open(path).readline()

Out[2]: '{ "a": "Mozilla\\5.0 (Windows NT 6.1; WOW64) AppleWebKit\\535.11 (KHTML, like Gecko) Chrome\\17.0.963.78 Safari\\535.11", "c": "US", "nk": 1, "tz": "America\\New_York", "gr": "MA", "g": "A6qOVH", "h": "wfLQtf", "l": "orofrog", "al": "en-US,en;q=0.8", "hh": "1.usa.gov", "r": "http:\\\\www.facebook.com\\l\\7AQEFzjSi\\1.usa.gov\\wfLQtf", "u": "http:\\\\www.ncbi.nlm.nih.gov\\pubmed\\22415991", "t": 1331923247, "hc": 1331822918, "cy": "Danvers", "ll": [ 42.576698, -70.954903 ] }\n'
```

- Python, bir JSON dizesini Python sözlüğü nesnesine dönüştürmek için hem yerleşik hem de üçüncü taraf kitaplıklara sahiptir. Burada json modülünü ve onun yüklemeye fonksiyonunu kullanacağız. Örnek dosyanın her satırında çağrıldı:
- Ortaya çıkan nesne kayıtları artık Python sözlerinin bir listesidir.

```
In [4]: import json
path = 'example.txt'
records = [json.loads(line) for line in open(path)]

In [5]: records[0]

Out[5]: {'a': 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.78 Safari/535.11',
          'c': 'US',
          'nk': 1,
          'tz': 'America/New_York',
          'gr': 'MA',
          'g': 'A6qOVH',
          'h': 'wfLQtf',
          'l': 'orofrog',
          'al': 'en-US,en;q=0.8',
          'hh': '1.usa.gov',
          'r': 'http://www.facebook.com/l/7AQEFzjSi/1.usa.gov/wfLQtf',
          'u': 'http://www.ncbi.nlm.nih.gov/pubmed/22415991',
          't': 1331923247,
          'hc': 1331822918,
          'cy': 'Danvers',
          'll': [42.576698, -70.954903]}
```

## Pure Python'da Zaman Dilimlerini Sayma

- Veri kümesinde en sık meydana gelen zaman dilimlerini bulmakla ilgili olduğumuzu varsayıyalım. (tz alanı). Bunu yapmanın birçok yolu var. İlk olarak, liste kavramayı kullanarak saat dilimlerinin bir listesini tekrar çıkaralım:

```
In [6]: time_zones = [rec['tz'] for rec in records]

-----
KeyError                                     Traceback (most recent call last)
Cell In[6], line 1
----> 1 time_zones = [rec['tz'] for rec in records]

Cell In[6], line 1, in <listcomp>(.0)
----> 1 time_zones = [rec['tz'] for rec in records]

KeyError: 'tz'
```

- Hata! Tüm kayıtların saat dilimi alanına sahip olmadığı ortaya çıktı. Listenin anlaşılmamasının sonuna rec'de 'tz' olup olmadığını kontrol edebildiğimiz için bunu yapmak kolaydır.

```
In [7]: time_zones = [rec['tz'] for rec in records if 'tz' in rec]

In [8]: time_zones[:10]

Out[8]: ['America/New_York',
         'America/Denver',
         'America/New_York',
         'America/Sao_Paulo',
         'America/New_York',
         'America/New_York',
         'Europe/Warsaw',
         '',
         '',
         '']
```

- Sadece ilk 10 zaman dilimine baktığımızda bazılarının bilinmediğini (boş dize) görüyoruz. Bunları da filtreleyebilirsiniz.
- Zaman dilimine göre sayımlar üretmek için iki yaklaşım vardır: daha zor yol (yalnızca Python standart kütüphanesini kullanarak) ve daha kolay yol (pandaları kullanarak). Saymayı yapmanın bir yolu, zaman dilimleri arasında yineleme yaparken sayıları depolamak için bir dict kullanmaktır:

```
In [9]: def get_counts(sequence):
    counts = {}
    for x in sequence:
        if x in counts:
            counts[x] += 1
        else:
            counts[x] = 1
    return counts
```

- Python standart kütüphanesindeki daha gelişmiş araçları kullanarak aynı şeyi daha kısaca yazabilirsiniz:

```
In [10]: from collections import defaultdict

def get_counts2(sequence):
    counts = defaultdict(int) # değerler 0 olarak başlatılacak
    for x in sequence:
        counts[x] += 1
    return counts
```

- Bu mantığı daha tekrar kullanılabilir hale getirmek için bir fonksiyona kullanıldı. Bunu saat dilimlerinde kullanmak için time\_zones listesini iletmeniz yeterlidir.

```
In [11]: counts = get_counts(time_zones)
```

```
In [12]: counts['America/New_York']
```

```
Out[12]: 1251
```

```
In [13]: len(time_zones)
```

```
Out[13]: 3440
```

- İlk 10 zaman dilimini ve bunların sayılarını istiyorsak:

```
In [17]: def top_counts(count_dict, n=10):
    value_key_pairs = [(count, tz) for tz, count in count_dict.items()]

In [20]: value_key_pairs.sort()
         return value_key_pairs[-n:]
```

```
In [22]: top_counts(counts)

Out[21]:
[(33, 'America/Sao_Paulo'),
 (35, 'Europe/Madrid'),
 (36, 'Pacific/Honolulu'),
 (37, 'Asia/Tokyo'),
 (74, 'Europe/London'),
 (191, 'America/Denver'),
 (382, 'America/Los_Angeles'),
 (400, 'America/Chicago'),
 (521, ''),
 (1251, 'America/New_York')]
```

- Python standart kütüphanesinde arama yaparsanız koleksiyonları bulabilirsiniz. Bu görevi çok daha kolaylaştıran sayaç sınıfı:

```
In [23]: from collections import Counter
```

```
In [24]: counts = Counter(time_zones)
```

```
In [25]: counts.most_common(10)
```

```
Out[25]: [('America/New_York', 1251),
 ('', 521),
 ('America/Chicago', 400),
 ('America/Los_Angeles', 382),
 ('America/Denver', 191),
 ('Europe/London', 74),
 ('Asia/Tokyo', 37),
 ('Pacific/Honolulu', 36),
 ('Europe/Madrid', 35),
 ('America/Sao_Paulo', 33)]
```

## Pandalarla Saat Dilimlerini Sayma

- Orijinal kayıt kümesinden bir DataFrame oluşturmak, kayıt listesini pandas.DataFrame'e aktarmak kadar kolaydır.

```
In [26]: import pandas as pd
```

```
In [27]: frame = pd.DataFrame(records)
```

```
In [28]: frame.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3560 entries, 0 to 3559
Data columns (total 18 columns):
 #   Column      Non-Null Count Dtype  
 ---  --          -----          ----  
 0   a            3440 non-null   object  
 1   c            2919 non-null   object  
 2   nk           3440 non-null   float64 
 3   tz           3440 non-null   object  
 4   gr           2919 non-null   object  
 5   g            3440 non-null   object  
 6   h            3440 non-null   object  
 7   l            3440 non-null   object  
 8   al           3094 non-null   object  
 9   hh           3440 non-null   object  
 10  r             3440 non-null   object  
 11  u             3440 non-null   object  
 12  t             3440 non-null   float64 
 13  hc           3440 non-null   float64 
 14  cy           2919 non-null   object  
 15  ll           2919 non-null   object  
 16  _heartbeat_  120 non-null   float64 
 17  kw           93 non-null    object  
dtypes: float64(4), object(14)
memory usage: 500.8+ KB

```

In [29]: `frame['tz'][:10]`

```

Out[29]: 0    America/New_York
          1    America/Denver
          2    America/New_York
          3    America/Sao_Paulo
          4    America/New_York
          5    America/New_York
          6    Europe/Warsaw
          7
          8
          9
Name: tz, dtype: object

```

- Frame için gösterilen çıktı, büyük DataFrame nesneleri için gösterilen özet görünümüdür. Daha sonra Series için value\_counts yöntemini kullanabiliriz:

In [30]: `tz_counts = frame['tz'].value_counts()`

In [31]: `tz_counts[:10]`

```

Out[31]: tz
          America/New_York    1251
                      521
          America/Chicago     400
          America/Los_Angeles  382
          America/Denver       191
          Europe/London        74
          Asia/Tokyo           37
          Pacific/Honolulu     36
          Europe/Madrid         35
          America/Sao_Paulo     33
Name: count, dtype: int64

```

- Bu verileri matplotlib kullanarak görselleştirebiliriz. Kayıtlardaki bilinmeyen ve eksik saat dilimi verilerinin yerine gelecek değeri doldurmak için biraz munging yapabilirsiniz. Eksik değerleri fillna yöntemiyle değiştiriyoruz ve boş dizeler için boole dizi indekslemeyi kullanıyoruz.

```
In [32]: clean_tz = frame['tz'].fillna('Missing')
```

```
In [33]: clean_tz[clean_tz == ''] = 'Unknown'
```

```
In [34]: tz_counts = clean_tz.value_counts()
```

```
In [36]: tz_counts[:10]
```

```
Out[36]: tz
America/New_York      1251
Unknown                521
America/Chicago        400
America/Los_Angeles   382
America/Denver         191
Missing                120
Europe/London          74
Asia/Tokyo              37
Pacific/Honolulu       36
Europe/Madrid           35
Name: count, dtype: int64
```

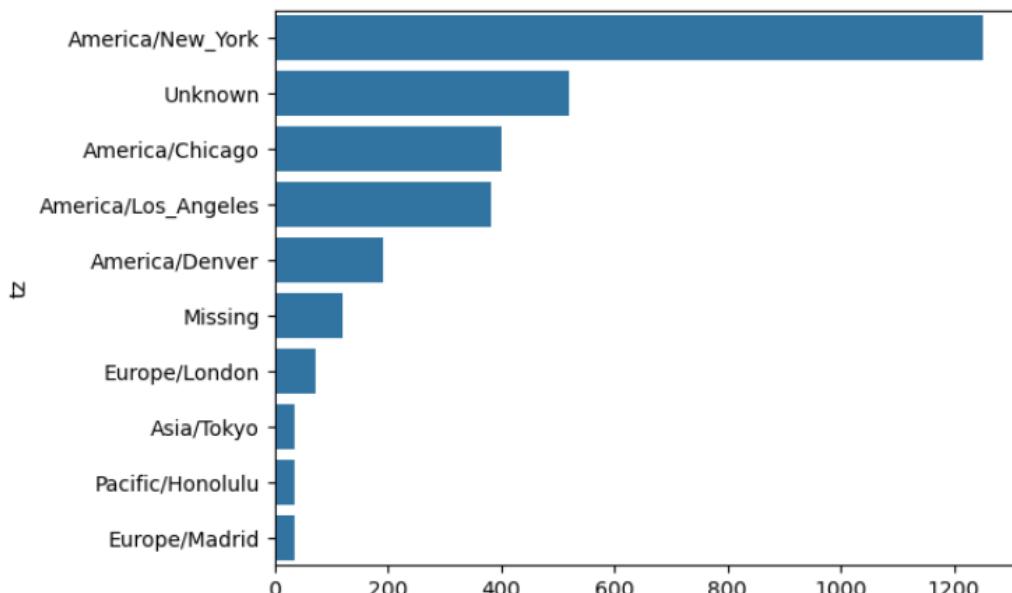
- Bu noktada yatay çubuk grafiği oluşturmak için seaborn paketini kullanabiliriz.

```
In [37]: import seaborn as sns
```

```
In [38]: subset = tz_counts[:10]
```

```
In [39]: sns.barplot(y=subset.index, x=subset.values)
```

```
Out[39]: <Axes: ylabel='tz'>
```



- a alanı, URL kısaltmayı gerçekleştirmek için kullanılan tarayıcı, cihaz veya uygulama hakkında bilgi içerir.

```
In [40]: frame['a'][1]
Out[40]: 'GoogleMaps/RochesterNY'

In [41]: frame['a'][50]
Out[41]: 'Mozilla/5.0 (Windows NT 5.1; rv:10.0.2) Gecko/20100101 Firefox/10.0.2'

In [42]: frame['a'][51][:50] # uzun çizgi
Out[42]: 'Mozilla/5.0 (Linux; U; Android 2.2.2; en-us; LG-P9'

In [43]: results = pd.Series([x.split()[0] for x in frame.a.dropna()])
In [44]: results[:5]
Out[44]: 0      Mozilla/5.0
1      GoogleMaps/RochesterNY
2      Mozilla/4.0
3      Mozilla/5.0
4      Mozilla/5.0
dtype: object

In [45]: results.value_counts()[:8]
Out[45]:
Mozilla/5.0           2594
Mozilla/4.0            601
GoogleMaps/RochesterNY   121
Opera/9.80              34
TEST_INTERNET_AGENT      24
GoogleProducer            21
Mozilla/6.0               5
BlackBerry8520/5.0.0.681    4
Name: count, dtype: int64
```

- Şimdi, en önemli saat dilimlerini Windows kullanıcıları ve Windows dışı kullanıcılar olarak ayırtmak istedığınızı varsayıyalım. Basitleştirme olarak, dizinin şu durumda olması durumunda kullanıcının Windows'ta olduğunu varsayıyalım: 'Windows' agent dizesindedir. Temsilcilerden bazıları eksik olduğundan bunları verilerden hariç tutulur.

```
In [46]: cframe = frame[frame.a.notnull()]
```

- Daha sonra her satırın Windows olup olmadığına ilişkin bir değer hesaplamak istenirse:

```
In [49]: cframe['os'] = np.where(cframe['a'].str.contains('Windows'),
                                'Windows', 'Not Windows')
```

```
In [50]: cframe['os'][:5]
```

```
out[50]: 0      Windows
          1    Not Windows
          2      Windows
          3    Not Windows
          4      Windows
Name: os, dtype: object
```

- Ardından verileri saat dilimi sütununa ve bu yeni işletim sistemleri listesine göre gruplandırabilirsiniz:
- Value\_counts işlevine benzer şekilde grup sayımları boyutla hesaplanabilir. Bu sonuç daha sonra unstack ile bir tabloya dönüştürülür.

```
In [51]: by_tz_os = cframe.groupby(['tz', 'os'])
```

```
In [52]: agg_counts = by_tz_os.size().unstack().fillna(0)
```

```
In [53]: agg_counts[:10]
```

```
Out[53]:
```

os	Not Windows	Windows
tz		
	245.0	276.0
Africa/Cairo	0.0	3.0
Africa/Casablanca	0.0	1.0
Africa/Ceuta	0.0	2.0
Africa/Johannesburg	0.0	1.0
Africa/Lusaka	0.0	1.0
America/Anchorage	4.0	1.0
America/Argentina/Buenos_Aires	1.0	0.0
America/Argentina/Cordoba	0.0	1.0
America/Argentina/Mendoza	0.0	1.0

- Son olarak, en iyi genel saat dilimlerini seçelim. Bunu yapmak için agg\_counts'taki satır sayımlarından dolaylı bir dizin dizisi oluşturulur.

```
In [54]: # Artan düzende sıralamak için kullanılan
           indexer = agg_counts.sum(1).argsort()
```

```
In [55]: indexer[:10]
```

```
Out[55]: tz
          24
          Africa/Cairo
          Africa/Casablanca
          Africa/Ceuta
          Africa/Johannesburg
          Africa/Lusaka
          America/Anchorage
          America/Argentina/Buenos_Aires
          America/Argentina/Cordoba
          America/Argentina/Mendoza
          dtype: int64
```

- Satırları bu sırayla seçmek için take kullanılır, ardından son 10 satır(en büyük değerler) dilimlenir.

```
In [56]: count_subset = agg_counts.take(indexer[-10:])
```

```
In [57]: count_subset
```

```
out[57]:
```

	os	Not Windows	Windows
	tz		
America/Sao_Paulo		13.0	20.0
Europe/Madrid		16.0	19.0
Pacific/Honolulu		0.0	36.0
Asia/Tokyo		2.0	35.0
Europe/London		43.0	31.0
America/Denver		132.0	59.0
America/Los_Angeles		130.0	252.0
America/Chicago		115.0	285.0
		245.0	276.0
America/New_York		339.0	912.0

- pandaların nlargest adında aynı şeyi yapan kullanışlı bir yöntemi vardır.

```
In [58]: agg_counts.sum(1).nlargest(10)
```

```
out[58]: tz
```

America/New_York	1251.0
	521.0
America/Chicago	400.0
America/Los_Angeles	382.0
America/Denver	191.0
Europe/London	74.0
Asia/Tokyo	37.0
Pacific/Honolulu	36.0
Europe/Madrid	35.0
America/Sao_Paulo	33.0

dtype: float64

- Daha sonra, önceki kod bloğunda gösterildiği gibi bu, bir çubuk grafiğinde çizilebilir.

```
In [59]: # Çizim için verileri yeniden düzenleyin
count_subset = count_subset.stack()
```

```
In [60]: count_subset.name = 'total'
```

```
In [61]: count_subset = count_subset.reset_index()
```

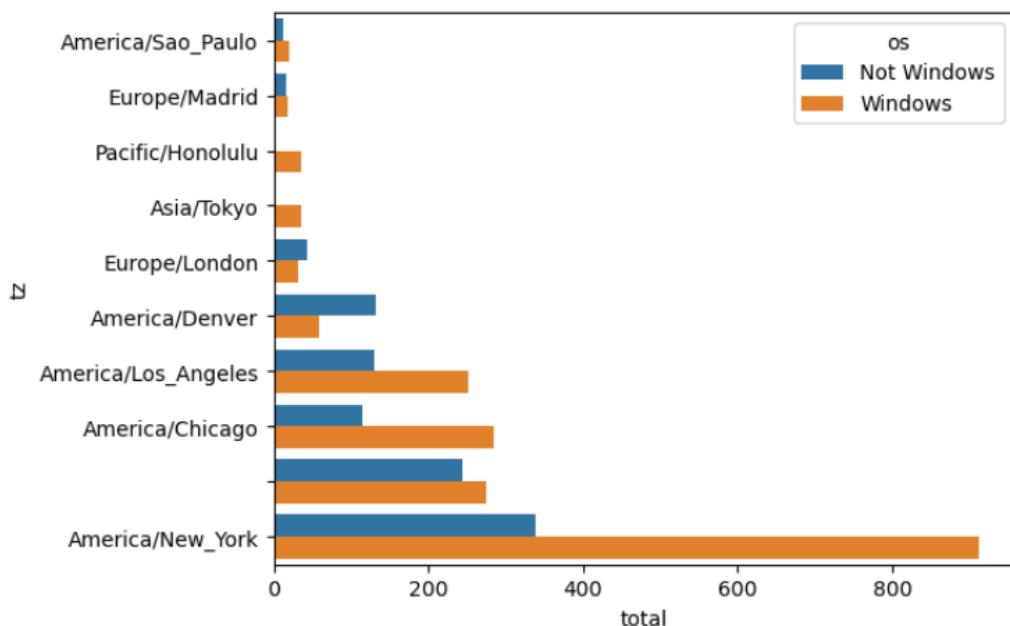
```
In [62]: count_subset[:10]
```

```
Out[62]:
```

	tz	os	total
0	America/Sao_Paulo	Not Windows	13.0
1	America/Sao_Paulo	Windows	20.0
2	Europe/Madrid	Not Windows	16.0
3	Europe/Madrid	Windows	19.0
4	Pacific/Honolulu	Not Windows	0.0
5	Pacific/Honolulu	Windows	36.0
6	Asia/Tokyo	Not Windows	2.0
7	Asia/Tokyo	Windows	35.0
8	Europe/London	Not Windows	43.0
9	Europe/London	Windows	31.0

```
In [63]: sns.barplot(x='total', y='tz', hue='os', data=count_subset)
```

```
Out[63]: <Axes: xlabel='total', ylabel='tz'>
```

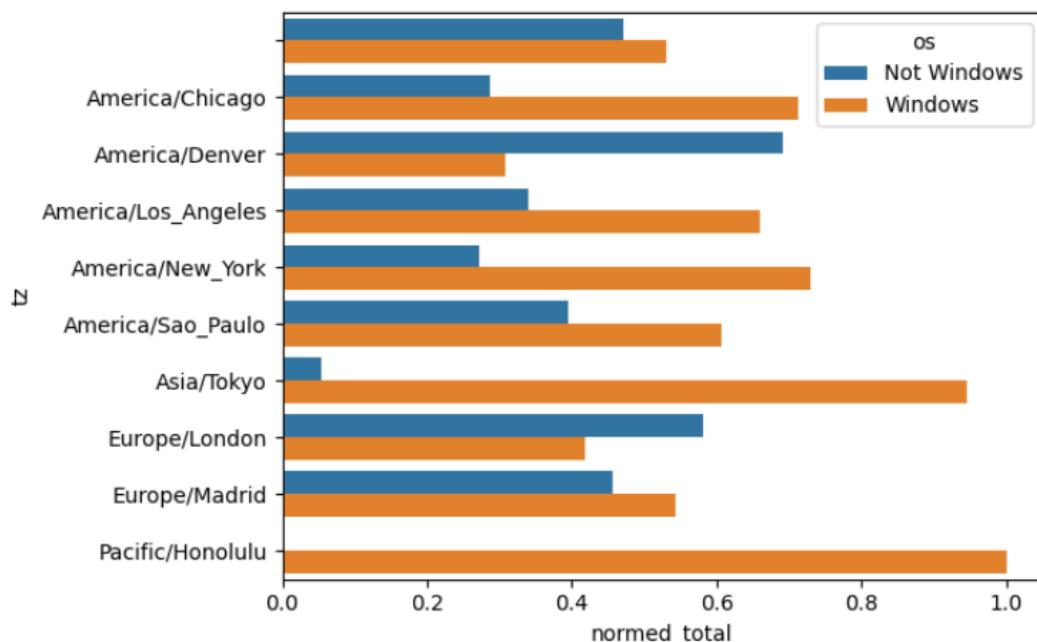


- Grafik, daha küçük gruptardaki Windows kullanıcılarının göreceli yüzdesini görmeyi kolaylaştırmıyor; bu nedenle grup yüzdelerini 1'e toplayacak şekilde normalleştirelim:

```
In [65]: def norm_total(group):  
    group['normed_total'] = group.total / group.total.sum()  
    return group  
  
results = count_subset.groupby('tz').apply(norm_total)
```

```
In [66]: sns.barplot(x='normed_total', y='tz', hue='os', data=results)
```

```
Out[66]: <Axes: xlabel='normed_total', ylabel='tz'>
```



- Normalleştirilmiş toplamı, transform yöntemini groupby ile kullanarak daha verimli bir şekilde hesaplanabilirdi.

```
In [67]: g = count_subset.groupby('tz')
```

```
In [68]: results2 = count_subset.total / g.total.transform('sum')
```

## MovieLens 1M Veri Kümesi

- GroupLens Research, 1990'ların sonu ve 2000'lerin başında MovieLens kullanıcılarından toplanan bir dizi film derecelendirme verisi koleksiyonu sağlar. Veriler, film derecelendirmelerini, film meta verilerini (türler ve yıl) ve kullanıcılarla ilgili demografik verileri (yaş, posta kodu, cinsiyet kimliği ve meslek) sağlar. Bu tür veriler genellikle makine öğrenimi algoritmalarına dayalı öneri sistemlerinin geliştirilmesinde ilgi çekicidir.
- MovieLens 1M veri seti, 4.000 film için 6.000 kullanıcıdan toplanan 1 milyon derecelendirmeyi içerir. Üç tabloya yayılmıştır: derecelendirmeler, kullanıcı bilgileri ve film bilgileri.
- Verileri ZIP dosyasından çıkardıktan sonra pandas.read\_table'ı kullanarak her tabloyu bir pandas DataFrame nesnesine yükleyebiliriz:

```
In [1]: import pandas as pd

In [2]: # Ekran küçült
pd.options.display.max_rows = 10

In [4]: unames = ['user_id', 'gender', 'age', 'occupation', 'zip']
users = pd.read_table('users.dat', sep='::',
                      header=None, names=unames)

In [5]: rnames = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_table('ratings.dat', sep='::',
                        header=None, names=rnames)

In [6]: mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('movies.dat', sep='::',
                      header=None, names=mnames)
```

```
In [7]: users[:5]
Out[7]:
   user_id  gender  age occupation      zip
0         1        F    1        10    48067
1         2        M   56        16   70072
2         3        M   25        15  55117
3         4        M   45        7  02460
4         5        M   25        20  55455
```

```
In [10]: ratings
Out[10]:
   user_id  movie_id  rating  timestamp
0         1       1193      5  978300760
1         1       661       3  978302109
2         1       914       3  978301968
3         1       3408      4  978300275
4         1       2355      5  978824291
...
1000204     6040      1091      1  956716541
1000205     6040      1094      5  956704887
1000206     6040       562      5  956704746
1000207     6040      1096      4  956715648
1000208     6040      1097      4  956715569
```

1000209 rows × 4 columns

4	5	Father of the Bride Part II (1995)	Comedy
---	---	------------------------------------	--------

- Pandaların birleştirme işlevini kullanarak, önce kullanıcılarla derecelendirmeleri, ardından bu sonucu film verileriyle birleştiriyoruz. pandalar, çakışan adlara göre merge (veya join) tuşları olarak hangi sütunların kullanılacağını belirler.

```
In [11]: data = pd.merge(pd.merge(ratings, users), movies)
In [12]: data
Out[12]:
   user_id  movie_id  rating  timestamp  gender  age  occupation  zip          title  genres
0         1       1193      5  978300760      F    1        10  48067  One Flew Over the Cuckoo's Nest (1975)  Drama
1         2       1193      5  978298413      M   56        16  70072  One Flew Over the Cuckoo's Nest (1975)  Drama
2         12      1193      4  978220179      M   25        12  32793  One Flew Over the Cuckoo's Nest (1975)  Drama
3         15      1193      4  978199279      M   25        7  22903  One Flew Over the Cuckoo's Nest (1975)  Drama
4         17      1193      5  978158471      M   50        1  95350  One Flew Over the Cuckoo's Nest (1975)  Drama
...       ...
1000204     5949     2198      5  958846401      M   18        17  47901  Modulations (1998)  Documentary
1000205     5675     2703      3  976029116      M   35        14  30030  Broken Vessels (1998)  Drama
1000206     5780     2845      1  958153068      M   18        17  92886  White Boys (1999)  Drama
1000207     5851     3607      5  957756608      F   18        20  55410  One Little Indian (1973)  Comedy|Drama|Western
1000208     5938     2909      4  957273353      M   25        1  35401  Five Wives, Three Secretaries and Me (1998)  Documentary
1000209 rows × 10 columns
```

```
In [13]: data.iloc[0]
Out[13]:
user_id           1
movie_id         1193
rating            5
timestamp        978300760
gender             F
age                1
occupation         10
zip               48067
title  One Flew Over the Cuckoo's Nest (1975)
genres            Drama
Name: 0, dtype: object
```

- Cinsiyete göre gruplandırılmış her film için ortalama film derecelendirmelerini almak için pivot\_table yöntemini kullanılabilir.

```
In [14]: mean_ratings = data.pivot_table('rating', index='title',
                                         columns='gender', aggfunc='mean')
In [15]: mean_ratings[:5]
Out[15]:
          gender      F      M
          title
$1,000,000 Duck (1971)  3.375000  2.761905
'Night Mother (1986)    3.388889  3.352941
'Til There Was You (1997) 2.675676  2.733333
'turbs, The (1989)     2.793478  2.962085
...And Justice for All (1979) 3.828571  3.689024
```

- Bu, film başlıklarının satır etiketleri ("index") ve cinsiyetin sütun etiketleri olduğu ortalama derecelendirmeleri içeren başka bir DataFrame üretti. Önce en az 250 puan alan filmlere filtre uygulanıyor; Bunu yapmak için veriler başlığa göre grupperlerdir ve her başlık için bir grup boyutu Serisi elde etmek için size() işlevi kullanılır.

```
In [16]: ratings_by_title = data.groupby('title').size()
```

```
In [17]: ratings_by_title[:10]
```

```
Out[17]: title
$1,000,000 Duck (1971)      37
'Night Mother (1986)          70
'Til There Was You (1997)     52
'burbs, The (1989)            303
...And Justice for All (1979) 199
1-900 (1994)                  2
10 Things I Hate About You (1999) 700
101 Dalmatians (1961)         565
101 Dalmatians (1996)         364
12 Angry Men (1957)           616
dtype: int64
```

```
In [18]: active_titles = ratings_by_title.index[ratings_by_title >= 250]
```

```
In [19]: active_titles
```

```
Out[19]: Index(['burbs, The (1989)', '10 Things I Hate About You (1999)',
 '101 Dalmatians (1961)', '101 Dalmatians (1996)', '12 Angry Men (1957)',
 '13th Warrior, The (1999)', '2 Days in the Valley (1996)',
 '20,000 Leagues Under the Sea (1954)', '2001: A Space Odyssey (1968)',
 '2010 (1984)',
 ...,
 'X-Men (2000)', 'Year of Living Dangerously (1982)',
 'Yellow Submarine (1968)', 'You've Got Mail (1998)',
 'Young Frankenstein (1974)', 'Young Guns (1988)',
 'Young Guns II (1990)', 'Young Sherlock Holmes (1985)',
 'Zero Effect (1998)', 'eXistenZ (1999)'],
 dtype='object', name='title', length=1216)
```

- En az 250 derecelendirme alan başlıkların dizini daha sonra ortalama\_ratings'ten satır seçmek için kullanılabilir.

```
In [20]: mean_ratings = mean_ratings.loc[active_titles]
# Indexteki satırları seç
```

```
In [21]: mean_ratings
```

```
Out[21]:
```

	gender	F	M
title			
'burbs, The (1989)	2.793478	2.962065	
10 Things I Hate About You (1999)	3.646552	3.311966	
101 Dalmatians (1961)	3.791444	3.500000	
101 Dalmatians (1996)	3.240000	2.911215	
12 Angry Men (1957)	4.184397	4.328421	
...	...	...	
Young Guns (1988)	3.371795	3.425620	
Young Guns II (1990)	2.934783	2.904025	
Young Sherlock Holmes (1985)	3.514706	3.363344	
Zero Effect (1998)	3.864407	3.723140	
eXistenZ (1999)	3.098592	3.289086	

1216 rows × 2 columns

- Kadın izleyiciler arasında en çok beğenilen filmleri görmek için F sütununa göre azalan şekilde sıralanabilir.

```
In [22]: top_female_ratings = mean_ratings.sort_values(by='F', ascending=False)
```

```
In [23]: top_female_ratings[:10]
```

Out[23]:

gender	F	M
title		
Close Shave, A (1995)	4.644444	4.473795
Wrong Trousers, The (1993)	4.588235	4.478261
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	4.572650	4.464589
Wallace & Gromit: The Best of Aardman Animation (1996)	4.563107	4.385075
Schindler's List (1993)	4.562602	4.491415
Shawshank Redemption, The (1994)	4.539075	4.560625
Grand Day Out, A (1992)	4.537879	4.293255
To Kill a Mockingbird (1962)	4.536667	4.372611
Creature Comforts (1990)	4.513889	4.272277
Usual Suspects, The (1995)	4.513317	4.518248

## Derecelendirme Anlaşmazlığının Ölçülmesi

- Erkek ve kadın izleyiciler arasında en fazla ayırm yaratan filmleri bulmak istediğiniz varsayıyalım. Bunun bir yolu, ortala\_ratings'e ortalamalar arasındaki farkı içeren bir sütun eklemek ve ardından buna göre sıralama yapmaktır.
- 'Fark'a (diff) göre sıralama, en büyük reyting farkına sahip filmleri verir, böylece hangilerinin kadınlar tarafından tercih edildiği görülebilir.

```
In [24]: mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']
```

```
In [25]: sorted_by_diff = mean_ratings.sort_values(by='diff')
```

```
In [26]: sorted_by_diff[:10]
```

Out[26]:

gender	F	M	diff
title			
Dirty Dancing (1987)	3.790378	2.959596	-0.830782
Jumpin' Jack Flash (1986)	3.254717	2.578358	-0.676359
Grease (1978)	3.975265	3.367041	-0.608224
Little Women (1994)	3.870588	3.321739	-0.548849
Steel Magnolias (1989)	3.901734	3.365957	-0.535777
Anastasia (1997)	3.800000	3.281609	-0.518391
Rocky Horror Picture Show, The (1975)	3.673016	3.160131	-0.512885
Color Purple, The (1985)	4.158192	3.659341	-0.498851
Age of Innocence, The (1993)	3.827068	3.339506	-0.487561
Free Willy (1993)	2.921348	2.438776	-0.482573

- Sıraların sırasını tersine çevrildiğinde ve ilk 10 sıra tekrar kesildiğinde, erkeklerin tercih ettiği, kadınların ise bu kadar yüksek puan vermediği filmler elde edilir.

```
In [27]: # Satırların sırasını ters çevirin, ilk 10 satırı alın
sorted_by_diff[::-1][:10]
```

Out[27]:

gender	F	M	diff
title			
Good, The Bad and The Ugly, The (1966)	3.494949	4.221300	0.726351
Kentucky Fried Movie, The (1977)	2.878788	3.555147	0.676359
Dumb & Dumber (1994)	2.697987	3.336595	0.638608
Longest Day, The (1962)	3.411765	4.031447	0.619682
Cable Guy, The (1996)	2.250000	2.863787	0.613787
Evil Dead II (Dead By Dawn) (1987)	3.297297	3.909283	0.611985
Hidden, The (1987)	3.137931	3.745098	0.607167
Rocky III (1982)	2.361702	2.943503	0.581801
Caddyshack (1980)	3.396135	3.969737	0.573602
For a Few Dollars More (1965)	3.409091	3.953795	0.544704

- Bunun yerine, cinsiyet kimliğinden bağımsız olarak izleyiciler arasında en fazla anlaşmazlığa neden olan filmleri istedığınızı varsayalım. Anlaşmazlık, derecelendirmelerin varyansı veya standart sapması ile ölçülebilir.

```
In [28]: # Başlığı göre gruplandırılmış derecelendirmenin standart sapması
rating_std_by_title = data.groupby('title')['rating'].std()
```

```
In [30]: # active_titles'e kadar filtreleyin
rating_std_by_title = rating_std_by_title.loc[active_titles]
```

```
In [31]: # Serileri değere göre azalan sırada sıralayın
rating_std_by_title.sort_values(ascending=False)[:10]
```

Out[31]: title

Dumb & Dumber (1994)	1.321333
Blair Witch Project, The (1999)	1.316368
Natural Born Killers (1994)	1.307198
Tank Girl (1995)	1.277695
Rocky Horror Picture Show, The (1975)	1.260177
Eyes Wide Shut (1999)	1.259624
Evita (1996)	1.253631
Billy Madison (1995)	1.249970
Fear and Loathing in Las Vegas (1998)	1.246408
Bicentennial Man (1999)	1.245533
Name: rating, dtype: float64	

## ABD Bebek İsimleri 1880–2010

- Amerika Birleşik Devletleri Sosyal Güvenlik İdaresi (SSA), 1880'den günümüze bebek isimlerinin sıklığına ilişkin verileri kullanıma sundu.
- Bu veri kümescini yüklemek için biraz veri düzenlemesi yapmamız gerekiyor, ancak bunu yaptığımızda şuna benzeyen bir DataFrame olacak:

```
In [4]: names.head(10)
Out[4]:
   name  sex  births  year
0  Mary    F     7065  1880
1  Anna    F     2604  1880
2  Emma    F     2003  1880
3 Elizabeth  F     1939  1880
4  Minnie  F     1746  1880
5 Margaret  F     1578  1880
6    Ida    F     1472  1880
7   Alice    F     1414  1880
8  Bertha  F     1320  1880
9   Sarah    F     1288  1880
```

Veri kümesiyle yapmak isteyebileceğiniz birçok şey vardır:

- Zaman içinde belirli bir ismin (kendi adınız veya başka bir adınız) verilen bebeklerin oranını gözünüzde canlandırın
- Bir ismin göreceli sıralamasını belirleyin
- Her yılın en popüler isimlerini veya popülerliği en çok artan veya azalan isimleri belirleyin
- İsimlerdeki eğilimleri analiz edin: ünlüler, ünsüzler, uzunluk, genel çeşitlilik, yazıldığı değişiklikler, ilk ve son harfler
- Trendlerin dış kaynaklarını analiz edin: İncil'deki isimler, ünlüler, demografik değişiklikler

```
In [ ]: !head -n 10 yob1880.txt
```

```
Mary,F,7065
Anna,F,2604
Emma,F,2003
Elizabeth,F,1939
Minnie,F,1746
Margaret,F,1578
Ida,F,1472
Alice,F,1414
Bertha,F,1320
Sarah,F,1288
```

- Bu komutla ilk on veri okunabilir.

- Bu zaten güzel bir şekilde virgülle ayrılmış bir formda olduğundan pandas.read\_csv ile bir DataFrame'e yüklenebilir.

```
In [39]: import pandas as pd
```

```
In [40]: names1880 = pd.read_csv('yob1880.txt',
                             names=['name', 'sex', 'births'])
```

```
In [41]: names1880
```

```
Out[41]:
```

	name	sex	births
0	Mary	F	7065
1	Anna	F	2604
2	Emma	F	2003
3	Elizabeth	F	1939
4	Minnie	F	1746
...	...	...	...
1995	Woodie	M	5
1996	Worthy	M	5
1997	Wright	M	5
1998	York	M	5
1999	Zachariah	M	5

2000 rows × 3 columns

- Bu dosyalar her yıl yalnızca en az beş kez geçen adları içerir; dolayısıyla basitlik adına doğular sütununun cinsiyete göre toplamını o yıldaki toplam doğum sayısı olarak kullanılabilir.

```
In [44]: names1880.groupby('sex').births.sum()
```

```
Out[44]: sex
```

F	90993
M	110493

Name: births, dtype: int64

- Veri seti yıllara göre dosyalara ayrıldığından yapılacak ilk şeylerden biri, tüm verileri tek bir DataFrame'de toplamak ve ayrıca bir yıl alanı eklemektir. Bunu pandas.concat kullanarak yapabilirsiniz.

```
In [45]: years = range(1880, 2011)
```

```
In [46]: pieces = []
columns = ['name', 'sex', 'births']
```

```
In [49]: for year in years:
    path = 'yob%d.txt' % year
    frame = pd.read_csv(path, names=columns)
```

```
In [50]: frame['year'] = year  
pieces.append(frame)
```

```
In [53]: # Her şeyi tek bir DataFrame'de birleştirin  
names = pd.concat(pieces, ignore_index=True)
```

```
In [54]: names
```

Out[13]:

	name	sex	births	year
0	Isabella	F	22731	2010
1	Sophia	F	20477	2010
2	Emma	F	17179	2010
3	Olivia	F	16860	2010
4	Ava	F	15300	2010
...	...	...	...	...
33833	Zymaire	M	5	2010
33834	Zyonne	M	5	2010
33835	Zyquarius	M	5	2010
33836	Zyran	M	5	2010
33837	Zzyzx	M	5	2010

33838 rows × 4 columns

- Burada dikkat edilmesi gereken birkaç şey var. İlk olarak, concat'in DataFrame nesnelerini varsayılan olarak satır bazında birbirine yapıştırduğunu unutmayın. İkinci olarak, ignore\_index=True yapmanız gereklidir çünkü read\_csv'den döndürülen orijinal satır numaralarını korumakla ilgilenmiyoruz. Artık her şeyi içeren çok büyük bir DataFrame vardır.
- Elimizde bulunan bu verilerle, groupby veya pivot\_table kullanarak verileri yıl ve cinsiyet düzeyinde toplayabiliriz.

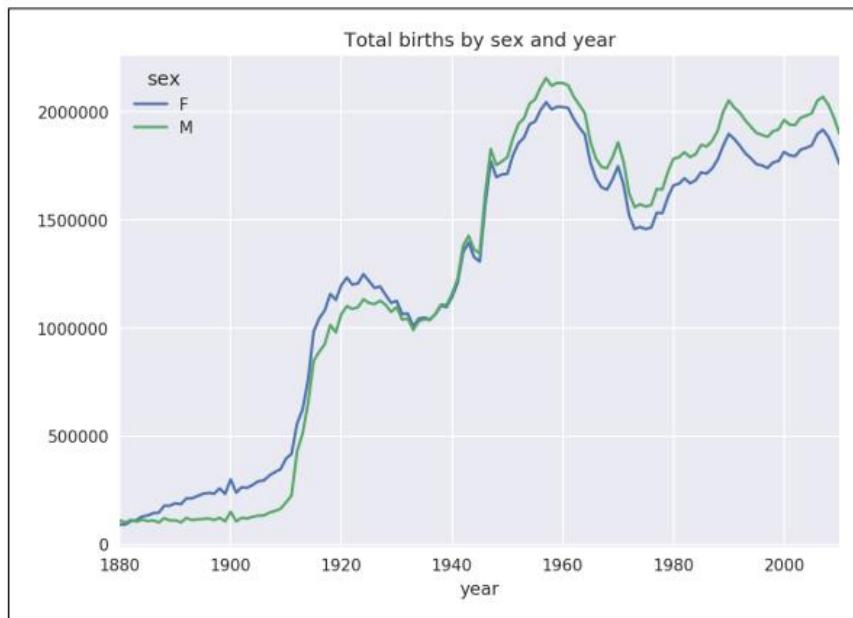
```
In [55]: total_births = names.pivot_table('births', index='year',  
                                         columns='sex', aggfunc='sum')
```

```
In [56]: total_births.tail()
```

Out[15]:

sex	F	M	>z1"
year			
2010	1759010	1898382	

```
In [57]: total_births.plot(title='Total births by sex and year')
```



- Daha sonra, her ismin verildiği bebeklerin toplam doğum sayısına göre oranını gösteren bir sütun desteği ekleyelim. 0,02'lük bir prop değeri, her 100 bebekten 2'sine belirli bir ismin verildiğini gösterir. Böylece verileri yıla ve cinsiyete göre gruplandırıyoruz, ardından her gruba yeni sütunu ekliyoruz.

```
In [17]: def add_prop(group):
    group['prop'] = group.births / group.births.sum()
    return group
names = names.groupby(['year', 'sex']).apply(add_prop)
```

```
In [18]: names
```

```
Out[18]:
```

			name	sex	births	year	prop
year	sex						
2010	F	0	Isabella	F	22731	2010	0.012923
		1	Sophia	F	20477	2010	0.011641
		2	Emma	F	17179	2010	0.009766
		3	Olivia	F	16860	2010	0.009585
		4	Ava	F	15300	2010	0.008698
		...	...	...	...	...	...
	M	33833	Zymaire	M	5	2010	0.000003
		33834	Zyonne	M	5	2010	0.000003
		33835	Zyquarius	M	5	2010	0.000003
		33836	Zyran	M	5	2010	0.000003
		33837	Zzyzx	M	5	2010	0.000003

33838 rows × 5 columns

- Verilerin bir alt kümesi: her cinsiyet/yıl kombinasyonu için en iyi 1000 isim.

```
In [24]: def get_top1000(group):
    return group.sort_values(by='births', ascending=False)[:1000]
grouped = names.groupby(['year', 'sex'])
top1000 = grouped.apply(get_top1000)
# Grup indeksini bırakın, gerekmeyen
top1000.reset_index(inplace=True, drop=True)
```

- Şu şekilde de yapılabilir.

```
In [25]: pieces = []
for year, group in names.groupby(['year', 'sex']):
    pieces.append(group.sort_values(by='births', ascending=False)[:1000])
top1000 = pd.concat(pieces, ignore_index=True)
```

```
In [108]: top1000
Out[108]:
      name  sex  births  year      prop
0      Mary   F     7065  1880  0.077643
1      Anna   F     2604  1880  0.028618
2      Emma   F     2003  1880  0.022013
3  Elizabeth   F     1939  1880  0.021309
4     Minnie   F     1746  1880  0.019188
...
261872    Camilo   M      194  2010  0.000102
261873    Destin   M      194  2010  0.000102
261874    Jaquan   M      194  2010  0.000102
261875    Jaydan   M      194  2010  0.000102
261876    Maxton   M      193  2010  0.000102
[261877 rows x 5 columns]
```

## Adlandırma Trendlerini Analiz Etme

- İlk 1000 ismi erkek ve kız çocuklarına ayırma işlevi:

```
In [ ]: boys = top1000[top1000.sex == 'M']
```

```
In [29]: girls = top1000[top1000.sex == 'F']
```

- Yıl ve isme göre toplam doğum sayısını gösteren pivot tablosu:

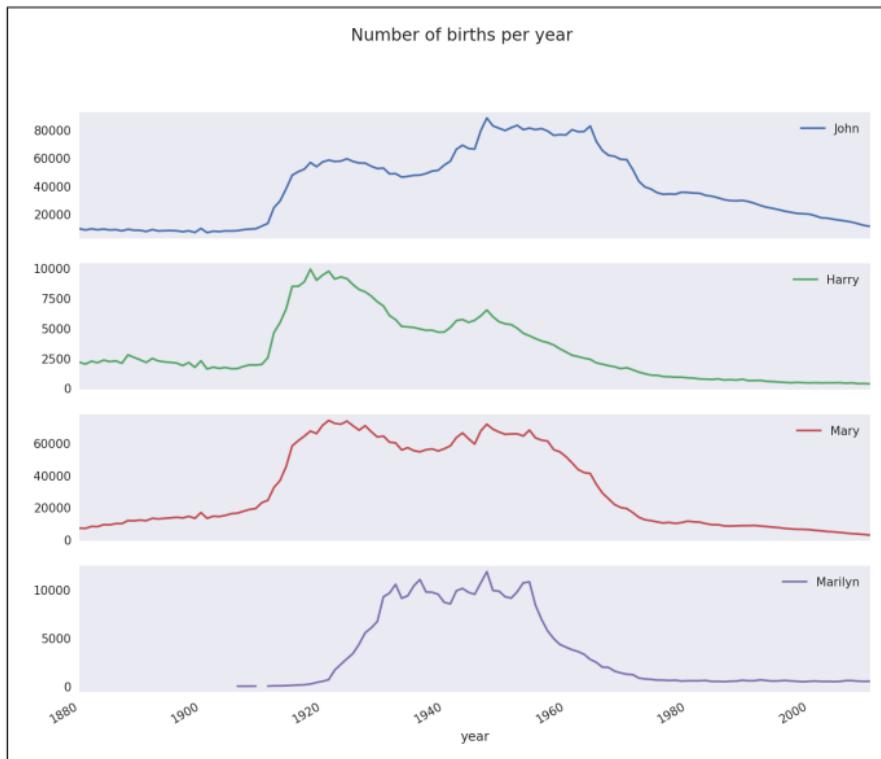
```
In [30]: total_births = top1000.pivot_table('births', index='year',
                                         columns='name',
                                         aggfunc=sum)
```

- Artık bu, DataFrame'in çizim yöntemiyle bazı adlar için çizilebilir.

```
In [ ]: total_births.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 131 entries, 1880 to 2010
Columns: 6868 entries, Aaden to Zuri
dtypes: float64(6868)
memory usage: 6.9 MB
```

```
In [ ]: subset = total_births[['John', 'Harry', 'Mary', 'Marilyn']]
```

```
In [ ]: subset.plot(subplots=True, figsize=(12, 10), grid=False,
                  title="Number of births per year")
```

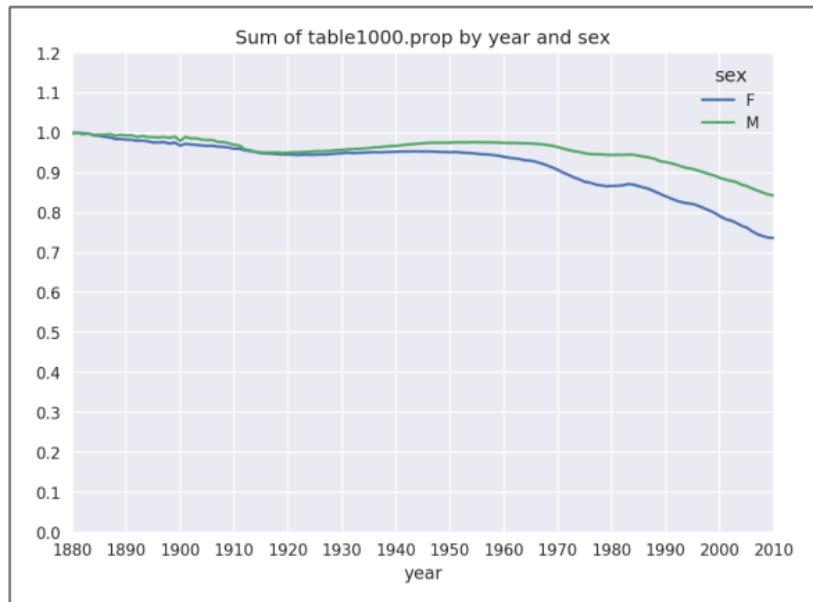


## Adlandırma Çeşitliliğindeki Artışın Ölçülmesi

- Parsellerdeki azalmanın bir açıklaması, daha az sayıda ebeveynin çocukları için ortak isimler seçmesidir. Bu hipotez verilerde araştırılabilir ve doğrulanabilir.
- Ölülerden biri, yıllara ve cinsiyete göre toplanıp planlanan, en popüler 1000 ismin temsil ettiği doğumların oranıdır.

```
In [ ]: table = top1000.pivot_table('prop', index='year',
                                    columns='sex', aggfunc=sum)
```

```
In [ ]: table.plot(title='Sum of table1000.prop by year and sex',
                  yticks=np.linspace(0, 1.2, 13), xticks=range(1880, 2020, 10))
```



- Gerçekten de isim çeşitliliğinin arttığını (ilk 1000'deki toplam oranın azaldığını) görebilirsiniz. Bir başka ilginç ölçüm de doğumların ilk %50'sinde popülerlik sırasına göre en yüksektен en düşüğe doğru alınan farklı isimlerin sayısıdır. Bu sayının hesaplanması biraz daha zordur.

```
In [ ]: df = boys[boys.year == 2010]
```

```
In [ ]: df
```

		name	sex	births	year	prop
260877		Jacob	M	21875	2010	0.011523
260878		Ethan	M	17866	2010	0.009411
260879		Michael	M	17133	2010	0.009025
260880		Jayden	M	17030	2010	0.008971
260881		William	M	16870	2010	0.008887
...		...	...	...	...	...
261872		Camilo	M	194	2010	0.000102
261873		Destin	M	194	2010	0.000102
261874		Jaquan	M	194	2010	0.000102
261875		Jaydan	M	194	2010	0.000102
261876		Maxton	M	193	2010	0.000102

[1000 rows x 5 columns]

- Nesneleri azalan düzende sıraladıktan sonra %50'ye ulaşmak için en popüler isimlerden kaç tanesinin gerektiğini bilmek istiyoruz. Bunu yapmak için bir for döngüsü yazabilirsiniz, ancak vektörleştirilmiş NumPy yöntemi biraz daha akıllıcadır. Prop'un kümülatif toplamını (cumsum) almak ve ardından searchsorted yöntemini çağırırmak, kümülatif toplamındaki, sıralı düzende tutmak için 0,5'in eklenmesi gereken konumu döndürür.

```
In [ ]: prop_cumsum = df.sort_values(by='prop', ascending=False).prop.cumsum()
```

```
In [ ]: prop_cumsum[:10]
```

```
260877    0.011523
260878    0.020934
260879    0.029959
260880    0.038930
260881    0.047817
260882    0.056579
260883    0.065155
260884    0.073414
260885    0.081528
260886    0.089621
Name: prop, dtype: float64
```

```
In [34]: prop_cumsum.values.searchsorted(0.5)
```

```
Out[122]: 116
```

- Artık bu işlemi her yıl/cinsiyet kombinasyonuna uygulayabilir, bu alanları grupperlendirir ve her grup için sayıyı döndüren bir işlev uygulayabilirsiniz.

```
In [ ]: def get_quantile_count(group, q=0.5):
    group = group.sort_values(by='prop', ascending=False)
    return group.prop.cumsum().values.searchsorted(q) + 1
```

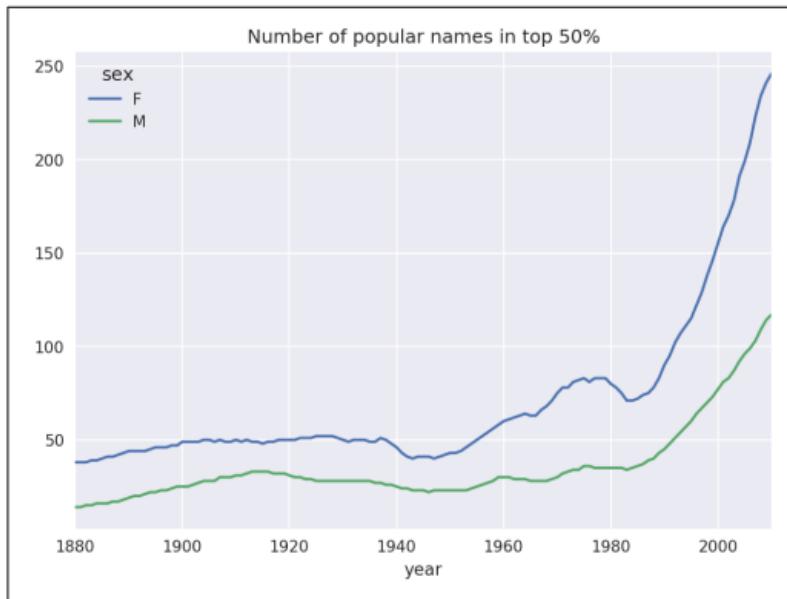
```
In [ ]: diversity = top1000.groupby(['year', 'sex']).apply(get_quantile_count)
diversity = diversity.unstack('sex')
```

- Ortaya çıkan bu DataFrame çeşitliliği artık her cinsiyet için bir tane olmak üzere yıla göre indekslenmiş iki zaman serisine sahiptir.

```
In [35]: diversity.head()
```

	sex	F	M
year			
1880	38	14	
1881	38	14	
1882	38	15	
1883	39	15	
1884	39	16	

```
In [36]: diversity.plot(title="Number of popular names in top 50%)")
```



- Görüdüğünüz gibi kız isimleri her zaman erkek isimlerinden daha çeşitli olmuştur ve zamanla daha da çeşitlenmiştir.

## “Son harf” Devrimi

- 2007 yılında bebek ismi araştırmacısi Laura Wattenberg, web sitesinde erkek isimlerinin son harfe göre dağılımının son 100 yılda önemli ölçüde değiştiğine dikkat çekti. Bunu görmek için öncelikle tam veri kümesindeki tüm doğumları yıla, cinsiyete ve son harfe göre topluyoruz:

```
In [40]: get_last_letter = lambda x: x[-1]
last_letters = names.name.map(get_last_letter)
last_letters.name = 'last_letter'
```

```
In [41]: table = names.pivot_table('births', index=last_letters,
                               columns=['sex', 'year'], aggfunc=sum)
```

```
In [ ]: subtable = table.reindex(columns=[1910, 1960, 2010], level='year')
```

```
In [43]: subtable.head()
```

sex	F			M		
year	1910	1960	2010	1910	1960	2010
last_letter						
a	108376.0	691247.0	670605.0	977.0	5204.0	28438.0
b	NaN	694.0	450.0	411.0	3912.0	38859.0
c	5.0	49.0	946.0	482.0	15476.0	23125.0
d	6750.0	3729.0	2607.0	22111.0	262112.0	44398.0
e	133569.0	435013.0	313833.0	28655.0	178823.0	129012.0

- Daha sonra, her harfle biten her cinsiyet için toplam doğumların oranını içeren yeni bir tablo hesaplamak için tabloyu toplam doğumlara göre normalleştirin:

```
In [44]: subtable.sum()
```

sex	year	
F	1910	396416.0
	1960	2022062.0
	2010	1759010.0
M	1910	194198.0
	1960	2132588.0
	2010	1898382.0

dtype: float64

```
In [ ]: letter_prop = subtable / subtable.sum()
```

```
In [ ]: letter_prop
```

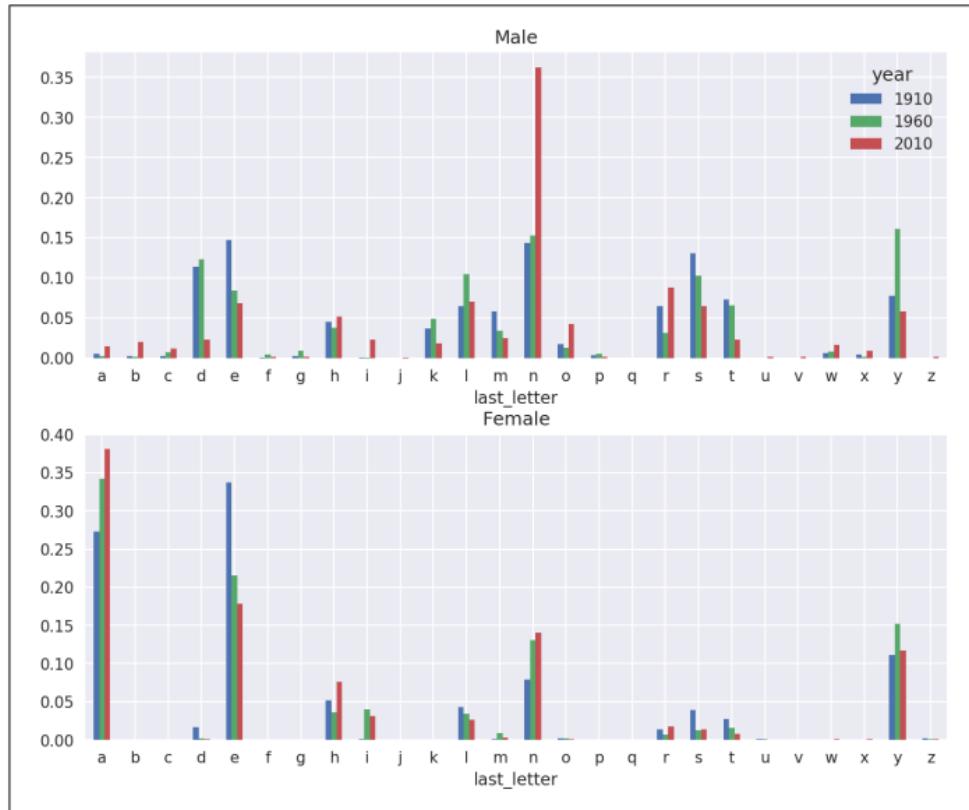
sex	F			M		
	1910	1960	2010	1910	1960	2010
year						
last_letter						
a	0.273390	0.341853	0.381240	0.005031	0.002440	0.014980
b	NaN	0.000343	0.000256	0.002116	0.001834	0.020470
c	0.000013	0.000024	0.000538	0.002482	0.007257	0.012181
d	0.017028	0.001844	0.001482	0.113858	0.122908	0.023387
e	0.336941	0.215133	0.178415	0.147556	0.083853	0.067959
...	...	...	...	...	...	...
v	NaN	0.000060	0.000117	0.000113	0.000037	0.001434
w	0.000020	0.000031	0.001182	0.006329	0.007711	0.016148
x	0.000015	0.000037	0.000727	0.003965	0.001851	0.008614
y	0.110972	0.152569	0.116828	0.077349	0.160987	0.058168
z	0.002439	0.000659	0.000704	0.000170	0.000184	0.001831

[26 rows x 6 columns]

- Elimizdeki harf oranlarıyla, her cinsiyet için yıllara göre ayrılmış çubuk grafikleri yapılabilir.

```
In [ ]: import matplotlib.pyplot as plt
```

```
In [ ]: fig, axes = plt.subplots(2, 1, figsize=(10, 8))
letter_prop['M'].plot(kind='bar', rot=0, ax=axes[0], title='Male')
letter_prop['F'].plot(kind='bar', rot=0, ax=axes[1], title='Female',
                      legend=False)
```



- Görülen gibi sonu n ile biten erkek isimleri 1960'lardan bu yana önemli bir büyümeye kaydetti. Daha önce oluşturulan tam tabloya dönerek yıllara ve cinsiyete göre yeniden normalleştirme yapılır ve erkek isimleri için bir harf alt kümesi seçin, son olarak her sütunu bir zaman serisi haline getirecek şekilde yer değiştirelim:

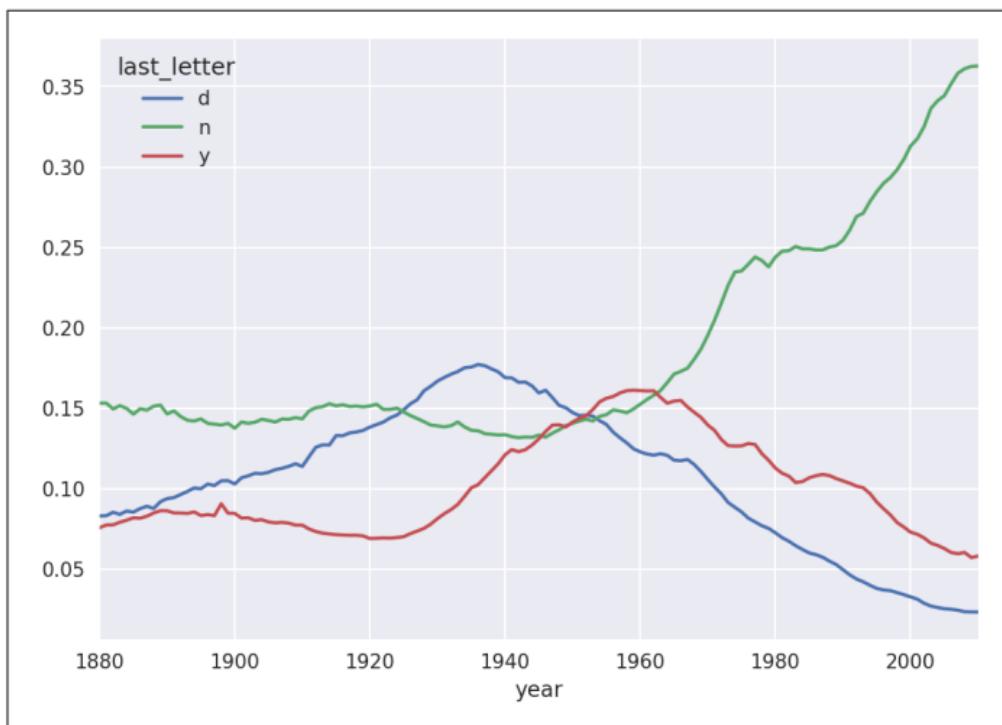
```
In [ ]: letter_prop = table / table.sum()
```

```
In [ ]: dny_ts = letter_prop.loc[['d', 'n', 'y'], 'M'].T
```

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

last_letter	d	n	y
year			
1880	0.083055	0.153213	0.075760
1881	0.083247	0.153214	0.077451
1882	0.085340	0.149560	0.077537
1883	0.084066	0.151646	0.079144
1884	0.086120	0.149915	0.080405

```
In [ ]: dny_ts.plot()
```



## Kız İsmi Haline Gelen Erkek İsimleri (ve Tersi)

- Bir örnek Lesley veya Leslie adıdır. Top1000 DataFrame'e dönerek veri setinde yer alan isimlerin "lesl" ile başlayan bir listesini hesaplanır.

```
In [ ]: all_names = pd.Series(top1000.name.unique())
```

```
In [ ]: lesley_like = all_names[all_names.str.lower().str.contains('lesl')]
```

```
In [45]: lesley_like
```

```
632    Leslie
2294   Lesley
4262   Leslee
4728    Lesli
6103   Lesly
dtype: object
```

- Buradan, göreli sıklıkları görmek için yalnızca adlara göre gruplandırılmış adlara ve toplam doğumlara filtre uygulanabilir.

```
In [ ]: filtered = top1000[top1000.name.isin(lesley_like)]
```

```
In [ ]: filtered.groupby('name').births.sum()
```

	name	
Leslee	1082	
Lesley	35022	
Lesli	929	
Leslie	370429	
Lesly	10067	

- Daha sonra cinsiyete ve yıla göre toplayalım ve yıl içinde normalleştirelim:

```
In [ ]: table = filtered.pivot_table('births', index='year',
                                     columns='sex', aggfunc='sum')
```

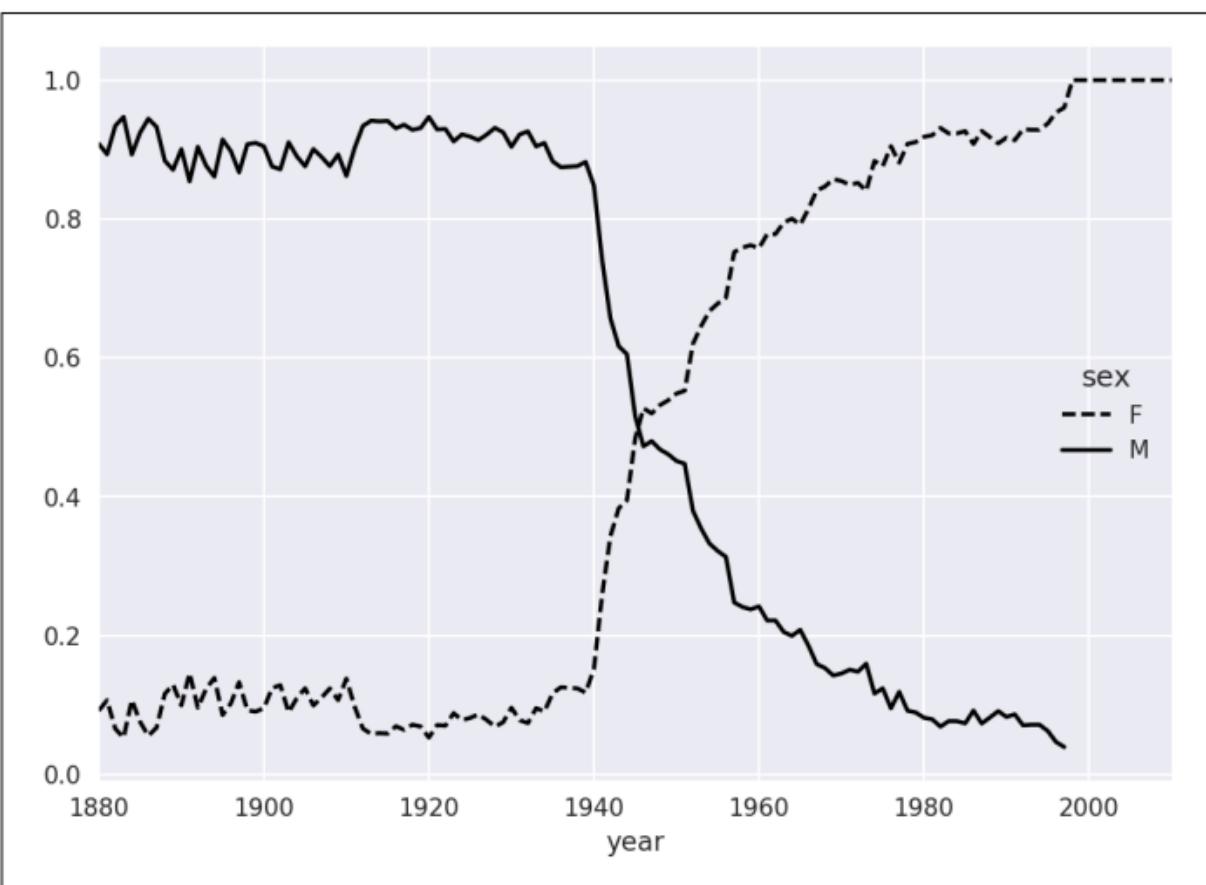
```
In [ ]: table = table.div(table.sum(1), axis=0)
```

```
In [ ]: table.tail()
```

sex	F	M
year		
2006	1.0	NaN
2007	1.0	NaN
2008	1.0	NaN
2009	1.0	NaN
2010	1.0	NaN

- Son olarak, zaman içinde cinsiyete göre dağılımın grafiği:

```
In [46]: table.plot(style={'M': 'k-', 'F': 'k--'})
```



## USDA Gıda Veritabanı

- ABD Tarım Bakanlığı gıda besin değerlerine ilişkin bir veri tabanı sunmaktadır. Programcı Ashley Williams bu veri tabanının JSON formatındaki bir versiyonunu kullanıma sundu.

```
In [51]: import json  
  
In [52]: db = json.load(open('database.json'))  
  
In [53]: len(db)  
Out[53]: 6636
```

- Db'deki her giriş, tek bir yiyeceğe ait tüm verileri içeren bir dict'tir. 'Besinler' alanı, her besin için bir tane olmak üzere bir sözlük listesidir.

```
In [54]: db[0].keys()  
Out[54]: dict_keys(['id', 'description', 'tags', 'manufacturer', 'group', 'portions',  
 'nutrients'])  
  
In [55]: db[0]['nutrients'][0]  
Out[55]: {'value': 25.18,  
          'units': 'g',  
          'description': 'Protein',  
          'group': 'Composition'}  
  
In [56]: nutrients = pd.DataFrame(db[0]['nutrients'])  
  
In [57]: nutrients[:7]  
Out[57]:
```

	value	units	description	group
0	25.18	g	Protein	Composition
1	29.20	g	Total lipid (fat)	Composition
2	3.06	g	Carbohydrate, by difference	Composition
3	3.28	g	Ash	Other
4	376.00	kcal	Energy	Energy
5	39.28	g	Water	Composition
6	1573.00	kJ	Energy	Energy

- Bir dict listesini DataFrame'e dönüştürürken, çıkarılacak alanların bir listesi belirtilir. Yiyecek adlarını, grubunu, kimliği ve üreticisi alınır.

```
In [58]: info_keys = ['description', 'group', 'id', 'manufacturer']

In [59]: info = pd.DataFrame(db, columns=info_keys)

In [60]: info[:5]

Out[60]:
```

	description	group	id	manufacturer
0	Cheese, caraway	Dairy and Egg Products	1008	
1	Cheese, cheddar	Dairy and Egg Products	1009	
2	Cheese, edam	Dairy and Egg Products	1018	
3	Cheese, feta	Dairy and Egg Products	1019	
4	Cheese, mozzarella, part skim milk	Dairy and Egg Products	1028	

```
In [61]: info.info()

<class 'pandas.core.frame.DataFrame'\>
RangeIndex: 6636 entries, 0 to 6635
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   description  6636 non-null   object 
 1   group        6636 non-null   object 
 2   id           6636 non-null   int64  
 3   manufacturer 5195 non-null   object 
dtypes: int64(1), object(3)
memory usage: 207.5+ KB
```

- Besin gruplarının dağılımını value\_counts ile görebilirsiniz:

```
In [62]: pd.value_counts(info.group)[:10]

Out[62]: group
Vegetables and Vegetable Products    812
Beef Products                         618
Baked Products                        496
Breakfast Cereals                     403
Legumes and Legume Products          365
Fast Foods                            365
Lamb, Veal, and Game Products        345
Sweets                                341
Fruits and Fruit Juices              328
Pork Products                          328
Name: count, dtype: int64
```

- Şimdi, tüm besin verileri üzerinde bazı analizler yapmak için, her bir gıdanın besin öğelerini tek bir büyük tabloda bir araya getirmek en kolay yoldur. Bunu yapmak için birkaç adım atılması gereklidir. İlk olarak, her bir gıda besin maddesi listesi bir DataFrame'e dönüştürülür, gıda kimliği için bir sütun eklenir ve DataFrame bir listeye eklenir. Daha sonra bunlar concat ile birleştirilebilir.

```
In [63]: nutrients
```

```
Out[63]:
```

	value	units	description	group
0	25.180	g	Protein	Composition
1	29.200	g	Total lipid (fat)	Composition
2	3.060	g	Carbohydrate, by difference	Composition
3	3.280	g	Ash	Other
4	376.000	kcal	Energy	Energy
...	...	...	...	...
157	1.472	g	Serine	Amino Acids
158	93.000	mg	Cholesterol	Other
159	18.584	g	Fatty acids, total saturated	Other
160	8.275	g	Fatty acids, total monounsaturated	Other
161	0.830	g	Fatty acids, total polyunsaturated	Other

162 rows × 4 columns

- Bu DataFrame'de kopyalar vardır. Bunları bırakmak işleri kolaylaştırır.
- 'Grup' ve 'açıklama' her iki DataFrame nesnesinde de olduğundan, netlik sağlamak amacıyla yeniden adlandırılabilir.

```
In [64]: nutrients.duplicated().sum() # kopya sayısı
```

```
Out[64]: 108
```

```
In [65]: nutrients = nutrients.drop_duplicates()
```

```
In [66]: col_mapping = {'description' : 'food',
                     'group' : 'fgroup'}
```

```
In [67]: info = info.rename(columns=col_mapping, copy=False)
```

```
In [68]: info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6636 entries, 0 to 6635
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          --          --      
 0   food        6636 non-null   object 
 1   fgroup      6636 non-null   object 
 2   id          6636 non-null   int64  
 3   manufacturer 5195 non-null   object 
dtypes: int64(1), object(3)
memory usage: 207.5+ KB
```

```
In [69]: col_mapping = {'description' : 'nutrient',
                      'group' : 'nutgroup'}
```

```
In [70]: nutrients = nutrients.rename(columns=col_mapping, copy=False)
```

```
In [71]: nutrients
```

Out[71]:

	value	units	nutrient	nutgroup
0	25.180	g	Protein	Composition
1	29.200	g	Total lipid (fat)	Composition
2	3.060	g	Carbohydrate, by difference	Composition
3	3.280	g	Ash	Other
4	376.000	kcal	Energy	Energy
5	39.280	g	Water	Composition
6	1573.000	kJ	Energy	Energy
7	0.000	g	Fiber, total dietary	Composition
8	673.000	mg	Calcium, Ca	Elements
9	0.640	mg	Iron, Fe	Elements
10	22.000	mg	Magnesium, Mg	Elements
11	490.000	mg	Phosphorus, P	Elements
12	93.000	mg	Potassium, K	Elements
13	690.000	mg	Sodium, Na	Elements
14	2.940	mg	Zinc, Zn	Elements
15	0.024	mg	Copper, Cu	Elements
16	0.021	mg	Manganese, Mn	Elements
17	14.500	mcg	Selenium, Se	Elements
18	1054.000	IU	Vitamin A, IU	Vitamins
19	262.000	mcg	Retinol	Vitamins
20	271.000	mcg_RAE	Vitamin A, RAE	Vitamins
21	0.000	mg	Vitamin C, total ascorbic acid	Vitamins
22	0.031	mg	Thiamin	Vitamins
23	0.450	mg	Riboflavin	Vitamins
24	0.180	mg	Niacin	Vitamins
25	0.190	mg	Pantothenic acid	Vitamins

25	0.190	mg	Pantothenic acid	Vitamins
26	0.074	mg	Vitamin B-6	Vitamins
27	18.000	mcg	Folate, total	Vitamins
28	0.270	mcg	Vitamin B-12	Vitamins
29	0.000	mcg	Folic acid	Vitamins
30	18.000	mcg	Folate, food	Vitamins
31	18.000	mcg_DFE	Folate, DFE	Vitamins
32	93.000	mg	Cholesterol	Other
33	18.584	g	Fatty acids, total saturated	Other
34	8.275	g	Fatty acids, total monounsaturated	Other
35	0.830	g	Fatty acids, total polyunsaturated	Other
36	0.324	g	Tryptophan	Amino Acids
37	0.896	g	Threonine	Amino Acids
38	1.563	g	Isoleucine	Amino Acids
39	2.412	g	Leucine	Amino Acids
40	2.095	g	Lysine	Amino Acids
41	0.659	g	Methionine	Amino Acids
42	0.126	g	Cystine	Amino Acids
43	1.326	g	Phenylalanine	Amino Acids
44	1.216	g	Tyrosine	Amino Acids
45	1.682	g	Valine	Amino Acids
46	0.952	g	Arginine	Amino Acids
47	0.884	g	Histidine	Amino Acids
48	0.711	g	Alanine	Amino Acids
49	1.618	g	Aspartic acid	Amino Acids
50	6.160	g	Glutamic acid	Amino Acids
51	0.439	g	Glycine	Amino Acids
52	2.838	g	Proline	Amino Acids
53	1.472	g	Serine	Amino Acids

- Bilginin besinlerle birleştirilmesi:

```
In [ ]: ndata = pd.merge(nutrients, info, on='id', how='outer')
```

```
In [ ]: ndata.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 375176 entries, 0 to 375175
Data columns (total 8 columns):
nutrient      375176 non-null object
nutgroup       375176 non-null object
units          375176 non-null object
value          375176 non-null float64
id             375176 non-null int64
food           375176 non-null object
fgroup         375176 non-null object
manufacturer   293054 non-null object
dtypes: float64(1), int64(1), object(6)
memory usage: 25.8+ MB

```

In [ ]: `ndata.iloc[30000]`

```

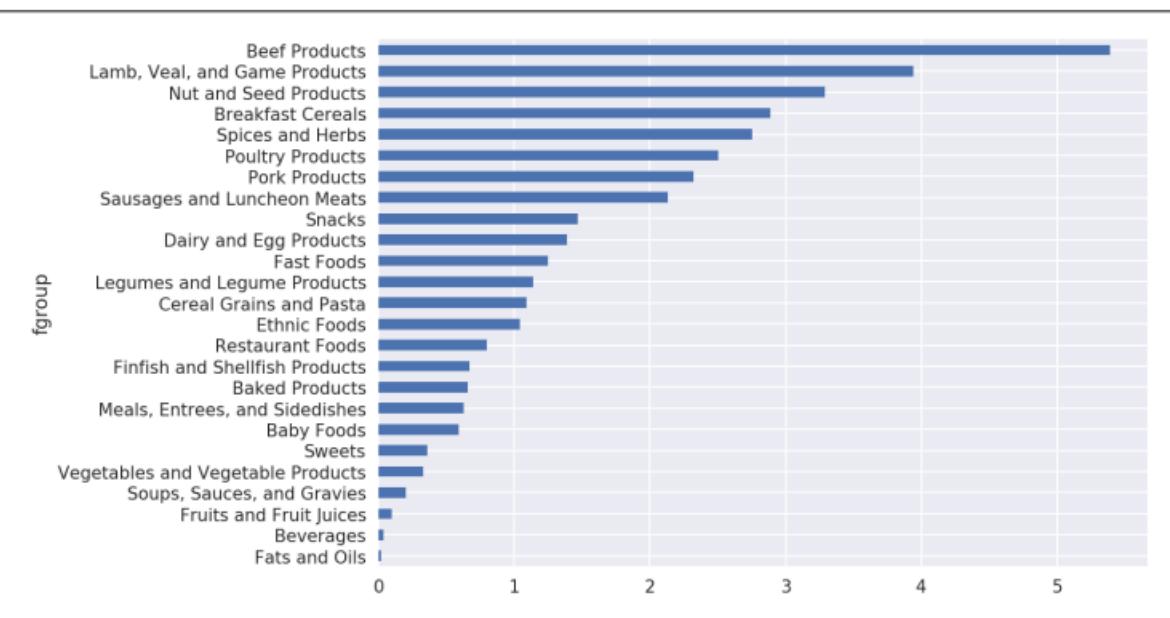
nutrient          Glycine
nutgroup         Amino Acids
units            g
value            0.04
id               6158
food             Soup, tomato bisque, canned, condensed
fgroup           Soups, Sauces, and Gravies
manufacturer
Name: 30000, dtype: object

```

- Besin grubuna ve besin türüne göre medyan değerlerin bir grafiği oluşturulabilir.

In [ ]: `result = ndata.groupby(['nutrient', 'fgroup'])['value'].quantile(0.5)`

In [ ]: `result['Zinc', 'Zn'].sort_values().plot(kind='barh')`



- Her bir besin maddesinin hangi gıdada en yoğun olduğunu bulunabilir.

```
In [79]: by_nutrient = ndata.groupby(['nutgroup', 'nutrient'])

get_maximum = lambda x: x.loc[x.value.idxmax()]
get_minimum = lambda x: x.loc[x.value.idxmin()]

max_foods = by_nutrient.apply(get_maximum)[['value', 'food']]

# yemeği biraz daha küçük yapın
max_foods.food = max_foods.food.str[:50]
```

- Ortaya çıkan DataFrame, kitapta görüntülenemeyecek kadar büyük; işte sadece 'Amino Asitler' besin grubu:

```
In [80]: max_foods.loc['Amino Acids']['food']

      nutrient
Alanine          Gelatins, dry powder, unsweetened
Arginine         Seeds, sesame flour, low-fat
Aspartic acid    Soy protein isolate
Cystine          Seeds, cottonseed flour, low fat (glandless)
Glutamic acid   Soy protein isolate
...
Serine           Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Threonine        Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Tryptophan       Sea lion, Steller, meat with fat (Alaska Native)
Tyrosine         Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Valine           Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
Name: food, Length: 19, dtype: object
```

## 2012 Federal Seçim Komisyonu Veritabanı

- ABD Federal Seçim Komisyonu, siyasi kampanyalara yapılan katkılara ilişkin verileri yayıyor. Bu, katkıda bulunanların adlarını, mesleğini ve işverenini, adresini ve katkı tutarını içerir. İlginç bir veri seti 2012 ABD başkanlık seçimlerinden alınmıştır.

```
In [81]: fec = pd.read_csv('P00000001-ALL.csv')
```

```
In [82]: fec.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1001731 entries, 0 to 1001730
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   cmte_id           1001731 non-null   object 
 1   cand_id           342 non-null     object 
 2   cand_nm           343 non-null     object 
 3   contbr_nm         309 non-null     object 
 4   contbr_city       311 non-null     object 
 5   contbr_st          260 non-null     object 
 6   contbr_zip         262 non-null     object 
 7   contbr_employer    262 non-null     object 
 8   contbr_occupation  162 non-null     object 
 9   contb_receipt_amt  18 non-null     object 
 10  contb_receipt_dt   13 non-null     object 
 11  receipt_desc       12 non-null     object 
 12  memo_cd            6 non-null     object 
 13  memo_text          6 non-null     object 
 14  form_tp             0 non-null     float64
 15  file_num;;;;;;     0 non-null     float64
dtypes: float64(2), object(14)
memory usage: 122.3+ MB
```

```
In [83]: fec.iloc[123456]
Out[83]: cmte_id          C00431445,"P80003338","Obama, Barack","ELLMAN, ...
cand_id
cand_nm
contbr_nm
contbr_city
contbr_st
contbr_zip
contbr_employer
contbr_occupation
contb_receipt_amt
contb_receipt_dt
receipt_desc
memo_cd
memo_text
form_tp
file_num;;;
Name: 123456, dtype: object
```

Verilerde herhangi bir siyasi parti bağlantısının bulunmadığını görüyorsunuz, bunu eklemekte fayda var. Unique kullanarak tüm benzersiz siyasi adayların bir listesini alabilirsiniz.

```
In [88]: unique_cands = fec.cand_nm.unique()
In [89]: unique_cands
array(['Bachmann, Michelle', 'Romney, Mitt', 'Obama, Barack',
       'Roemer, Charles E. 'Buddy' III', 'Pawlenty, Timothy',
       'Johnson, Gary Earl', 'Paul, Ron', 'Santorum, Rick', 'Cain, Herman',
       'Gingrich, Newt', 'McCotter, Thaddeus G', 'Huntsman, Jon',
       'Perry, Rick'], dtype=object)
In [90]: unique_cands[2]
Out[189]: 'Obama, Barack'
```

- Parti üyeliğini belirtmenin bir yolu dict kullanmaktadır.

```
In [91]: parties = {'Bachmann, Michelle': 'Republican',
                 'Cain, Herman': 'Republican',
                 'Gingrich, Newt': 'Republican',
                 'Huntsman, Jon': 'Republican',
                 'Johnson, Gary Earl': 'Republican',
                 'McCotter, Thaddeus G': 'Republican',
                 'Obama, Barack': 'Democrat',
                 'Paul, Ron': 'Republican',
                 'Pawlenty, Timothy': 'Republican',
                 'Perry, Rick': 'Republican',
                 "Roemer, Charles E. 'Buddy' III": 'Republican',
                 'Romney, Mitt': 'Republican',
                 'Santorum, Rick': 'Republican'}
```

## Mesleğe ve İşverene Göre Bağış İstatistikleri

- Mesleğe göre bağışlar da sıklıkla incelenen bir istatistik. Örneğin, avukatlar (avukatlar) Demokratlara daha fazla para bağışlama eğilimindeyken şirket yöneticileri daha fazla para bağışlama eğilimindedir.

```
In [94]: fec.contbr_occupation.value_counts()[:10]
```

```
RETIRED                    233990
INFORMATION REQUESTED      35107
ATTORNEY                   34286
HOMEMAKER                  29931
PHYSICIAN                  23432
INFORMATION REQUESTED PER BEST EFFORTS 21138
ENGINEER                     14334
TEACHER                      13990
CONSULTANT                   13273
PROFESSOR                   12555
Name: contbr_occupation, dtype: int64
```

- Mesleklerde baktığınızda çoğunun aynı temel iş türüne atıfta bulunduğu veya aynı şeyin birkaç çeşidi olduğunu fark edeceksiniz. Aşağıdaki kod parçasığı, bir mesleği diğerine eşleyerek birkaçını temizleme tekniğini göstermektedir; Eşlemesi olmayan mesleklerde izin vermek için dict.get kullanmanın "püf noktasına" dikkat edin.

```
In [95]: occ_mapping = {
    'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
    'INFORMATION REQUESTED' : 'NOT PROVIDED',
    'INFORMATION REQUESTED (BEST EFFORTS)' : 'NOT PROVIDED',
    'C.E.O.' : 'CEO'
}
```

```
In [97]: # Eşleme sağlanmadıysa x değerini döndür
f = lambda x: occ_mapping.get(x, x)
fec.contbr_occupation = fec.contbr_occupation.map(f)
```

- Aynı şey işverenler için de yapılabilir.

```
In [98]: emp_mapping = {
    'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
    'INFORMATION REQUESTED' : 'NOT PROVIDED',
    'SELF' : 'SELF-EMPLOYED',
    'SELF EMPLOYED' : 'SELF-EMPLOYED',
}
```

```
In [99]: # Eşleme sağlanmadıysa x değerini döndür
f = lambda x: emp_mapping.get(x, x)
fec.contbr_employer = fec.contbr_employer.map(f)
```

- Artık verileri partiye ve mesleğe göre toplamak için pivot\_table'ı kullanabilir, ardından toplamda en az 2 milyon dolar bağış yapan alt kümeye filtre uygulayabilirsiniz:

```
In [ ]: by_occupation = fec.pivot_table('contb_receipt_amt',
                                         index='contbr_occupation',
                                         columns='party', aggfunc='sum')
```

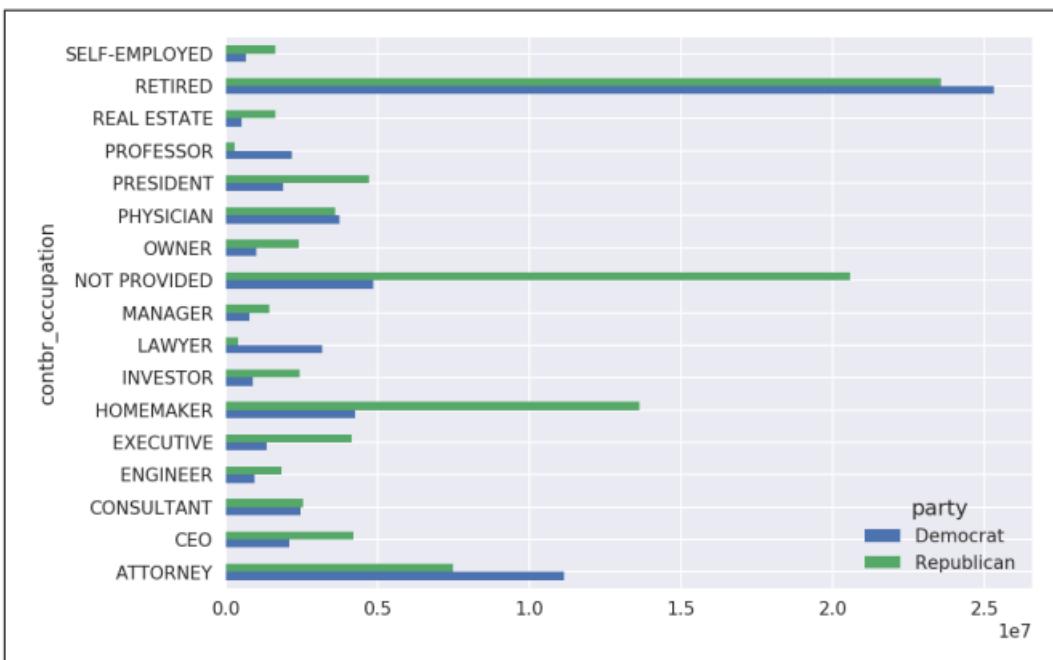
```
In [ ]: over_2mm = by_occupation[by_occupation.sum(1) > 2000000]
```

```
In [ ]: over_2mm
```

party	Democrat	Republican
contbr_occupation		
ATTORNEY	11141982.97	7.477194e+06
CEO	2074974.79	4.211041e+06
CONSULTANT	2459912.71	2.544725e+06
ENGINEER	951525.55	1.818374e+06
EXECUTIVE	1355161.05	4.138850e+06
...	...	...
PRESIDENT	1878509.95	4.720924e+06
PROFESSOR	2165071.08	2.967027e+05
REAL ESTATE	528902.09	1.625902e+06
RETIRED	25305116.38	2.356124e+07
SELF-EMPLOYED	672393.40	1.640253e+06

[17 rows x 2 columns]

```
In [ ]: over_2mm.plot(kind='barh')
```



- En iyi bağışçı meslekleri veya Obama ve Romney'e bağış yapan en iyi şirketler ilginizi çekebilir. Bunu yapmak için aday adına göre gruplandırılabilir ve bir değişken kullanabilirsiniz.

```
In [103]: def get_top_amounts(group, key, n=5):
    totals = group.groupby(key)['contb_receipt_amt'].sum()
    return totals.nlargest(n)
```

```
In [ ]: grouped = fec_mrbo.groupby('cand_nm')
```

```
In [ ]: grouped.apply(get_top_amounts, 'contbr_occupation', n=7)
```

cand_nm	contbr_occupation	
Obama, Barack	RETIRED	25305116.38
	ATTORNEY	11141982.97
	INFORMATION REQUESTED	4866973.96
	HOMEMAKER	4248875.80
	PHYSICIAN	3735124.94
		...
Romney, Mitt	HOMEMAKER	8147446.22
	ATTORNEY	5364718.82
	PRESIDENT	2491244.89
	EXECUTIVE	2300947.03
	C.E.O.	1968386.11
		Name: contb_receipt_amt, Length: 14, dtype: float64

```
In [105]: grouped.apply(get_top_amounts, 'contbr_employer', n=10)
```

cand_nm	contbr_employer	
Obama, Barack	RETIRED	22694358.85
	SELF-EMPLOYED	17080985.96
	NOT EMPLOYED	8586308.70
	INFORMATION REQUESTED	5053480.37
	HOMEMAKER	2605408.54
		...
Romney, Mitt	CREDIT SUISSE	281150.00
	MORGAN STANLEY	267266.00
	GOLDMAN SACH & CO.	238250.00
	BARCLAYS CAPITAL	162750.00
	H.I.G. CAPITAL	139500.00
		Name: contb_receipt_amt, Length: 20, dtype: float64

## Bağış Tutarlarının Gruplandırılması

- Bu verileri analiz etmenin yararlı bir yolu, katkıda bulunanların miktarlarını katkı boyutuna göre gruptara ayırmak için kesme işlevini kullanmaktadır.

```
In [108]: bins = np.array([0, 1, 10, 100, 1000, 10000,
                      100000, 1000000, 10000000])
```

```
In [ ]: labels = pd.cut(fec_mrbo.contb_receipt_amt, bins)
```

```
In [ ]: labels
```

```

411      (10, 100]
412      (100, 1000]
413      (1000, 10000]
414      (10, 100]
415      (10, 100]

...
701381      (10, 100]
701382      (100, 1000]
701383      (1, 10]
701384      (10, 100]
701385      (100, 1000]
Name: contb_receipt_amt, Length: 694282, dtype: category
Categories (8, interval[int64]): [(0, 1] < (1, 10] < (10, 100] < (100, 1000] < (1
000, 10000] < (10000, 100000] < (100000, 1000000] < (1000000,
10000000]]

```

- Daha sonra bağış boyutuna göre bir histogram elde etmek için Obama ve Romney'e ait verileri ad ve kutu etiketine göre gruplayabiliriz.

```
In [ ]: grouped = fec_mrbo.groupby(['cand_nm', labels])
```

```
In [ ]: grouped.size().unstack(0)
```

cand_nm	Obama	Barack	Romney	Mitt
contb_receipt_amt				
(0, 1]	493.0		77.0	
(1, 10]	40070.0		3681.0	
(10, 100]	372280.0		31853.0	
(100, 1000]	153991.0		43357.0	
(1000, 10000]	22284.0		26186.0	
(10000, 100000]	2.0		1.0	
(100000, 1000000]	3.0		Nan	
(1000000, 10000000]	4.0		Nan	

- Bu veriler Obama'nın Romney'den çok daha fazla sayıda küçük bağış aldığıını gösteriyor. Ayrıca adaylara göre her büyüklükteki toplam bağışların yüzdesini görselleştirmek için katkı tutarlarını toplayabilir ve gruplar halinde normalleştirebilirsiniz.

```
In [ ]: bucket_sums = grouped.contb_receipt_amt.sum().unstack(0)
```

```
In [ ]: normed_sums = bucket_sums.div(bucket_sums.sum(axis=1), axis=0)
```

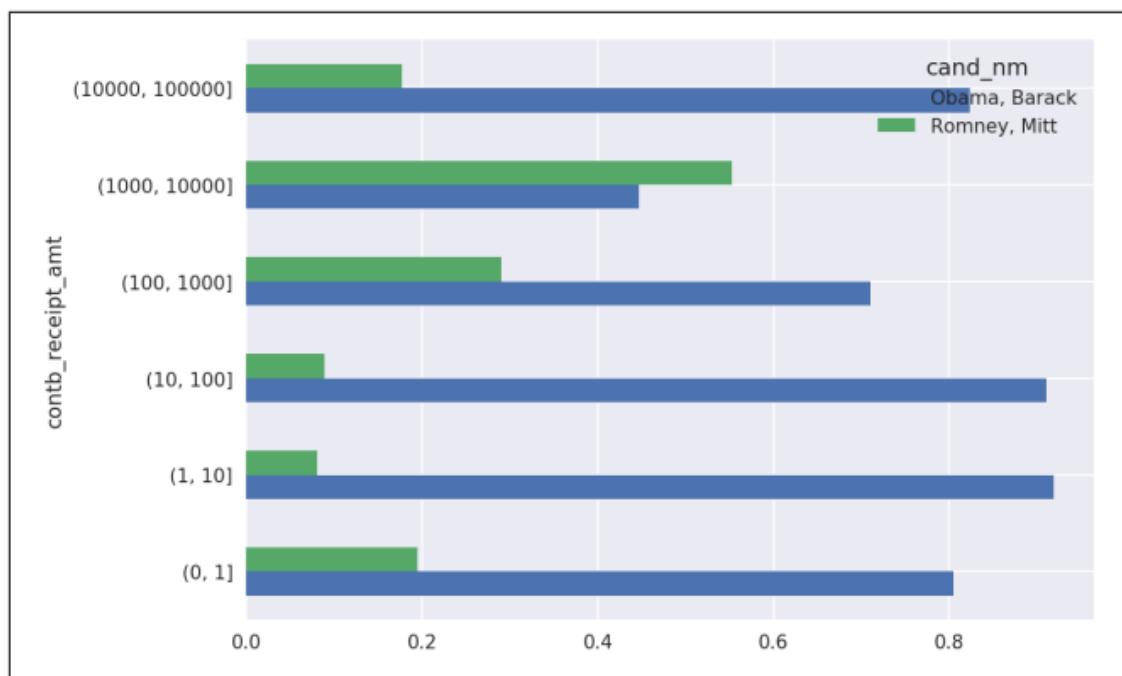
```
In [ ]: normed_sums
```

```

cand_nm          Obama, Barack  Romney, Mitt
contb_receipt_amt
(0, 1]           0.805182    0.194818
(1, 10]          0.918767    0.081233
(10, 100]         0.910769    0.089231
(100, 1000]       0.710176    0.289824
(1000, 10000]     0.447326    0.552674
(10000, 100000]   0.823120    0.176880
(100000, 1000000] 1.000000    NaN
(1000000, 10000000] 1.000000    NaN

```

```
In [ ]: normed_sums[:2].plot(kind='barh')
```



## Eyaletlere Göre Bağış İstatistikleri

- Verilerin adaya ve eyalete göre toplanması rutin bir iştir.

```
In [ ]: grouped = fec_mrbo.groupby(['cand_nm', 'contbr_st'])
```

```
In [ ]: totals = grouped.contb_receipt_amt.sum().unstack(0).fillna(0)
```

```
In [ ]: totals = totals[totals.sum(1) > 100000]
```

```
In [ ]: totals[:10]
```

```

cand_nm    Obama, Barack  Romney, Mitt
contbr_st
AK        281840.15    86204.24
AL        543123.48   527303.51
AR        359247.28   105556.00
AZ        1506476.98  1888436.23
CA        23824984.24 11237636.60
CO        2132429.49  1506714.12
CT        2068291.26  3499475.45
DC        4373538.80  1025137.50
DE        336669.14   82712.00
FL        7318178.58  8338458.81

```

- Her satırı toplam katkı tutarına bölerseniz, her aday için eyalete göre toplam bağışların göreceli yüzdesini elde edersiniz:

```
In [ ]: percent = totals.div(totals.sum(1), axis=0)
```

```
In [ ]: percent[:10]
```

```

cand_nm    Obama, Barack  Romney, Mitt
contbr_st
AK        0.765778    0.234222
AL        0.507390    0.492610
AR        0.772902    0.227098
AZ        0.443745    0.556255
CA        0.679498    0.320502
CO        0.585970    0.414030
CT        0.371476    0.628524
DC        0.810113    0.189887
DE        0.802776    0.197224
FL        0.467417    0.532583

```

## EK A | Gelişmiş NumPy

### A.1 ndarray Nesne Dahili Öğeleri

- NumPy ndarray, homojen bir veri bloğunu (bitişik veya adım adım) çok boyutlu bir dizi nesnesi olarak yorumlamak için bir araç sağlar. Veri türü veya dtype, verilerin float, integer, boolean veya baktığımız diğer türlerden herhangi biri olarak nasıl yorumlanacağını belirler.
- Ndarray'i esnek kılan şeylerden biri de her dizi nesnesinin bir veri bloğu üzerinde aşamalı bir görünüm olmasıdır. Örneğin dizi görünümü arr[::-2, ::-1]'in nasıl herhangi bir veriyi kopyalamadığını merak edebilirsiniz. Bunun nedeni, ndarray'in yalnızca bir bellek yiğini ve bir tür tipinden daha fazlası olmasıdır; aynı zamanda dizinin değişen adım boyutlarıyla bellekte hareket etmesini sağlayan "adım adım" bilgisine de sahiptir. Daha doğrusu, ndarray dahili olarak aşağıdakilerden oluşur:

- Verilere yönelik bir işaretçi; yani RAM'deki veya bellek eşlemeli dosyadaki bir veri bloğu
- Dizideki sabit boyutlu değer hücrelerini açıklayan veri türü veya dtype
- Dizinin şeklini gösteren bir demet (shape)
- Bir boyut boyunca bir öğeyi ilerletmek için "adım" atılacak bayt sayısını belirten tamsayılardan oluşan bir dizi adım

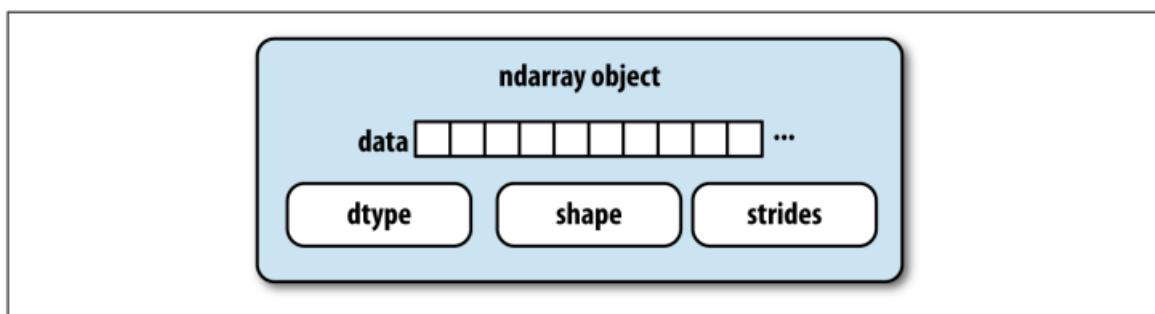
- Örneğin,  $10 \times 5$ 'lik bir dizi  $(10, 5)$  şeklinde olacaktır:

```
In [4]: np.ones((10, 5)).shape
Out[4]: (10, 5)
```

- Tipik (C düzeni)  $3 \times 4 \times 5$  float64 (8 bayt) değer dizisi, adımlara  $(160, 40, 8)$  sahiptir.

```
In [5]: np.ones((3, 4, 5), dtype=np.float64).strides
Out[5]: (160, 40, 8)
```

- Tipik bir NumPy kullanıcısının dizi adımlarıyla ilgilenmesi nadir olsa da bunlar "sıfır kopya" dizi görüntülerinin oluşturulmasında kritik bileşendir. Adımlar negatif bile olabilir, bu da bir dizinin bellekte "geriye doğru" hareket etmesini sağlar (bu, örneğin `obj[::-1]` veya `obj[:, ::-1]` gibi bir dilimde geçerli olabilir).



## NumPy dtype Hiyerarşisi

- Bazen bir dizinin integer, float sayıları, dizeler veya Python nesneleri içерip içermediğini kontrol etmesi gereken kodunuz olabilir. Float sayılarının birden fazla türü (float16'dan float128'e kadar) olduğundan, dtype'in bir tür listesi arasında olup olmadığını kontrol etmek çok ayrıntılı olacaktır. dtype'lerin np.integer ve np.floating gibi üst sınıfları vardır ve bunlar np.issubdtype işleviyle birlikte kullanılabilir.

```
In [6]: ints = np.ones(10, dtype=np.uint16)
In [7]: floats = np.ones(10, dtype=np.float32)
In [8]: np.issubdtype(ints.dtype, np.integer)
Out[8]: True
In [9]: np.issubdtype(floats.dtype, np.floating)
Out[9]: True
```

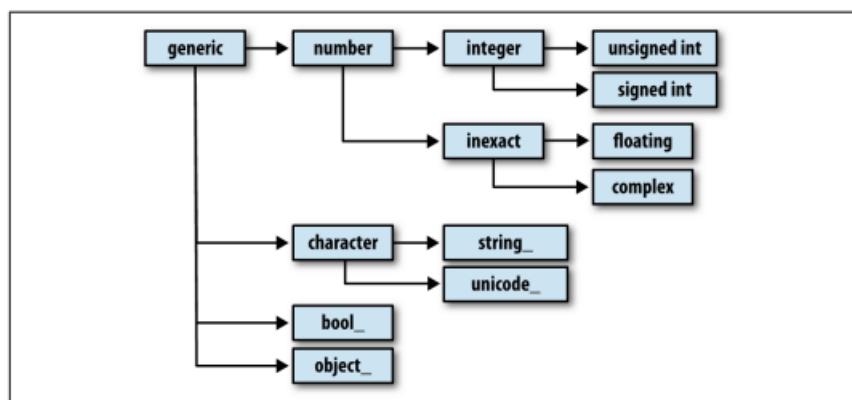
- Belirli bir dtype'in tüm ebeveyn sınıflarını, tipin mro yöntemini çağırarak görebilirsiniz:

```
In [10]: np.float64.mro()
```

```
Out[10]: [numpy.float64,
           numpy.floating,
           numpy.inexact,
           numpy.number,
           numpy.generic,
           float,
           object]
```

```
In [11]: np.issubdtype(ints.dtype, np.number)
```

```
Out[11]: True
```



numPy dtype sınıfı hiyerarşisi

## A.2 Gelişmiş Dizi Manipülasyonu

### Dizileri Yeniden Şekillendirme

- Çoğu durumda, herhangi bir veriyi kopyalamadan bir diziyi bir şeilden diğerine dönüştürebilirsiniz. Bunu yapmak için, yeni şekli belirten bir diziyi yeniden şekillendirme dizi örneği yöntemine iletin. Örneğin, bir matris halinde yeniden düzenlemek istediğimiz tek boyutlu bir değerler dizisine sahip olduğumuzu varsayıyalım:

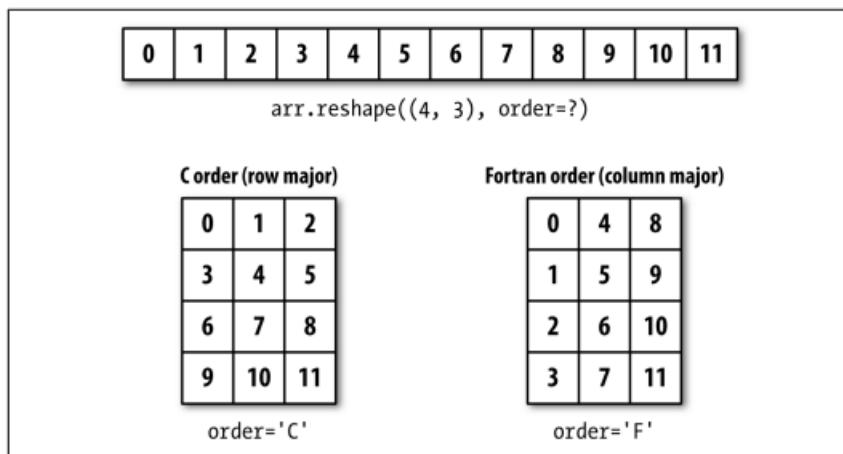
```
In [12]: arr = np.arange(8)

In [13]: arr

Out[13]: array([0, 1, 2, 3, 4, 5, 6, 7])

In [14]: arr.reshape((4, 2))

Out[14]: array([[0, 1],
   [2, 3],
   [4, 5],
   [6, 7]])
```



C (büyük satır) veya Fortran (büyük sütun) sırasına göre yeniden şekillendirme

- Çok boyutlu bir dizi de yeniden şekillendirilebilir:

```
In [15]: arr.reshape((4, 2)).reshape((2, 4))

Out[15]: array([[0, 1, 2, 3],
   [4, 5, 6, 7]])
```

- Geçirilen şekil boyutlarından biri -1 olabilir, bu durumda bunun için kullanılan değer boyut verilerden çıkarılacaktır.
- Bir dizinin şekil özelliği bir tanımlama grubu olduğundan, yeniden şekillendirmeye de iletilebilir:

```
In [19]: other_arr = np.ones((3, 5))
```

```
In [20]: other_arr.shape
```

```
Out[20]: (3, 5)
```

```
In [21]: arr.reshape(other_arr.shape)
```

```
Out[21]: array([[ 0,  1,  2,  3,  4],
   [ 5,  6,  7,  8,  9],
   [10, 11, 12, 13, 14]])
```

- Tek boyutluandan daha yüksek bir boyuta doğru yeniden şekillendirme işleminin tersi işlemi, tipik olarak düzleştirme veya sökme olarak bilinir.

```
In [22]: arr = np.arange(15).reshape((5, 3))
```

```
In [23]: arr
```

```
Out[23]: array([[ 0,  1,  2],
   [ 3,  4,  5],
   [ 6,  7,  8],
   [ 9, 10, 11],
   [12, 13, 14]])
```

```
In [24]: arr.ravel()
```

```
Out[24]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

- Sonuçtaki değerler orijinal dizide bitişikse ravel temel değerlerin bir kopyasını üretmez. Flatten yöntemi, her zaman verinin bir kopyasını döndürmesi dışında ravel gibi davranır:

```
In [25]: arr.flatten()
```

```
Out[25]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

## C'ye Karşı Fortran Düzeni

- NumPy, verilerinizin bellekteki düzeni üzerinde kontrol ve esneklik sağlar. Varsayılan olarak NumPy dizileri ana satır sırasına göre oluşturulur. Uzamsal olarak bu, iki boyutlu bir veri diziniz varsa, dizinin her satırındaki öğelerin bitişik bellek konumlarında depolandığı anlamına gelir. Satır ana sıralamasının alternatifi, sütun ana sıralamasıdır; bu, her bir veri sütunundaki değerlerin bitişik bellek konumlarında saklandığı anlamına gelir.
- Tarihsel nedenlerden dolayı satır ve sütun ana sırası sırasıyla C ve Fortran sırası olarak da bilinir. FORTRAN 77 dilinde matrislerin tümü ana sütundur.
- Reshape ve ravel gibi işlevler, dizideki verilerin kullanılma sırasını belirten bir sıra argümanını kabul eder. Bu genellikle çoğu durumda 'C' veya 'F' olarak ayarlanır.

```
In [26]: arr = np.arange(12).reshape((3, 4))

In [27]: arr

Out[27]: array([[ 0,  1,  2,  3],
   [ 4,  5,  6,  7],
   [ 8,  9, 10, 11]])

In [28]: arr.ravel()

Out[28]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

In [29]: arr.ravel('F')

Out[29]: array([ 0,  4,  8,  1,  5,  9,  2,  6, 10,  3,  7, 11])
```

- İkiden fazla boyuta sahip dizileri yeniden şekillendirmek biraz kafa karıştırıcı olabilir C ve Fortran sırası arasındaki temel fark, boyutların yürütme şeklidir:

#### C/satır ana sırası

Once yüksek boyutları geçin (örneğin, eksen 0'da ilerlemeden önce eksen 1).

#### Fortran/sütun ana sırası

Daha yüksek boyutları en son geçin (örneğin, eksen 1'de ilerlemeden önce eksen 0).

## Dizileri Birleştirme ve Bölme

- `numpy.concatenate` dizilerin bir sırasını (tuple, list vb.) alır ve bunları giriş eksenini boyunca sırayla birleştirir:

```
In [30]: arr1 = np.array([[1, 2, 3], [4, 5, 6]])

In [31]: arr2 = np.array([[7, 8, 9], [10, 11, 12]])

In [32]: np.concatenate([arr1, arr2], axis=0)

Out[32]: array([[ 1,  2,  3],
   [ 4,  5,  6],
   [ 7,  8,  9],
   [10, 11, 12]])

In [33]: np.concatenate([arr1, arr2], axis=1)

Out[33]: array([[ 1,  2,  3,  7,  8,  9],
   [ 4,  5,  6, 10, 11, 12]])
```

- Yaygın birleştirme türleri için vstack ve hstack gibi bazı kolaylık sağlayan işlevler vardır. Önceki işlemler şu şekilde ifade edilebilirdi:

```
In [34]: np.vstack((arr1, arr2))
```

```
Out[34]: array([[ 1,  2,  3],
                 [ 4,  5,  6],
                 [ 7,  8,  9],
                 [10, 11, 12]])
```

```
In [35]: np.hstack((arr1, arr2))
```

```
Out[35]: array([[ 1,  2,  3,  7,  8,  9],
                 [ 4,  5,  6, 10, 11, 12]])
```

- Öte yandan bölme, bir diziyi bir eksen boyunca birden çok dizİYE ayırır.

```
In [36]: arr = np.random.randn(5, 2)
```

```
In [37]: arr
```

```
Out[37]: array([[-0.59145266,  0.53762928],
                 [-0.05855363,  2.05743286],
                 [-0.14787861,  1.39973341],
                 [ 0.56633586, -0.45658297],
                 [-2.04417555, -0.10830662]])
```

```
In [38]: first, second, third = np.split(arr, [1, 3])
```

```
In [39]: first
```

```
Out[39]: array([[-0.59145266,  0.53762928]])
```

```
In [40]: second
```

```
Out[40]: array([[-0.05855363,  2.05743286],
                 [-0.14787861,  1.39973341]])
```

```
In [41]: third
```

```
Out[41]: array([[ 0.56633586, -0.45658297],
                 [-2.04417555, -0.10830662]])
```

- Np.split'e iletilen [1, 3] değeri, dizinin parçalara bölüneceği endeksleri belirtir.

## Dizi birleştirme işlevleri

Function	Description
concatenate	Most general function, concatenates collection of arrays along one axis
vstack, row_stack	Stack arrays row-wise (along axis 0)
hstack	Stack arrays column-wise (along axis 1)
column_stack	Like hstack, but converts 1D arrays to 2D column vectors first
dstack	Stack arrays “depth”-wise (along axis 2)
split	Split array at passed locations along a particular axis
hsplit/vsplit	Convenience functions for splitting on axis 0 and 1, respectively

## Yığınlama yardımcıları: r\_ ve c\_

- NumPy ad alanında, dizileri daha kısa ve öz hale getiren r\_ ve c\_ olmak üzere iki özel nesne vardır.

```
In [42]: arr = np.arange(6)
```

```
In [43]: arr1 = arr.reshape((3, 2))
```

```
In [44]: arr2 = np.random.randn(3, 2)
```

```
In [45]: np.r_[arr1, arr2]
```

```
Out[45]: array([[ 0.        ,  1.        ],
   [ 2.        ,  3.        ],
   [ 4.        ,  5.        ],
   [ 1.21365  , -0.62272881],
   [ 1.12593209, -0.88655254],
   [ 0.6133935 , -0.25180023]])
```

```
In [46]: np.c_[np.r_[arr1, arr2], arr]
```

```
Out[46]: array([[ 0.        ,  1.        ,  0.        ],
   [ 2.        ,  3.        ,  1.        ],
   [ 4.        ,  5.        ,  2.        ],
   [ 1.21365  , -0.62272881,  3.        ],
   [ 1.12593209, -0.88655254,  4.        ],
   [ 0.6133935 , -0.25180023,  5.        ]])
```

- Bunlar ayrıca dilimleri dizilere çevirebilir:

```
In [47]: np.c_[1:6, -10:-5]
```

```
Out[47]: array([[ 1, -10],
   [ 2, -9],
   [ 3, -8],
   [ 4, -7],
   [ 5, -6]])
```

## Tekrarlanan Öğeler: Döşeme (Tile) ve Tekrarlama (Repeat)

```
In [48]: arr = np.arange(3)
```

```
In [49]: arr
```

```
Out[49]: array([0, 1, 2])
```

```
In [50]: arr.repeat(3)
```

```
Out[50]: array([0, 0, 0, 1, 1, 1, 2, 2, 2])
```

- Varsayılan olarak, bir tam sayı iletirseniz her öğe bu sayıda tekrarlanır. Bir tamsayı dizisi iletirseniz, her öğe farklı sayıda tekrarlanabilir.

```
In [51]: arr.repeat([2, 3, 4])
Out[51]: array([0, 0, 1, 1, 1, 2, 2, 2, 2])

In [52]: arr = np.random.randn(2, 2)
In [53]: arr
Out[53]: array([[ 2.19696878, -0.27900399],
   [-0.50788502,  1.57527683]])

In [54]: arr.repeat(2, axis=0)
Out[54]: array([[ 2.19696878, -0.27900399],
   [ 2.19696878, -0.27900399],
   [-0.50788502,  1.57527683],
   [-0.50788502,  1.57527683]])
```

- Çok boyutlu dizilerin elemanları belirli bir eksen boyunca tekrarlanabilir.
- Hiçbir eksen geçilmezse, dizinin önce düzleştirileceğini unutmayın; bu muhtemelen istediğiniz şey değildir. Benzer şekilde, belirli bir dilimi farklı sayıda tekrarlamak için çok boyutlu bir diziyi tekrarlarken bir tamsayı dizisini iletебilirsiniz.

```
In [55]: arr.repeat([2, 3], axis=0)
Out[55]: array([[ 2.19696878, -0.27900399],
   [ 2.19696878, -0.27900399],
   [-0.50788502,  1.57527683],
   [-0.50788502,  1.57527683],
   [-0.50788502,  1.57527683]])

In [56]: arr.repeat([2, 3], axis=1)
Out[56]: array([[ 2.19696878,  2.19696878, -0.27900399, -0.27900399, -0.27900399],
   [-0.50788502, -0.50788502,  1.57527683,  1.57527683,  1.57527683]])
```

- Öte yandan döşeme, bir dizinin kopyalarını bir eksen boyunca istiflemek için kullanılan bir kısayoldur. Görsel olarak bunu "fayans döşemeye" benzer bir şey olarak düşünebilirsiniz:

```
In [57]: arr
Out[57]: array([[ 2.19696878, -0.27900399],
   [-0.50788502,  1.57527683]])

In [59]: np.tile(arr, 2)
Out[59]: array([[ 2.19696878, -0.27900399,  2.19696878, -0.27900399],
   [-0.50788502,  1.57527683, -0.50788502,  1.57527683]])
```

- İkinci argüman ise karo sayısıdır; Skaler ile döşeme sütun sütun yerine satır satır yapılır. Döşemenin ikinci argümanı, "döşemenin" düzenini gösteren bir tuple olabilir.

```
In [60]: arr
Out[60]: array([[ 2.19696878, -0.27900399],
   [-0.50788502,  1.57527683]])

In [61]: np.tile(arr, (2, 1))
Out[61]: array([[ 2.19696878, -0.27900399],
   [-0.50788502,  1.57527683],
   [ 2.19696878, -0.27900399],
   [-0.50788502,  1.57527683]])

In [62]: np.tile(arr, (3, 2))
Out[62]: array([[ 2.19696878, -0.27900399,  2.19696878, -0.27900399],
   [-0.50788502,  1.57527683, -0.50788502,  1.57527683],
   [ 2.19696878, -0.27900399,  2.19696878, -0.27900399],
   [-0.50788502,  1.57527683, -0.50788502,  1.57527683],
   [ 2.19696878, -0.27900399,  2.19696878, -0.27900399],
   [-0.50788502,  1.57527683, -0.50788502,  1.57527683]])
```

## Süslü (Fancy) İndeksleme Eşdeğerleri: Al (Take) ve Koy (Put)

- Dizilerin alt kümelerini almanın ve ayarlamanın bir yolu, tamsayı dizilerini kullanarak süslü indeksleme yapmaktadır.

```
In [63]: arr = np.arange(10) * 100
In [64]: inds = [7, 1, 2, 6]
In [65]: arr[inds]
Out[65]: array([700, 100, 200, 600])
```

- Yalnızca tek eksende seçim yapmanın özel durumunda yararlı olan alternatif ndarray yöntemleri vardır.

```
In [66]: arr.take(inds)
Out[66]: array([700, 100, 200, 600])

In [67]: arr.put(inds, 42)
In [68]: arr
Out[68]: array([  0,  42,  42, 300, 400, 500,  42,  42, 800, 900])

In [69]: arr.put(inds, [40, 41, 42, 43])
In [70]: arr
Out[70]: array([  0,  41,  42, 300, 400, 500,  43,  40, 800, 900])
```

- Diğer eksenleri birlikte kullanmak için eksen anahtar sözcüğünü iletebilirsiniz:

```
In [71]: inds = [2, 0, 2, 1]
In [72]: arr = np.random.randn(2, 4)
In [73]: arr
Out[73]: array([[-1.07528402,  1.25894201, -0.35373226, -2.01609798],
   [-0.08901271, -0.09459925,  1.09417523, -0.25082438]])
In [74]: arr.take(inds, axis=1)
Out[74]: array([[ -0.35373226, -1.07528402, -0.35373226,  1.25894201],
   [ 1.09417523, -0.08901271,  1.09417523, -0.09459925]])
```

## A.3 Yayıncılık

- Yayın, farklı şekillerdeki diziler arasında aritmetiğin nasıl çalıştığını açıklar. Bu güçlü bir özellik olabilir ancak deneyimli kullanıcılar için bile kafa karışıklığına neden olabilir. Yayının en basit örneği, bir skaler değerin bir diziyle birleştirildiğinde ortaya çıkar.

```
In [75]: arr = np.arange(5)
In [76]: arr
Out[76]: array([0, 1, 2, 3, 4])
In [77]: arr * 4
Out[77]: array([ 0,  4,  8, 12, 16])
```

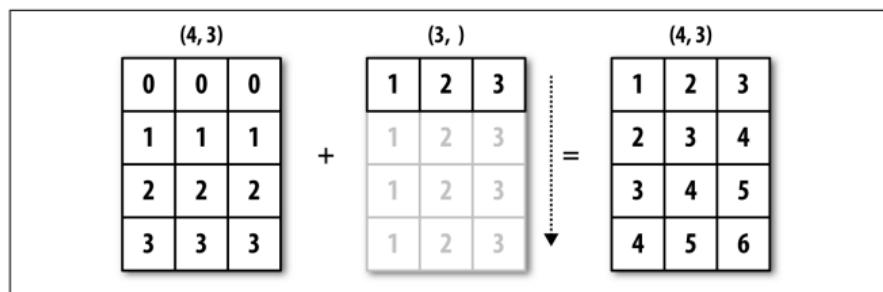
- Burada 4 skaler değerinin çarpma işlemindeki diğer tüm elemanlara yayındığını söylüyoruz.
- Örneğin, bir dizinin her sütununu, sütun ortalamalarını çıkararak küçültebiliriz. Bu durumda çok basit:

```
In [78]: arr = np.random.randn(4, 3)
In [79]: arr.mean(0)
Out[79]: array([-0.06103467, -0.61100009,  0.77847567])
In [80]: demeaned = arr - arr.mean(0)
In [81]: demeaned
Out[81]: array([[ -0.53630362, -0.5271783 ,  0.36098141],
   [-1.4216797 ,  0.2267708 , -0.49210923],
   [ 0.45625591,  1.03029905, -0.25564576],
   [ 1.50172741, -0.72989155,  0.38677358]])
In [82]: demeaned.mean(0)
Out[82]: array([-5.55111512e-17,  0.00000000e+00, -4.16333634e-17])
```

- Sıraları bir yayın operasyonu olarak küçük düşürmek biraz daha dikkat gerektiriyor. Kurallara uyulduğu sürece, bir dizinin herhangi bir boyutu boyunca potansiyel olarak daha düşük boyutlu değerlerin yayınlanması (iki boyutlu bir dizinin her sütunundan satır ortalamalarının çıkarılması gibi) mümkündür.

### Yayın Kuralı

- Her takip boyutu için (yani uçtan başlayarak) eksen uzunlukları eşleşiyorsa veya uzunluklardan herhangi biri 1 ise, iki dizi yayın için uyumludur. Daha sonra yayın, eksik veya uzunluk 1 boyutları üzerinden gerçekleştirilir.



- Son örneği düşünün ve bunun yerine ortalama değeri her satırdan çıkarmak istediğimizi varsayıyalım. `arr.mean(0)`'ın uzunluğu 3 olduğundan, `arr`'daki takip boyutu 3 olduğundan ve dolayısıyla eşleştiğinden, 0 eksenin boyunca yayın yapmak için uygundur. Kurallara göre, eksen 1 üzerinden çıkarma yapmak (yani her satırdan satır ortalamasını çıkarmak) için, daha küçük olan dizinin şekli (4, 1) olmalıdır:

```
In [83]: arr
Out[83]: array([[-0.59733828, -1.13817839,  1.13945708],
   [-1.48271437, -0.38422929,  0.28636644],
   [ 0.39522125,  0.41929895,  0.52282991],
   [ 1.44069274, -1.34089164,  1.16524925]])
```

```
In [84]: row_means = arr.mean(1)
In [85]: row_means.shape
Out[85]: (4,)

In [86]: row_means.reshape(4, 1)
Out[86]: array([[-0.19868653],
   [-0.52685907],
   [ 0.44578337],
   [ 0.42168345]])
```

```
In [87]: demeaned = arr - row_means.reshape(4, 1)
In [88]: demeaned.mean(1)
Out[88]: array([-7.40148683e-17,  0.00000000e+00,  1.85037171e-17, -7.40148683e-17])
```

$(4, 3)$	$(4, 1)$	$(4, 3)$																								
<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>2</td><td>2</td><td>2</td></tr> <tr><td>3</td><td>3</td><td>3</td></tr> </table>	0	0	0	1	1	1	2	2	2	3	3	3	+	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>2</td><td>2</td><td>2</td></tr> <tr><td>3</td><td>3</td><td>3</td></tr> <tr><td>4</td><td>4</td><td>4</td></tr> </table>	1	1	1	2	2	2	3	3	3	4	4	4
0	0	0																								
1	1	1																								
2	2	2																								
3	3	3																								
1	1	1																								
2	2	2																								
3	3	3																								
4	4	4																								

2D dizinin 1. eksenin üzerinden yayın yapma

- Başka bir örnek için, bu sefer iki boyutlu bir diziyi bir diziye ekleyerek eksen 0 boyunca üç boyutlu olan.

$(3, 4, 2)$	$(4, 2)$	$(3, 4, 2)$																
<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td>0</td><td>1</td></tr> <tr><td>2</td><td>3</td></tr> <tr><td>4</td><td>5</td></tr> <tr><td>6</td><td>7</td></tr> </table>	0	1	2	3	4	5	6	7	+	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr><td>0</td><td>1</td></tr> <tr><td>2</td><td>3</td></tr> <tr><td>4</td><td>5</td></tr> <tr><td>6</td><td>7</td></tr> </table>	0	1	2	3	4	5	6	7
0	1																	
2	3																	
4	5																	
6	7																	
0	1																	
2	3																	
4	5																	
6	7																	

3D dizinin 0 eksenin üzerinden yayın yapma

## Diger Eksenlerde Yayin

- Daha yüksek boyutlu dizilerle yayın yapmak daha da kafa karıştırıcı görünebilir, ancak bu aslında kurallara uyma meselesidir. Bunu yapmazsanız şöyle bir hata alırsınız:

```
In [89]: arr - arr.mean(1)
```

```
ValueError
Cell In[89], line 1
----> 1 arr - arr.mean(1)
```

```
Traceback (most recent call last)
```

```
ValueError: operands could not be broadcast together with shapes (4,3) (4,)
```

- 0 eksenin dışındaki eksenlerde daha düşük boyutlu bir diziyle aritmetik işlem yapmak istemek oldukça yaygındır. Yayın kuralına göre, daha küçük dizide “yayın boyutları” 1 olmalıdır. Burada gösterilen satır küçültme örneğinde bu, satırın yeniden şekillendirilmesinin (4,) yerine (4, 1) şeklinde olması anlamına geliyordu:

```
In [90]: arr = arr.mean(1).reshape((4, 1))

Out[90]: array([[-0.39865175, -0.93949186,  1.33814361],
               [-0.95585529,  0.14262978,  0.81322551],
               [-0.05056212, -0.02648442,  0.07704654],
               [ 1.01900929, -1.76257509,  0.7435658 ]])
```

- Üç boyutlu durumda, üç boyuttan herhangi biri üzerinden yayın yapmak yalnızca verilerin şekil uyumlu olacak şekilde yeniden şekillendirilmesi meselesidir.
- Bu nedenle yaygın bir sorun, özellikle yayın amaçları için uzunluğu 1 olan yeni bir eksenin eklenmesi ihtiyacıdır. Yeniden şekillendirmeyi kullanmak bir seçenekdir ancak bir eksen eklemek yeni şekli belirten bir demet oluşturmayı gerektirir. Bu çoğu zaman sıkıcı bir egzersiz olabilir. Böylece NumPy dizileri indeksleme yoluya yeni eksenler eklemek için özel bir sözdizimi sunar. Yeni eksenin eklemek için "tam" dilimlerle birlikte özel np.newaxis özelliğini kullanırız:

```
In [91]: arr = np.zeros((4, 4))

In [92]: arr_3d = arr[:, np.newaxis, :]

In [93]: arr_3d.shape

Out[93]: (4, 1, 4)

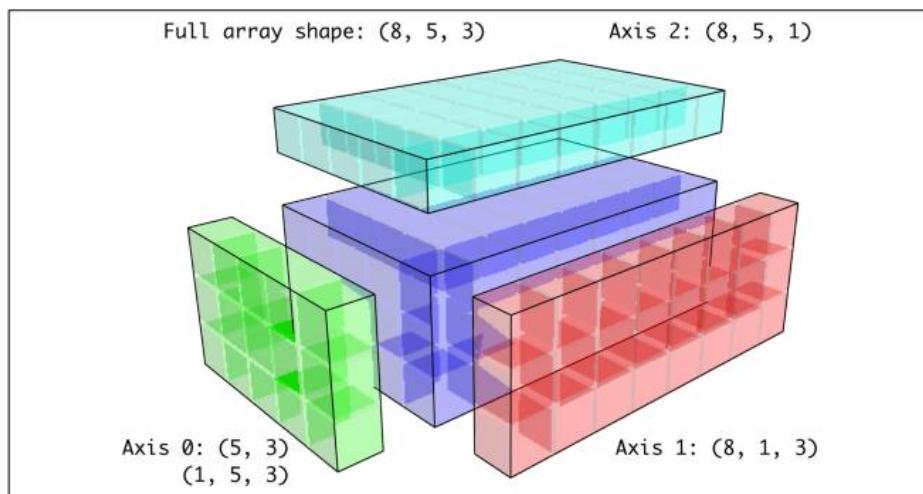
In [94]: arr_1d = np.random.normal(size=3)

In [95]: arr_1d[:, np.newaxis]

Out[95]: array([[ 0.34397363],
               [-1.22251952],
               [-2.58106364]])

In [96]: arr_1d[np.newaxis, :]

Out[96]: array([[ 0.34397363, -1.22251952, -2.58106364]])
```



3D dizi üzerinden yayın yapmak için uyumlu 2D dizi şekilleri

- Dolayısıyla, eğer üç boyutlu bir dizimiz olsaydı ve eksen 2'yi küçültmek isteseydik, şunu yazmamız gerekirdi:

```
In [97]: arr = np.random.randn(3, 4, 5)

In [98]: depth_means = arr.mean(2)

In [99]: depth_means

Out[99]: array([[ 0.20255971, -0.23616011, -0.30385224,  0.20662981],
   [-0.57749186, -0.08303911, -0.10125356, -0.01921815],
   [ 0.29547483, -0.28934479,  0.24857978,  0.32266552]])

In [100]: depth_means.shape

Out[100]: (3, 4)

In [101]: demeaned = arr - depth_means[:, :, np.newaxis]

In [102]: demeaned.mean(2)

Out[102]: array([[-4.44089210e-17, -8.88178420e-17,  0.00000000e+00,
   4.44089210e-17],
   [-1.11022302e-16, -8.32667268e-18,  0.00000000e+00,
   4.16333634e-18],
   [ 6.66133815e-17, -6.66133815e-17,  0.00000000e+00,
  -4.44089210e-17]])
```

## Dizi Değerlerini Yayına Göre Ayarlama

- Aritmetik işlemleri yöneten aynı yayın kuralı, dizi indeksleme yoluyla değerlerin ayarlanması için de geçerlidir. Basit bir durumda şöyle şeyler yapabiliriz:

```
In [103]: arr = np.zeros((4, 3))

In [104]: arr[:] = 5

In [105]: arr

Out[105]: array([[5., 5., 5.],
   [5., 5., 5.],
   [5., 5., 5.],
   [5., 5., 5.]])
```

- Ancak eğer elimizde olsaydı dizinin sütunlarına yerleştirmek istediğimiz tek boyutlu değerler dizisi, şekil uyumlu olduğu sürece bunu yapabiliriz:

```
In [106]: col = np.array([1.28, -0.42, 0.44, 1.6])
In [107]: arr[:] = col[:, np.newaxis]
In [108]: arr
Out[108]: array([[ 1.28,  1.28,  1.28],
   [-0.42, -0.42, -0.42],
   [ 0.44,  0.44,  0.44],
   [ 1.6 ,  1.6 ,  1.6 ]])

In [109]: arr[:2] = [[-1.37], [0.509]]
In [110]: arr
Out[110]: array([[-1.37, -1.37, -1.37],
   [ 0.509,  0.509,  0.509],
   [ 0.44,  0.44,  0.44],
   [ 1.6 ,  1.6 ,  1.6 ]])
```

## A.4 Gelişmiş Fonksiyon Kullanımı

### ufunc Örnek Yöntemleri

- NumPy'nin ikili fonksiyonlarının her biri, belirli türdeki özel vektörleştirilmiş işlemleri gerçekleştirmek için özel yöntemlere sahiptir.
- reduce tek bir dizi alır ve bir dizi ikili işlem gerçekleştirerek değerlerini istege bağlı olarak bir eksen boyunca toplar. Örneğin, bir dizideki öğeleri toplamanın alternatif bir yolu np.add.reduce kullanmaktadır:

```
In [111]: arr = np.arange(10)
In [112]: np.add.reduce(arr)
Out[112]: 45
In [113]: arr.sum()
Out[113]: 45
```

- Başlangıç değeri (ekleme için 0) ufunc'a bağlıdır. Bir eksen geçilirse azaltma o eksen boyunca gerçekleştirilir. Bu, belirli türdeki soruları kısa ve öz bir şekilde yanıtlanmanıza olanak tanır. Örnek olarak, bir dizinin her satırındaki değerlerin sıralanıp sıralanmadığını kontrol etmek için np.logical\_and komutunu kullanabiliriz:

```
In [114]: np.random.seed(12346) # tekrarlanabilirlik için
In [115]: arr = np.random.randn(5, 5)
In [116]: arr[::-2].sort(1) # birkaç satırı sırala
In [117]: arr[:, :-1] < arr[:, 1:]
Out[117]: array([[ True,  True,  True,  True],
       [False,  True, False, False],
       [ True,  True,  True,  True],
       [ True, False,  True,  True],
       [ True,  True,  True,  True]])
In [118]: np.logical_and.reduce(arr[:, :-1] < arr[:, 1:], axis=1)
Out[118]: array([ True, False,  True, False,  True])
```

- Mantıksal\_and.reduce'un all yöntemine eşdeğer olduğunu unutmayın.
- cumsum'un toplamla ilişkili olduğu gibi, biriktirme de azaltmayla ilişkilidir. Ara "birikmiş" değerlerle aynı boyutta bir dizi üretir:

```
In [119]: arr = np.arange(15).reshape((3, 5))
In [120]: np.add.accumulate(arr, axis=1)
Out[120]: array([[ 0,  1,  3,  6, 10],
       [ 5, 11, 18, 26, 35],
       [10, 21, 33, 46, 60]])
```

- dış (outer) ikili olarak gerçekleştirir iki dizi arasındaki çapraz çarpım:

```
In [121]: arr = np.arange(3).repeat([1, 2, 2])
In [122]: arr
Out[122]: array([0, 1, 1, 2, 2])
In [123]: np.multiply.outer(arr, np.arange(5))
Out[123]: array([[0, 0, 0, 0, 0],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 2, 4, 6, 8],
       [0, 2, 4, 6, 8]])
```

- Dışın çıktısı, girdilerin boyutlarının toplamı olan bir boyuta sahip olacaktır:

```
In [124]: x, y = np.random.randn(3, 4), np.random.randn(5)
```

```
In [125]: result = np.subtract.outer(x, y)
```

```
In [126]: result.shape
```

```
Out[126]: (3, 4, 5)
```

- Son yöntem olan `reducat`, bir "yerel azaltma" gerçekleştirir, aslında dizi dilimlerinin bir araya toplandığı bir dizi gruplama işlemi gerçekleştirir. Değerlerin nasıl bölüneceğini ve toplanacağını gösteren bir dizi "kutu kenarları"nı kabul eder.

```
In [127]: arr = np.arange(10)
```

```
In [128]: np.add.reduceat(arr, [0, 5, 8])
```

```
Out[128]: array([10, 18, 17])
```

- Sonuçlar `arr[0:5]`, `arr[5:8]` ve `arr[8:]` üzerinden gerçekleştirilen indirimelerdir (burada toplamlar). Diğer yöntemlerde olduğu gibi eksen argümanını iletebilirsiniz:

```
In [129]: arr = np.multiply.outer(np.arange(4), np.arange(5))
```

```
In [130]: arr
```

```
Out[130]: array([[ 0,  0,  0,  0,  0],
   [ 0,  1,  2,  3,  4],
   [ 0,  2,  4,  6,  8],
   [ 0,  3,  6,  9, 12]])
```

```
In [131]: np.add.reduceat(arr, [0, 2, 4], axis=1)
```

```
Out[131]: array([[ 0,  0,  0],
   [ 1,  5,  4],
   [ 2, 10,  8],
   [ 3, 15, 12]])
```

## İşlev Yöntemleri

Method	Description
<code>reduce(x)</code>	Aggregate values by successive applications of the operation
<code>accumulate(x)</code>	Aggregate values, preserving all partial aggregates
<code>reduceat(x, bins)</code>	"Local" reduce or "group by"; reduce contiguous slices of data to produce aggregated array
<code>outer(x, y)</code>	Apply operation to all pairs of elements in x and y; the resulting array has shape <code>x.shape + y.shape</code>

## Python'da Yeni İşlevler Yazma

- `numpy.frompyfunc`, giriş ve çıkış sayısı spesifikasyonuyla birlikte bir Python işlevini kabul eder. Örneğin, elementleri akıllıca ekleyen basit bir fonksiyon şu şekilde belirtilecektir:

```
In [1]: def add_elements(x, y):
         return x + y

In [4]: add_them = np.frompyfunc(add_elements, 2, 1)

In [5]: add_them(np.arange(8), np.arange(8))

Out[5]: array([0, 2, 4, 6, 8, 10, 12, 14], dtype=object)
```

- `Frompyfunc` kullanılarak oluşturulan işlevler her zaman Python nesnelerinin dizilerini döndürür; bu da sakıncalı olabilir. Çıktı türünü belirlemenize olanak tanıyan alternatif (ancak biraz daha az özellikli) bir işlev olan `numpy.vectorize` vardır:

```
In [6]: add_them = np.vectorize(add_elements, otypes=[np.float64])

In [7]: add_them(np.arange(8), np.arange(8))

Out[7]: array([ 0.,  2.,  4.,  6.,  8., 10., 12., 14.])
```

- Bu işlevler, `ufunc` benzeri işlevler oluşturmanın bir yolunu sağlar, ancak çok yavaşırlar çünkü her bir öğeyi hesaplamak için bir Python işlev çağrıları gereklidir; bu, NumPy'nin C tabanlı `ufunc` döngülerinden çok daha yavaşır:

```
In [8]: arr = np.random.randn(10000)

In [12]: %timeit add_them(arr, arr)
1.83 ms ± 62.7 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)

In [13]: %timeit np.add(arr, arr)
3.21 µs ± 164 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

## A.5 Yapılandırılmış ve Kayıt Dizileri

- Şu ana kadar `ndarray`'in homojen bir veri taşıyıcısı olduğunu fark etmiş olabilirsiniz; yani her ögenin `dtype` tarafından belirlenen aynı sayıda bayt kapladığı bir bellek bloğunu temsil eder. Görünüşte bu, heterojen veya tablo benzeri verileri temsil etmenize izin vermiyor gibi görünebilir. Yapılandırılmış bir dizi, her ögenin C'deki bir yapıyı (dolayısıyla "yapılardırılmış" ad) veya SQL tablosundaki birden çok adlandırılmış alana sahip bir satırı temsil ettiği düşünülebilen bir `ndarray`'dır:

```
In [14]: dtype = [('x', np.float64), ('y', np.int32)]  
  
In [15]: sarr = np.array([(1.5, 6), (np.pi, -2)], dtype=dtype)  
  
In [16]: sarr  
  
Out[16]: array([(1.5, 6), (3.14159265, -2)],  
               dtype=[('x', '<f8'), ('y', '<i4')])
```

- Yapılandırılmış bir dtype belirtmenin birkaç yolu vardır. Tipik bir yol, (alan\_adi, alan\_veri\_tipi) içeren tuplelar'ın listesidir. Artık dizinin elemanları, elemanlarına bir sözlük gibi erişilebilen tuple benzeri nesnelerdir:

```
In [17]: sarr[0]  
Out[17]: (1.5, 6)  
  
In [18]: sarr[0]['y']  
Out[18]: 6
```

- Alan adları dtype.names özelliğinde saklanır. Yapılandırılmış dizideki bir alana eriştiğinizde, veriler üzerinde aşamalı bir görünüm döndürülür, böylece hiçbir şey kopyalanmaz:

```
In [19]: sarr['x']  
Out[19]: array([1.5, 3.14159265])
```

## İç İçe Tipler ve Çok Boyutlu Alanlar

- Yapılandırılmış bir dtype belirlerken ek olarak bir şekli de iletebilirsiniz (int veya tuple olarak).

```
In [20]: dtype = [('x', np.int64, 3), ('y', np.int32)]  
  
In [21]: arr = np.zeros(4, dtype=dtype)  
  
In [22]: arr  
  
Out[22]: array([[0, 0, 0], 0), [[0, 0, 0], 0), [[0, 0, 0], 0), [[0, 0, 0], 0)],  
               dtype=[('x', '<i8', (3,)), ('y', '<i4')])
```

- Bu durumda, x alanı artık her kayıt için 3 uzunluğunda bir diziye karşılık gelir.

```
In [23]: arr[0]['x']

Out[23]: array([0, 0, 0], dtype=int64)
```

- Uygun bir şekilde, arr['x'] öğesine erişim, önceki örneklerde olduğu gibi tek boyutlu bir dizi yerine iki boyutlu bir dizi döndürür.

```
In [24]: arr['x']

Out[24]: array([[0, 0, 0],
                 [0, 0, 0],
                 [0, 0, 0],
                 [0, 0, 0]], dtype=int64)
```

- Bu, daha karmaşık, iç içe geçmiş yapıları bir dizideki tek bir bellek bloğu olarak ifade etmenize olanak sağlar. Daha karmaşık yapılar oluşturmak için dtype'leri de iç içe kullanabilirsiniz.

```
In [25]: dtype = [('x', [(('a', 'f8'), ('b', 'f4'))]), ('y', np.int32)]

In [26]: data = np.array([(1, 2), (3, 4)], dtype=dtype)

In [27]: data['x']

Out[27]: array([(1., 2.), (3., 4.)], dtype=[('a', '<f8'), ('b', '<f4')])

In [28]: data['y']

Out[28]: array([5, 6])

In [29]: data['x']['a']

Out[29]: array([1., 3.])
```

## Neden Yapılandırılmış Diziler Kullanılmalı?

- Örneğin bir pandasın DataFrame'i ile karşılaştırıldığında, NumPy yapılandırılmış dizileri nispeten düşük seviyeli bir araçtır. Bir bellek bloğunu gelişigüzel karmaşık iç içe geçmiş sütunlara sahip bir tablo yapısı olarak yorumlamak için bir araç sağlarlar. Dizideki her öğe bellekte sabit sayıda bayt olarak temsil edildiğinden, yapılandırılmış diziler diske ve diskten veri yazmanın (bellek haritaları dahil), ağ üzerinden taşınmasının ve benzeri diğer kullanımların çok hızlı ve etkili bir yolunu sağlar.

## A.6 Sıralama Hakkında Daha Fazla Bilgi (Sorting)

- Python'un yerleşik listesi gibi, ndarray sort örneği yöntemi de yerinde sıralamadır; bu, dizi içeriklerinin yeni bir dizi üretilmeden yeniden düzenlendiği anlamına gelir.

```
In [30]: arr = np.random.randn(6)

In [31]: arr.sort()

In [32]: arr

Out[32]: array([-0.82072344, -0.46211388, -0.41997449,  0.28818772,  0.90804652,
       1.43093117])
```

- Dizileri yerinde sıralarken, dizinin farklı bir ndarray görünümü olması durumunda orijinal dizinin değiştirileceğini bilmeniz gereklidir.

```
In [33]: arr = np.random.randn(3, 5)

In [34]: arr

Out[34]: array([[ -1.69772933,   0.00747051,  -0.67130137,   1.45937823,   0.99518516],
       [-0.29315372,  -0.56031247,   2.08470829,  -0.77047569,   0.60013754],
       [ 1.51557106,   0.82770709,   0.07961657,  -0.42965668,   1.5197362 ]])

In [36]: arr[:, 0].sort() # İlk sütun değerlerini yerinde sıralayın

In [37]: arr

Out[37]: array([[ -1.69772933,   0.00747051,  -0.67130137,   1.45937823,   0.99518516],
       [-0.29315372,  -0.56031247,   2.08470829,  -0.77047569,   0.60013754],
       [ 1.51557106,   0.82770709,   0.07961657,  -0.42965668,   1.5197362 ]])
```

- Öte yandan, numpy.sort bir dizinin yeni, sıralanmış bir kopyasını oluşturur. Aksi takdirde, ndarray.sort ile aynı argümanları (tür gibi) kabul eder.

```
In [38]: arr = np.random.randn(5)

In [39]: arr

Out[39]: array([ 1.55060304,  0.38711744, -0.24629807,  1.446323 , -0.21019207])

In [40]: np.sort(arr)

Out[40]: array([-0.24629807, -0.21019207,  0.38711744,  1.446323 ,  1.55060304])

In [41]: arr

Out[41]: array([ 1.55060304,  0.38711744, -0.24629807,  1.446323 , -0.21019207])
```

- Bu sıralama yöntemlerinin tümü, veri bölümlerini iletilen eksen boyunca bağımsız olarak sıralamak için bir eksen bağımsız değişkeni alır.

```
In [42]: arr = np.random.randn(3, 5)
```

```
In [43]: arr
```

```
Out[43]: array([[ 0.07863922, -0.16831688, -0.81578809, -1.6053349 ,  0.63851316],
   [-1.59660287,  0.84720246, -1.01648514,  0.72060266,  0.6426473 ],
   [ 0.55001508, -0.55582771,  1.79383698,  0.53955583,  1.58780404]])
```

```
In [44]: arr.sort(axis=1)
```

```
In [45]: arr
```

```
Out[45]: array([[-1.6053349 , -0.81578809, -0.16831688,  0.07863922,  0.63851316],
   [-1.59660287, -1.01648514,  0.6426473 ,  0.72060266,  0.84720246],
   [-0.55582771,  0.53955583,  0.55001508,  1.58780404,  1.79383698]])
```

- Sıralama yöntemlerinden hiçbirinin azalan düzende sıralama seçeneğinin olmadığını fark edebilirsiniz. Bu pratikte bir sorundur çünkü dizi dilimleme görünümler üretir, dolayısıyla bir kopya üretmez veya herhangi bir hesaplama çalışması gerektirmez. Birçok Python kullanıcısı, bir liste değerleri için, değerlerin[::-1] ters sırada bir liste döndürmesi "hilesine" aşınadır. Aynı şey ndarray'ler için de geçerlidir.

```
In [46]: arr[:, ::-1]
```

```
Out[46]: array([[ 0.63851316,  0.07863922, -0.16831688, -0.81578809, -1.6053349 ],
   [ 0.84720246,  0.72060266,  0.6426473 , -1.01648514, -1.59660287],
   [ 1.79383698,  1.58780404,  0.55001508,  0.53955583, -0.55582771]])
```

## Dolaylı Sıralamalar: argsort ve lexsort

- Veri analizinde veri kümelerini bir veya daha fazla anahtara göre yeniden sıralamanız gerekebilir. Örneğin, bazı öğrencilerle ilgili veri tablosunun önce soyadına, ardından adına göre sıralanması gerekebilir. Bu dolaylı sıralamanın bir örneğidir. Bir anahtar veya anahtarlar (bir değer dizisi veya birden çok değer dizisi) verildiğinde, verileri sıralı düzende nasıl yeniden sıralayacağınızı söyleyen bir tamsayı indeksleri dizisi (bunlara halk dilinde indeksleyiciler denilebilir) elde etmek istersiniz. Bunun için iki yöntem argsort ve numpy.lexsort'tur.

```
In [47]: values = np.array([5, 0, 1, 3, 2])  
  
In [48]: indexer = values.argsort()  
  
In [49]: indexer  
  
Out[49]: array([1, 2, 4, 3, 0], dtype=int64)  
  
In [50]: values[indexer]  
  
Out[50]: array([0, 1, 2, 3, 5])
```

- Daha karmaşık bir örnek olarak, bu kod iki boyutlu bir diziyi ilk satırına göre yeniden sıralar:

```
In [51]: arr = np.random.randn(3, 5)  
  
In [52]: arr[0] = values  
  
In [53]: arr  
  
Out[53]: array([[ 5.          ,  0.          ,  1.          ,  3.          ,  2.          ],  
                 [ 1.19340004,  1.21070658, -0.07253555, -1.59057047, -1.18884933],  
                 [ 2.71313756, -0.13933997, -0.15534891,  1.11936402,  1.31342319]])  
  
In [54]: arr[:, arr[0].argsort()]  
  
Out[54]: array([[ 0.          ,  1.          ,  2.          ,  3.          ,  5.          ],  
                 [ 1.21070658, -0.07253555, -1.18884933, -1.59057047,  1.19340004],  
                 [-0.13933997, -0.15534891,  1.31342319,  1.11936402,  2.71313756]])
```

- lexsort argsort'a benzer, ancak birden çok anahtar dizide dolaylı sözlükbilimsel sıralama gerçekleştirir. Ad ve soyadına göre tanımlanan bazı verileri sıralamak istediğimizi varsayıyalım:

```
In [55]: first_name = np.array(['Bob', 'Jane', 'Steve', 'Bill', 'Barbara'])  
  
In [56]: last_name = np.array(['Jones', 'Arnold', 'Arnold', 'Jones', 'Walters'])  
  
In [57]: sorter = np.lexsort((first_name, last_name))  
  
In [58]: sorter  
  
Out[58]: array([1, 2, 3, 0, 4], dtype=int64)  
  
In [59]: zip(last_name[sorter], first_name[sorter])  
  
Out[59]: <zip at 0x16e59de2340>
```

- Lexsort'u ilk kullandığınızda biraz kafa karıştırıcı olabilir çünkü verileri sıralamak için kullanılan tuşların sırası, geçirilen son diziyle başlar. Burada soyadı, ad'dan önce kullanıldı.

## Alternatif Sıralama Algoritmaları

- Kararlı bir sıralama algoritması eşit öğelerin göreceli konumunu korur. Bu, göreceli sıralamanın anlamlı olduğu dolaylı sıralamalarda özellikle önemli olabilir.

```
In [60]: values = np.array(['2:first', '2:second', '1:first', '1:second',
                           '1:third'])
```

```
In [61]: key = np.array([2, 2, 1, 1, 1])
```

```
In [62]: indexer = key.argsort(kind='mergesort')
```

```
In [63]: indexer
```

```
Out[63]: array([2, 3, 4, 0, 1], dtype=int64)
```

```
In [64]: values.take(indexer)
```

```
Out[64]: array(['1:first', '1:second', '1:third', '2:first', '2:second'],
               dtype='<U8')
```

## Dizi sıralama yöntemleri

Kind	Speed	Stable	Work space	Worst case
'quicksort'	1	No	0	$O(n^2)$
'mergesort'	2	Yes	$n / 2$	$O(n \log n)$
'heapsort'	3	No	0	$O(n \log n)$

## Dizileri Kısmen Sıralama

- Sıralamanın amaçlarından biri dizideki en büyük veya en küçük öğeleri belirlemek olabilir. NumPy, bir diziyi  $k$ 'inci en küçük öğe etrafında bölmlemek için `numpy.partition` ve `np.argpartition` gibi optimize edilmiş yöntemlere sahiptir.

```
In [65]: np.random.seed(12345)

In [66]: arr = np.random.randn(20)

In [67]: arr

Out[67]: array([-0.20470766,  0.47894334, -0.51943872, -0.5557303 ,  1.96578057,
       1.39340583,  0.09290788,  0.28174615,  0.76902257,  1.24643474,
       1.00718936, -1.29622111,  0.27499163,  0.22891288,  1.35291684,
       0.88642934, -2.00163731, -0.37184254,  1.66902531, -0.43856974])

In [68]: np.partition(arr, 3)

Out[68]: array([-2.00163731, -1.29622111, -0.5557303 , -0.51943872, -0.37184254,
       -0.43856974, -0.20470766,  0.28174615,  0.76902257,  0.47894334,
       1.00718936,  0.09290788,  0.27499163,  0.22891288,  1.35291684,
       0.88642934,  1.39340583,  1.96578057,  1.66902531,  1.24643474])
```

- partition(arr, 3) öğesini çağrıdıktan sonra sonuçtaki ilk üç öğe, belirli bir sıraya göre sıralanmayan en küçük üç değerdir. numpy.argmax, numpy.argsort'a benzer şekilde, verileri eşdeğer sıraya göre yeniden düzenleyen dizinleri döndürür.

```
In [69]: indices = np.argmax(arr, 3)

In [70]: indices

Out[70]: array([16, 11, 3, 2, 17, 19, 0, 7, 8, 1, 10, 6, 12, 13, 14, 15, 5,
       4, 18, 9], dtype=int64)

In [71]: arr.take(indices)

Out[71]: array([-2.00163731, -1.29622111, -0.5557303 , -0.51943872, -0.37184254,
       -0.43856974, -0.20470766,  0.28174615,  0.76902257,  0.47894334,
       1.00718936,  0.09290788,  0.27499163,  0.22891288,  1.35291684,
       0.88642934,  1.39340583,  1.96578057,  1.66902531,  1.24643474])
```

## numpy.searchsorted: Sıralanmış Bir Dizideki Öğeleri Bulma

- searchsorted, sıralanmış bir dizide ikili arama gerçekleştiren ve sıralılığı korumak için değerin eklenmesi gereken dizideki konumu döndüren bir dizi yöntemidir.

```
In [72]: arr = np.array([0, 1, 7, 12, 15])

In [73]: arr.searchsorted(9)

Out[73]: 3
```

- Ayrıca bir dizi indeksi geri almak için bir dizi değer de iletebilirsiniz.

```
In [74]: arr.searchsorted([0, 8, 11, 16])
```

```
Out[74]: array([0, 3, 3, 5], dtype=int64)
```

- Arama sıralamasının 0 ögesi için 0 döndürdügünü fark etmiş olabilirsiniz. Bunun nedeni, varsayılan davranışın, dizini eşit değerlerden oluşan bir grubun sol tarafında döndürmek olmalıdır.

```
In [75]: arr = np.array([0, 0, 0, 1, 1, 1, 1])
```

```
In [76]: arr.searchsorted([0, 1])
```

```
Out[76]: array([0, 3], dtype=int64)
```

```
In [77]: arr.searchsorted([0, 1], side='right')
```

```
Out[77]: array([3, 7], dtype=int64)
```

- Başka bir arama sıralaması uygulaması olarak, 0 ile 10.000 arasında bir değer dizimiz ve verileri bölmek için kullanmak istediğimiz ayrı bir "kova kearları" dizimiz olduğunu varsayıyalım:

```
In [78]: data = np.floor(np.random.uniform(0, 10000, size=50))
```

```
In [79]: bins = np.array([0, 100, 1000, 5000, 10000])
```

```
In [80]: data
```

```
Out[80]: array([9940., 6768., 7908., 1709., 268., 8003., 9037., 246., 4917.,
   5262., 5963., 519., 8950., 7282., 8183., 5002., 8101., 959.,
   2189., 2587., 4681., 4593., 7095., 1780., 5314., 1677., 7688.,
   9281., 6094., 1501., 4896., 3773., 8486., 9110., 3838., 3154.,
   5683., 1878., 1258., 6875., 7996., 5735., 9732., 6340., 8884.,
   4954., 3516., 7142., 5039., 2256.])
```

- Daha sonra her bir veri noktasının hangi aralığa ait olduğuna dair bir etiketleme elde etmek için (burada 1, paket [0, 100) anlamına gelir), basitçe arama sıralamasını kullanabiliriz.

```
In [81]: labels = bins.searchsorted(data)
```

```
In [82]: labels
```

```
Out[82]: array([4, 4, 4, 3, 2, 4, 4, 2, 3, 4, 4, 4, 2, 4, 4, 4, 4, 4, 4, 2, 3, 3, 3, 3,
   4, 3, 4, 3, 4, 4, 4, 3, 3, 3, 4, 4, 3, 3, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 4, 4, 3], dtype=int64)
```

- Bu, pandaların gruplandırmasıyla birleştirildiğinde verileri gruplamak için kullanılabilir.

```
In [85]: pd.Series(data).groupby(labels).mean()
```

```
Out[85]: 2    498.000000
          3    3064.277778
          4   7389.035714
          dtype: float64
```

## A.7 Numba ile Hızlı NumPy Fonksiyonlarının Yazılması

- Numba, CPU'ları, GPU'ları veya diğer donanımları kullanarak NumPy benzeri veriler için hızlı işlevler oluşturan açık kaynaklı bir projedir. Python kodunu derlenmiş makine koduna çevirmek için LLVM Projesini kullanır.
- Numba'yı tanıtmak için, for döngüsü kullanarak  $(x - y).mean()$  ifadesini hesaplayan saf bir Python fonksiyonunu ele alalım:

```
In [90]: def mean_distance(x, y):
    nx = len(x)
    result = 0.0
    count = 0
    for i in range(nx):
        result += x[i] - y[i]
        count += 1
    return result / count
```

- Bu işlev çok yavaştır.

```
In [91]: x = np.random.randn(10000000)
```

```
In [92]: y = np.random.randn(10000000)
```

```
In [93]: %timeit mean_distance(x, y)
```

4.31 s ± 108 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

```
In [94]: %timeit (x - y).mean()
```

46.9 ms ± 1.27 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

- Numba rastgele Python kodunu derleyemez, ancak sayısal algoritmalar yazmak için en kullanışlı olan saf Python'un önemli bir alt kümesini destekler.

```
In [98]: from numba import float64, njit

@njit(float64(float64[:, :], float64[:, :]))
def mean_distance(x, y):
    return (x - y).mean()
```

## Numba ile Özel numpy.ufunc Nesneleri Oluşturma

- numba.vectorize işlevi, yerleşik ufunc'lar gibi davranışları derlenmiş NumPy ufunc'lar oluşturur. Numpy.add'in Python uygulamasını ele alalım:

```
In [99]: from numba import vectorize

@vectorize
def nb_add(x, y):
    return x + y

In [100]: x = np.arange(10)

In [101]: nb_add(x, x)

Out[101]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18], dtype=int64)
```

## A.8 Gelişmiş Dizi Girişи ve Çıkışı

### Bellek Eşlemeli Dosyalar

- Bellek eşlemeli dosya, diskteki ikili verilerle sanki bir bellek içi dizide saklanmış gibi etkileşim kurmaya yönelik bir yöntemdir. NumPy, ndarray benzeri bir memmap nesnesi uygulayarak büyük bir dosyanın küçük bölmelerinin, tüm diziyi belleğe okumadan okunmasını ve yazılmasını sağlar. Ek olarak, bir memmap, bir bellek içi dizile aynı yöntemlere sahiptir ve bu nedenle, bir ndarray'in beklentiği birçok algoritmanın yerine kullanılabilir.
- Yeni bir bellek haritası oluşturmak için np.memmap işlevini kullanın ve bir dosya yolu, dtype, şekil ve dosya modu iletin.

```
In [104]: mmap = np.memmap('my mmap', dtype='float64', mode='w+', shape=(10000, 10000))

In [105]: mmap

Out[105]: memmap([[0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.],
 ...,
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.]])
```

- Bir memmap'in dilimlenmesi, diskteki verilere ilişkin görüntülerini döndürür.

```
In [106]: section = mmap[:5]
```

- Bunlara veri atarsanız, bellekte arabelleğe alınır (Python dosya nesnesi gibi), ancak bunu flush'u çağırarak diske yazabilirsiniz.

```
In [107]: section[:] = np.random.randn(5, 10000)
```

```
In [108]: mmap.flush()
```

```
In [109]: mmap
```

```
Out[109]: memmap([[ 1.37140985,  0.93127837,  0.60573747, ..., -0.62115557,
   -0.46780136,  0.47874865],
   [ 0.42296545,  0.83060431,  0.69976547, ...,  1.28831447,
   0.58858679, -1.42755372],
   [ 2.16005954, -1.24616489,  2.44470054, ...,  0.86866129,
   0.28019716,  2.13008671],
   ...,
   [ 0.          ,  0.          ,  0.          , ...,  0.          ,
   0.          ,  0.          ],
   [ 0.          ,  0.          ,  0.          , ...,  0.          ,
   0.          ,  0.          ],
   [ 0.          ,  0.          ,  0.          , ...,  0.          ,
   0.          ,  0.          ],
   [ 0.          ,  0.          ,  0.          , ...,  0.          ,
   0.          ,  0.          ]])
```

```
In [110]: del mmap
```

- Bir bellek haritası kapsam dışına çıktığında ve çöp olarak toplandığında, tüm değişiklikler de diske aktarılacaktır. Mevcut bir hafıza haritasını açarken yine de dosya, diskte meta veri içermeyen yalnızca bir ikili veri bloğu olduğundan, türü ve şekli belirtin.

```
In [111]: mmap = np.memmap('my mmap', dtype='float64', shape=(10000, 10000))
```

```
In [112]: mmap
```

```
Out[112]: memmap([[ 1.37140985,  0.93127837,  0.60573747, ..., -0.62115557,
   -0.46780136,  0.47874865],
   [ 0.42296545,  0.83060431,  0.69976547, ...,  1.28831447,
   0.58858679, -1.42755372],
   [ 2.16005954, -1.24616489,  2.44470054, ...,  0.86866129,
   0.28019716,  2.13008671],
   ...,
   [ 0.          ,  0.          ,  0.          , ...,  0.          ,
   0.          ,  0.          ],
   [ 0.          ,  0.          ,  0.          , ...,  0.          ,
   0.          ,  0.          ],
   [ 0.          ,  0.          ,  0.          , ...,  0.          ,
   0.          ,  0.          ],
   [ 0.          ,  0.          ,  0.          , ...,  0.          ,
   0.          ,  0.          ]])
```

## HDF5 ve Diğer Dizi Depolama Seçenekleri

- PyTables ve h5py, dizi verilerini verimli ve sıkıştırılabilir HDF5 formatında (HDF, hiyerarşik veri formatı anlamına gelir) depolamak için NumPy dostu arayüzler sağlayan iki Python projesidir. Yüzlerce gigabayt ve hatta terabaytlarca veriyi HDF5 formatında güvenle saklayabilirsiniz.

## A.9 Performans İpuçları

- NumPy kullanarak koddan iyi performans elde etmek genellikle basittir, çünkü dizi işlemleri genellikle nispeten son derece yavaş olan saf Python döngülerinin yerini alır.

Aşağıdaki liste akılda tutulması gereken bazı hususları kısaca özetlemektedir:

- Python döngülerini ve koşullu mantığı dizi işlemlerine ve boole dizi işlemlerine dönüştürün
- Mükün olduğunda yayını kullanın
- Verilerin kopyalanmasını önlemek için dizi görünümelerini (dilimleme) kullanın
- Ufuncs ve ufunc yöntemlerini kullanın

## Bitişik Belleğin Önemi

- Bir dizinin bellek düzeninin bitişik olduğunu söylemek, öğelerin Fortran (büyük sütun) veya C (büyük satır) sıralamasına göre dizide göründükleri sırayla bellekte depolandığı anlamına gelir. Varsayılan olarak NumPy dizileri C bitişik veya yalnızca bitişik olarak oluşturulur. Bir sütun ana dizisi, örneğin devrik C-bitşik bir diziye bu nedenle Fortran-bitşik olduğu söylenir. Bu özellikler ndarray'deki flags özelliği aracılığıyla açıkça kontrol edilebilir.

```
In [113]: arr_c = np.ones((1000, 1000), order='C')
```

```
In [114]: arr_f = np.ones((1000, 1000), order='F')
```

```
In [115]: arr_c.flags
```

```
Out[115]: C_CONTIGUOUS : True  
F_CONTIGUOUS : False  
OWNDATA : True  
WRITEABLE : True  
ALIGNED : True  
WRITEBACKIFCOPY : False
```

```
In [116]: arr_f.flags
```

```
Out[116]: C_CONTIGUOUS : False  
F_CONTIGUOUS : True  
OWNDATA : True  
WRITEABLE : True  
ALIGNED : True  
WRITEBACKIFCOPY : False
```

```
In [117]: arr_f.flags.f_contiguous
```

```
Out[117]: True
```

- Bu örnekte, bu dizilerin satırlarının toplanması teorik olarak arr\_c için arr\_f'den daha hızlı olmalıdır çünkü satırlar bellekte bitişiktir. Burada IPython'da %timeit kullanılır.

```
In [118]: %timeit arr_c.sum(1)
1.11 ms ± 50.9 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

```
In [119]: %timeit arr_f.sum(1)
503 µs ± 34.8 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

- NumPy'den daha fazla performans elde etmek istediğinizde burası genellikle biraz çaba harcamanız gereken bir yerdir. İstenilen bellek sırasına sahip olmayan bir diziniz varsa, 'C' veya 'F'yi kopyalayıp iletmemeyi kullanabilirsiniz.

```
In [120]: arr_f.copy('C').flags
Out[120]: C_CONTIGUOUS : True
           F_CONTIGUOUS : False
           OWNDATA : True
           WRITEABLE : True
           ALIGNED : True
           WRITEBACKIFCOPY : False
```

- Bir dizide görünüm oluştururken sonucun bitişik olmasının garanti edilmediğini unutmayın.

```
In [121]: arr_c[:50].flags.contiguous
Out[121]: True
```

```
In [122]: arr_c[:, :50].flags
Out[122]: C_CONTIGUOUS : False
           F_CONTIGUOUS : False
           OWNDATA : False
           WRITEABLE : True
           ALIGNED : True
           WRITEBACKIFCOPY : False
```

# EK B | IPython Sistemi Hakkında Daha Fazlası

## B.1 Komut Geçmişini Kullanma

- IPython, yürüttüğünüz her komutun metnini içeren küçük bir disk üzerinde veritabanı tutar. Bu çeşitli amaçlara hizmet eder:
  - Daha önce yürütülen komutları minimum düzeyde yazma işlemiyle arama, tamamlama ve yürütme
  - Oturumlar arasında komut geçmişinin sürdürülmesi
  - Giriş/çıkış geçmişini bir dosya kaydetme

### Komut Geçmişini Arama ve Yeniden Kullanma

- IPython kabuğu, önceki kodu veya diğer komutları aramanıza ve yürütmenize olanak tanır. Kendinizi sıkılıkla aynı komutları tekrarlarken bulabileceğiniz için bu faydalıdır. %run komutunu veya başka bir kod parçasını kullanın.

```
In [1]: %run first/second/third/data_script.py
```

- Diyelim ki: komutunu çalıştırınız ve daha sonra betiğin sonuçlarını araştırdınız (başarılı bir şekilde çalıştığını varsayıarak), ancak yanlış bir hesaplama yaptığınızı buldunuz. Sorunu çözdükten ve data\_script.py'yi değiştirdikten sonra, %run komutunun birkaç harfini yazmaya başlayabilir ve ardından Ctrl-P tuş kombinasyonuna veya yukarı ok tuşuna basabilirsiniz. Bu, yazdığınız harflerle eşleşen önceki ilk komutu komut geçmişinde arayacaktır. Ctrl-P veya yukarı ok tuşuna birden çok kez basmak, geçmişte arama yapmaya devam edecektir. Yürütmek istediğiniz komutu atlarsanız korkmayın. İleriye gidebilirsin Ctrl-N veya aşağı ok tuşuna basarak komut geçmişinde gezinebilirsiniz. Bunu birkaç kez yaptıktan sonra hiç düşünmeden bu tuşlara basmaya başlayabilirsiniz!
- Ctrl-R'yi kullanmak, bash kabuğu gibi Unix tarzı kabuklarda kullanılan okuma satırının sağladığı aynı kısmı artımlı arama yeteneğini sağlar. Windows'ta okuma satırı işlevi IPython tarafından taklit edilir. Bunu kullanmak için Ctrl-R tuşlarına basın ve ardından aramak istediğiniz giriş satırında bulunan birkaç karakteri yazın.

```
In [2]: a_command = foo(x, y, z)
```

- Ctrl-R tuşlarına basmak, yazdığınız karakterlerle eşleşen her satırın geçmişinde geçiş yapacaktır.

## Giriş ve Çıkış Değişkenleri

- Bir IPython oturumu hem giriş komutlarına hem de çıktı Python nesnelerine olan referansları özel değişkenlerde saklar. Önceki iki çıktı sırasıyla `_` (bir alt çizgi) ve `__` (iki alt çizgi) değişkenlerinde saklanır.

```
In [4]: 2 ** 27
```

```
Out[4]: 134217728
```

```
In [5]: _
```

```
Out[5]: 134217728
```

- Giriş değişkenleri `_iX` gibi adlandırılan değişkenlerde saklanır; burada X, giriş satırı numarasıdır. Her giriş değişkenine karşılık gelen bir çıkış değişkeni `_X` vardır. Yani, giriş satırı 27'den sonra, diyelim ki, iki yeni değişken olacak: `_27` (çıkış için) ve giriş için `_i27`:

```
In [6]: foo = 'bar'
```

```
In [7]: foo
```

```
Out[7]: 'bar'
```

```
In [9]: _i7
```

```
Out[9]: 'foo'
```

```
In [10]: _7
```

```
Out[10]: 'bar'
```

- Giriş değişkenleri dizeler olduğundan Python exec anahtar sözcüğüyle tekrar çalıştırılabilirler.

```
In [12]: exec(_i7)
```

- Burada `_i7`, In [7]'deki kod girişini ifade eder.
- Çeşitli magic işlevler, giriş ve çıkış geçmişiyle çalışmanızı olanak tanır. `%hist`, giriş geçmişinin tamamını veya bir kısmını satır numaraları olsun veya olmasın yazdırabilir. `%reset`, etkileşimli ad alanını ve isteğe bağlı olarak giriş ve çıkış önbelleklerini temizlemek içindir. `%xdel` magic işlevi, IPython makinesinden belirli bir nesneye yapılan tüm referansları kaldırmak için tasarlanmıştır.

## B.2 İşletim Sistemiyle Etkileşim

- IPython'un bir diğer özelliği de dosya sistemine ve işletim sistemi kabuğuna sorunsuz bir şekilde erişmenize olanak sağlamaşıdır. Bu, diğer şeylerin yanı sıra, Windows veya Unix (Linux, macOS) kabuğunda yaptığınız gibi standart komut satırı eylemlerinin çoğunu IPython'dan çıkışına gerek kalmadan gerçekleştirebileceğiniz anlamına gelir. Buna kabuk komutları, dizinlerin değiştirilmesi ve bir komutun sonuçlarının bir Python nesnesinde (liste veya string). Ayrıca basit komut takma adı ve dizin yerimi özellikleri de vardır.

### IPython sistemiyle ilgili komutlar

Command	Description
<code>!cmd</code>	Execute <code>cmd</code> in the system shell
<code>output = !cmd args</code>	Run <code>cmd</code> and store the stdout in <code>output</code>
<code>%alias alias_name cmd</code>	Define an alias for a system (shell) command
<code>%bookmark</code>	Utilize IPython's directory bookmarking system
<code>%cd directory</code>	Change system working directory to passed directory
<code>%pwd</code>	Return the current system working directory
<code>%pushd directory</code>	Place current directory on stack and change to target directory
<code>%popd</code>	Change to directory popped off the top of the stack
<code>%dirs</code>	Return a list containing the current directory stack
<code>%dhist</code>	Print the history of visited directories
<code>%env</code>	Return the system environment variables as a dict
<code>%matplotlib</code>	Configure matplotlib integration options

### Kabuk (Shell) Komutları ve Takma Adlar

- IPython'da bir satırın ünlem işaretü ! veya bang ile başlatılması, IPython'a sistem kabuğundaki bang'den sonraki her şeyi yürütmesini söyler. Bu, dosyaları silebileceğiniz (işletim sisteminize bağlı olarak rm veya del kullanarak), dizinleri değiştirebileceğiniz veya başka herhangi bir işlemi gerçekleştirebileceğiniz anlamına gelir.
- Bir kabuk komutunun konsol çıktısını, ! ile kaçan ifadeyi bir değişkene atayarak bir değişkende saklayabilirsiniz. Örneğin, Ethernet üzerinden interneye bağlanan Linux tabanlı makinede IP adresini Python değişkeni olarak alabilirsiniz:

```
In [1]: ip_info = !ifconfig wlan0 | grep "inet "
In [2]: ip_info[0].strip()
Out[2]: 'inet addr:10.0.0.11 Bcast:10.0.0.255 Mask:255.255.255.0'
```

- Döndürülen Python nesnesi ip\_info aslında konsol çıktısının çeşitli sürümlerini içeren özel bir liste türündür.

- IPython ayrıca ! kullanırken mevcut ortamda tanımlanan Python değerlerini de `macrodata.csv` `spx.csv` `tips.csv` değiştirebilir. Bunu yapmak için değişken adının

```
In [15]: foo = 'test*'
```

```
In [16]: !ls $foo
```

önüne dolar işaretini \$ koyun:

- %alias magic işlevi, kabuk komutları için özel kısayollar tanımlayabilir. Basit bir örnek olarak:

```
In [19]: %alias ll ls -l
```

```
In [20]: ll /usr
```

```
total 332
drwxr-xr-x  2 root root  69632 2012-01-29 20:36 bin/
drwxr-xr-x  2 root root   4096 2010-08-23 12:05 games/
drwxr-xr-x 123 root root  20480 2011-12-26 18:08 include/
drwxr-xr-x 265 root root 126976 2012-01-29 20:36 lib/
drwxr-xr-x  44 root root  69632 2011-12-26 18:08 lib32/
lrwxrwxrwx  1 root root     3 2010-08-23 16:02 lib64 -> lib/
drwxr-xr-x  15 root root  4096 2011-10-13 19:03 local/
drwxr-xr-x  2 root root  12288 2012-01-12 09:32 sbin/
drwxr-xr-x 387 root root 12288 2011-11-04 22:53 share/
drwxrwsr-x  24 root src   4096 2011-07-17 18:38 src/
```

- Birden fazla komutu, tipki komut satırında olduğu gibi, noktalı virgülle ayırarak çalıştırabilirsiniz:

```
In [22]: %alias test_alias (cd examples; ls; cd ..)
```

```
In [24]: test_alias
test4.py  test.py  test.xml
```

## Dizin Yer İmi (Bookmark) Sistemi

- IPython, ortak dizinler için takma adlar kaydetmenize olanak tanıyan basit bir dizin işaretleme sistemine sahiptir, böylece çok kolay bir şekilde dolaşabilirsiniz. Örneğin, bu kitabın ek materyallerine işaret eden bir yerimi oluşturmak istediğiniz varsayıyalım:

```
In [25]: %bookmark py4da /home/wesm/code/pydata-book
```

- Bunu yaptıktan sonra %cd magic kullandığımızda tanımladığımız herhangi bir yer imini kullanabiliriz:

```
In [26]: cd py4da
```

```
(bookmark:py4da) -> /home/wesm/code/pydata-book
[WinError 3] Sistem belirtilen yolu bulamıyor: '/home/wesm/code/pydata-book'
```

```
In [27]: %bookmark -l
```

```
Current bookmarks:
py4da -> /home/wesm/code/pydata-book
```

- Bir yer imi adı geçerli çalışma dizinizdeki bir dizin adıyla çakışıyorsa şunu kullanabilirsiniz: Yer imi konumunu geçersiz kılmak ve kullanmak için -b bayrağı. %bookmark ile -l seçeneğini kullanmak tüm yer imlerinizi listeler.

### B.3 Yazılım Geliştirme Araçları

- IPython, etkileşimli bilgi işlem ve veri araştırması için rahat bir ortam olmasının yanı sıra, genel Python yazılım geliştirme için de yararlı bir yardımcı olabilir. Veri analizi uygulamalarında öncelikle doğru koda sahip olmak önemlidir. Neyse ki IPython, yerleşik Python pdb hata ayıklayıcısını yakından entegre etti ve geliştirdi. İkinci olarak kodunuzun hızlı olmasını istiyorsunuz. Bunun için IPython'un kullanımı kolay kod zamanlaması ve profil oluşturma araçları vardır.

### Etkileşimli Hata Ayıklayıcı

- IPython'un hata ayıklayıcısı, istisna geri izlemelerindeki her satır için sekme tamamlama, sözdizimi vurgulama ve bağlam ile pdb'yi geliştirir. Kodda hata ayıklamak için en iyi zamanlardan biri, bir hata oluştuktan hemen sonrasındadır. %debug komutu, bir istisnadan hemen sonra girildiğinde, "postmortem" hata ayıklayıcıyı çağırır ve sizi istisnanın ortaya çıktıği yığın çerçevesine bırakır.

```
In [29]: run examples/ipython_bug.py
```

```
AssertionError                                                 Traceback (most recent call last)
/home/wesm/code/pydata-book/examples/ipython_bug.py in <module>()
    13     throws_an_exception()
    14
--> 15 calling_things()

/home/wesm/code/pydata-book/examples/ipython_bug.py in calling_things()
    11 def calling_things():
    12     works_fine()
--> 13     throws_an_exception()
    14
    15 calling_things()

/home/wesm/code/pydata-book/examples/ipython_bug.py in throws_an_exception()
     7     a = 5
     8     b = 6
-->  9     assert(a + b == 10)
    10
    11 def calling_things():

AssertionError:

In [3]: %debug
> /home/wesm/code/pydata-book/examples/ipython_bug.py(9) throws_an_exception()
    8     b = 6
-->  9     assert(a + b == 10)
    10

ipdb>
```

- Hata ayıklayıcıya girdikten sonra, istege bağlı Python kodunu çalıştırabilir ve her yiğin çerçevesindeki (yorumlayıcı tarafından "canlı tutulan") tüm nesneleri ve verileri keşfedebilirsiniz. Varsayılan olarak hatanın olduğu en düşük seviyeden başlarsınız. u (yükarı) ve d (aşağı) tuşlarına basarak yiğin izlemenin seviyeleri arasında geçiş yapabilirsiniz:

```
ipdb> u
> /home/wesm/code/pydata-book/examples/ipython_bug.py(13)calling_things()
  12      works_fine()
  ---> 13      throws_an_exception()
  14
```

- %pdb komutunun çalıştırılması, IPython'un herhangi bir istisnadan sonra otomatik olarak hata ayıklayıcıyı çağırmasını sağlar; bu, birçok kullanıcının özellikle yararlı bulacağı bir moddur.
- Kod geliştirmeye yardımcı olması için hata ayıklayıcıyı kullanmak da kolaydır, özellikle de kesme noktaları ayarlamak veya her aşamadaki durumu incelemek için bir işlevin veya komut dosyasının yürütülmesinde adım adım ilerlemek istediğinizde. Bunu başarmanın birkaç yolu vardır. Bunlardan ilki, iletilen komut dosyasındaki herhangi bir kodu çalıştırmadan önce hata ayıklayıcıyı çağrıran -d bayrağıyla birlikte %run'u kullanmaktadır. Komut dosyasına girmek için hemen s (adım) tuşuna basmalısınız:

In [37]: `run -d ipython_bug.py`

```
Breakpoint 1 at /home/wesm/code/pydata-book/examples/ipython_bug.py:1
NOTE: Enter 'c' at the ipdb> prompt to start your script.
> <string>(1)<module>()

ipdb> s
--Call--
> /home/wesm/code/pydata-book/examples/ipython_bug.py(1)<module>()
1---> 1 def works_fine():
  2     a = 5
  3     b = 6
```

- Works\_fine yöntemini kullanarak ve c (devam) tuşuna basarak kesme noktasına (breakpoint) ulaşana kadar scripti çalıştırın:

```
ipdb> b 12
ipdb> c
> /home/wesm/code/pydata-book/examples/ipython_bug.py(12)calling_things()
  11 def calling_things():
  2---> 12      works_fine()
  13      throws_an_exception()
```

- Bu noktada, bir sonraki satıra geçmek için n (sonraki) tuşuna basarak Works\_fine()'a geçebilir veya Works\_fine()'ı çalıştırabilirsiniz:

```
ipdb> n
> /home/wesm/code/pydata-book/examples/ipython_bug.py(13)calling_things()
  2      12      works_fine()
  ---> 13      throws_an_exception()
  14
```

- Daha sonra `throws_an_Exception`'a adım atıp hatanın olduğu satırı ilerleyebilir ve kapsam içindeki değişkenlere bakabiliriz. Hata ayıklayıcı komutlarının değişken adlara göre öncelikli olduğunu unutmayın; bu gibi durumlarda değişkenlerin başına ! işaretlerini incelemek için:

```

ipdb> s
--Call--
> /home/wesm/code/pydata-book/examples/ipython_bug.py(6) throws_an_exception()
  5
----> 6 def throws_an_exception():
    7     a = 5

ipdb> n
> /home/wesm/code/pydata-book/examples/ipython_bug.py(7) throws_an_exception()
  6 def throws_an_exception():
----> 7     a = 5
    8     b = 6

ipdb> n
> /home/wesm/code/pydata-book/examples/ipython_bug.py(8) throws_an_exception()
  7     a = 5
----> 8     b = 6
    9     assert(a + b == 10)

ipdb> n
> /home/wesm/code/pydata-book/examples/ipython_bug.py(9) throws_an_exception()
  8     b = 6
----> 9     assert(a + b == 10)
    10

ipdb> !a
5
ipdb> !b
6

```

## (I) Python Hata Ayıklayıcı Komutları

Command	Action
<code>h(elp)</code>	Display command list
<code>help <i>command</i></code>	Show documentation for <i>command</i>
<code>c(ontinue)</code>	Resume program execution
<code>q(uit)</code>	Exit debugger without executing any more code
<code>b(reak) <i>number</i></code>	Set breakpoint at <i>number</i> in current file
<code>b <i>path/to/file.py:number</i></code>	Set breakpoint at line <i>number</i> in specified file
<code>s(tep)</code>	Step <i>into</i> function call
<code>n(ext)</code>	Execute current line and advance to next line at current level
<code>u(p)/d(own)</code>	Move up/down in function call stack
<code>a(rgs)</code>	Show arguments for current function
<code>debug <i>statement</i></code>	Invoke statement <i>statement</i> in new (recursive) debugger
<code>l(ist) <i>statement</i></code>	Show current position and context at current level of stack
<code>w(here)</code>	Print full stack trace with context at current position

## Hata Ayıklayıcıyı Kullanmanın Diğer Yolları

- Hata ayıklayıcıyı çağrımanın birkaç yararlı yolu daha vardır. Birincisi özel bir set\_trace işlevi kullanmaktadır (adını pdb.set\_trace'den almıştır), bu aslında "fakir adamın kırılma noktası"dır.

```
In [38]: from IPython.core.debugger import Pdb
```

```
In [41]: def set_trace():
    Pdb(color_scheme='Windows').set_trace(sys._getframe().f_back)
```

```
In [42]: def debug(f, *args, **kwargs):
    pdb = Pdb(color_scheme='Windows')
    return pdb.runcall(f, *args, **kwargs)
```

- İlk işlev olan set\_trace çok basittir. Daha yakından incelemek için kodunuzun geçici olarak durdurmak istediğiniz herhangi bir bölümünde set\_trace kullanabilirsiniz.

```
In [7]: run examples/ipython_bug.py
> /home/wesm/code/pydata-book/examples/ipython_bug.py(16)calling_things()
  15     set_trace()
  16     throws_an_exception()
  17
```

- c (devam) tuşuna basmak, kodun herhangi bir zarar vermeden normal şekilde devam etmesine neden olur.
- Az önce incelediğimiz hata ayıklama işlevi, rastgele bir işlev çağrısında etkileşimli hata ayıklayıcıyı kolayca çağrımanıza olanak tanır. Diyelim ki aşağıdaki gibi bir fonksiyon yazdık ve onun mantığını aşmak istedik:

```
In [44]: def f(x, y, z=1):
    tmp = x + y
    return tmp / z
```

- Normalde f kullanmak şuna benzer: f(1, 2, z=3). Bunun yerine f'ye adım atmak için, hata ayıklamaya yönelik ilk argüman olarak f'yi ve ardından f'ye iletilecek konumsal ve anahtar kelime argümanlarını iletin:

```
In [47]: debug(f, 1, 2, z=3)
> <ipython-input>(2)f()
  1 def f(x, y, z):
  2     tmp = x + y
  3     return tmp / z
ipdb>
```

- Son olarak hata ayıklayıcı %run ile kullanılabilir. %run -d ile bir komut dosyası çalıştırıldığınızda, doğrudan hata ayıklayıcıya bırakılacak ve herhangi bir kesme noktası ayarlamaya ve komut dosyasını başlatmaya hazır olacaksınız:

```
In [1]: %run -d examples/ipython_bug.py
Breakpoint 1 at /home/wesm/code/pydata-book/examples/ipython_bug.py:1
NOTE: Enter 'c' at the ipdb> prompt to start your script.
> <string>(1)<module>()

ipdb>
```

- b'yi bir satır numarasıyla eklemek, hata ayıklayıcıyı önceden ayarlanmış bir kesme noktasıyla başlatır:

```
In [2]: %run -d -b2 examples/ipython_bug.py
Breakpoint 1 at /home/wesm/code/pydata-book/examples/ipython_bug.py:2
NOTE: Enter 'c' at the ipdb> prompt to start your script.
> <string>(1)<module>()

ipdb> c
> /home/wesm/code/pydata-book/examples/ipython_bug.py(2)works_fine()
  1 def works_fine():
  2     a = 5
  3     b = 6

ipdb>
```

## Zamanlama Kodu: %time ve %timeit

- Daha büyük ölçekli veya daha uzun süre çalışan veri analizi uygulamaları için, çeşitli bileşenlerin veya bireysel ifadelerin veya işlev çağrılarının yürütme süresini ölçmek isteyebilirsiniz. Karmaşık bir süreçte hangi işlevlerin en çok zaman aldığı gösteren bir rapor isteyebilirsiniz. IPython, kodunuza geliştirirken ve test ederken bu bilgilere çok kolay ulaşmanızı sağlar.
- Yerleşik zaman modülünü ve onun time.clock ve time.time işlevlerini kullanarak elle zamanlama kodu yazmak genellikle sıkıcı ve tekrarlayıcıdır, çünkü aynı ilgi çekici olmayan standart kodu yazmanız gereklidir:

```
In [1]: import time

In [4]: start = time.time()
for i in range(iterations):
    #burada çalıştırılacak bazı kodlar
elapsed_per = (time.time() - start) / iterations
```

- Bu çok yaygın bir işlem olduğundan, IPython'un bu süreci sizin için otomatikleştirmek için %time ve %timeit olmak üzere iki sihirli (magic) işlevi vardır.
- %time bir ifadeyi bir kez çalıştırarak toplam yürütme süresini bildirir. Diyelim ki geniş bir dizi listemiz var ve belirli bir önekle başlayan tüm dizeleri seçmenin farklı yöntemlerini karşılaştırmak istiyoruz. İşte 600.000 dizeden oluşan basit bir liste ve yalnızca 'foo' ile başlayanları seçmenin iki özdeş yöntemi:

```
In [8]: # çok geniş bir dize listesi
strings = ['foo', 'foobar', 'baz', 'qux',
           'python', 'Guido Van Rossum'] * 100000
```

```
In [9]: method1 = [x for x in strings if x.startswith('foo')]
```

```
In [10]: method2 = [x for x in strings if x[:3] == 'foo']
```

- Performans açısından hemen hemen aynı olmaları gerekiyor gibi görünüyor, %time kullanarak emin olarak kontrol edebiliriz:

```
In [11]: %time method1 = [x for x in strings if x.startswith('foo')]
CPU times: total: 93.8 ms
Wall time: 149 ms
```

```
In [12]: %time method2 = [x for x in strings if x[:3] == 'foo']
CPU times: total: 93.8 ms
Wall time: 120 ms
```

- The wall time ("wall-clock time"ın kısaltması)'ın ana ilgi alanıdır. Yani ilk yöntem iki kat daha fazla zaman alıyor gibi görünüyor, ancak bu çok kesin bir ölçüm değil. Bu ifadeleri kendiniz birden çok kez %time-ing'i denerseniz, sonuçların biraz değişken olduğunu göreceksiniz. Daha kesin bir ölçüm elde etmek için şunu kullanın: %timeit sihirli işlevi. Rastgele bir ifade verildiğinde, daha doğru bir ortalama çalışma süresi üretmek için bir ifadeyi birden çok kez çalıştırmak için bir buluşsal yöntem vardır:

```
In [13]: %timeit [x for x in strings if x.startswith('foo')]
117 ms ± 9.71 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
In [14]: %timeit [x for x in strings if x[:3] == 'foo']
93.7 ms ± 8.46 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

- Daha büyük ölçekli veri analizi uygulamalarında milisaniyeler artmaya başlar.
- %timeit özellikle mikrosaniye (saniyenin milyonda biri) veya nano saniye (saniyenin milyarda biri) düzeyinde bile çok kısa yürütme sürelerine sahip ifadeleri ve işlevleri analiz etmek için kullanışlıdır. Bunlar ömensiz bir zaman gibi görünebilir, ancak elbette 1 milyon kez çağrılan 20 mikrosaniyelik bir işlev, 5 mikrosaniyelik bir işlevden 15 saniye daha uzun sürer. Önceki örnekte, performans özelliklerini anlamak için iki dize işlemini doğrudan karşılaştırabiliriz:

```
In [19]: x = 'foobar'

In [20]: y = 'foo'

In [21]: %timeit x.startswith(y)
          201 ns ± 7.52 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)

In [22]: %timeit x[:3] == y
          193 ns ± 6.7 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
```

## Temel Profil Oluşturma: %prun ve %run -p

- Profil oluşturma kodu, zamanın nerede harcandığının belirlenmesiyle ilgili olması dışında, zamanlama koduya yakından ilgilidir. Ana Python profil oluşturma aracı, IPython'a özel olmayan cProfile modülüdür. cProfile, her işlevde ne kadar zaman harcadığını takip ederken bir programı veya herhangi bir rastgele kod bloğunu çalıştırır.
- cProfile kullanmanın yaygın bir yolu, komut satırında tüm programı çalıştmak ve işlev başına toplam süreyi çıkarmaktır. Bir döngüde bazı doğrusal cebirler yapan ( $100 \times 100$ 'lük bir matris serisinin maksimum mutlak özdeğerlerini hesaplayan) basit bir komut dosyamız olduğunu varsayıyalım:

```
In [23]: import numpy as np
         from numpy.linalg import eigvals

In [33]: def run_experiment(niter=100):
          K = 100
          results = []
          for _ in xrange(niter):
              mat = np.random.randn(K, K)
              max_eigenvalue = np.abs(eigvals(mat)).max()
              results.append(max_eigenvalue)
          return results
some_results = run_experiment()
print 'Largest one we saw: %s' % np.max(some_results)
```

- Bu script'i, komut satırında aşağıdakileri kullanarak cProfile aracılığıyla çalıştırabilirsiniz:

```
In [34]: python -m cProfile cprof_example.py
```

- Bunu denerseniz çıktıının işlev adına göre sıralandığını göreceksiniz. Bu, en çok zamanın nerede harcandığına dair bir fikir edinmeyi biraz zorlaştırır, bu nedenle -s bayrağını kullanarak bir sıralama düzeni belirlemek çok yaygındır.

```
$ python -m cProfile -s cumulative cprof_example.py
Largest one we saw: 11.923204422
    15116 function calls (14927 primitive calls) in 0.720 seconds

Ordered by: cumulative time

      ncalls  tottime  percall  cumtime  percall filename:lineno(function)
            1   0.001    0.001    0.721    0.721 cprof_example.py:1(<module>)
        100   0.003    0.000    0.586    0.006 linalg.py:702(eigvals)
       200   0.572    0.003    0.572    0.003 {numpy.linalg.lapack_lite.dgeev}
         1   0.002    0.002    0.075    0.075 __init__.py:106(<module>)
       100   0.059    0.001    0.059    0.001 {method 'randn'}
         1   0.000    0.000    0.044    0.044 add_newdocs.py:9(<module>)
         2   0.001    0.001    0.037    0.019 __init__.py:1(<module>)
         2   0.003    0.002    0.030    0.015 __init__.py:2(<module>)
         1   0.000    0.000    0.030    0.030 type_check.py:3(<module>)
         1   0.001    0.001    0.021    0.021 __init__.py:15(<module>)
         1   0.013    0.013    0.013    0.013 numeric.py:1(<module>)
         1   0.000    0.000    0.009    0.009 __init__.py:6(<module>)
         1   0.001    0.001    0.008    0.008 __init__.py:45(<module>)
       262   0.005    0.000    0.007    0.000 function_base.py:3178(add_newdoc)
      100   0.003    0.000    0.005    0.000 linalg.py:162(_assertFinite)
      ...

```

- Çıktının yalnızca ilk 15 satırı gösterilir. Her işlevin içinde toplam ne kadar süre harcadığını görmek için cumtime sütununu aşağı doğru tarayarak okumak en kolay yoldur. Not: eğer bir fonksiyon başka bir fonksiyonu çağrırsa saatin çalışması durmaz. cProfile, her işlev çağrısının başlangıç ve bitiş zamanını kaydeder ve bunu zamanlamayı oluşturmak için kullanır.
- Komut satırı kullanımına ek olarak cProfile, yeni bir işlem çalıştırılmaya gerek kalmadan rastgele kod bloklarının profilini çıkarmak için programlı olarak da kullanılabilir. IPython, %prun komutunu ve %run için -p seçeneğini kullanan bu yeteneğe uygun bir arayüze sahiptir. %prun, cProfile ile aynı "komut satırı seçeneklerini" alır ancak tam bir .py dosyası yerine rastgele bir Python ifadesinin profilini çıkarır.

```
In [4]: %prun -l 7 -s cumulative run_experiment()
    4203 function calls in 0.643 seconds

Ordered by: cumulative time
List reduced from 32 to 7 due to restriction <7>

      ncalls  tottime  percall  cumtime  percall filename:lineno(function)
            1   0.000    0.000    0.643    0.643 <string>:1(<module>)
            1   0.001    0.001    0.643    0.643 cprof_example.py:4(run_experiment)
        100   0.003    0.000    0.583    0.006 linalg.py:702(eigvals)
       200   0.569    0.003    0.569    0.003 {numpy.linalg.lapack_lite.dgeev}
       100   0.058    0.001    0.058    0.001 {method 'randn'}
       100   0.003    0.000    0.005    0.000 linalg.py:162(_assertFinite)
       200   0.002    0.000    0.002    0.000 {method 'all' of 'numpy.ndarray'}
```

- Benzer şekilde, %run -p -s kümülatif cprof\_example.py çağrıması, IPython'dan asla ayrılmak zorunda kalmamanız dışında, komut satırı yaklaşımıyla aynı etkiye sahiptir.

## Bir Fonksiyonun Satır Satır Profilini Oluşturma

- Bazı durumlarda %prun'dan (veya başka bir cProfile tabanlı profil yönteminden) elde ettiğiniz bilgiler, bir işlevin yürütme süresiyle ilgili tüm hikayeyi anlatmayabilir veya o kadar karmaşık olabilir ki, işlev adına göre toplanan sonuçların yorumlanması zor olabilir. Bu durumda line\_profiler adında küçük bir kütüphane vardır (PyPI veya paket yönetim araçlarından biri aracılığıyla edinilebilir). Bir veya daha fazla işlevin satır satır profilini hesaplayan yeni bir magic işlev olan %lprun'u etkinleştirilen bir IPython uzantısı içerir. IPython yapılandırmanızı değiştirerek bu uzantıyı etkinleştirebilirsiniz. Aşağıdaki satırı eklemek için:

```
In [39]: # Yüklenen IPython uzantılarının noktalı modül adlarının listesi.
c.TerminalIPythonApp.extensions = ['line_profiler']
```

- Ayrıca şu komutu da çalıştırabilirsiniz:

```
In [40]: %load_ext line_profiler
```

- line\_profiler programlı olarak kullanılabilir, ancak IPython'da etkileşimli olarak kullanıldığında belki de en güçlüsüdür. Bazı NumPy dizi işlemlerini gerçekleştiren aşağıdaki koda sahip bir prof\_mod modülünüz olduğunu varsayıalım:

```
In [41]: from numpy.random import randn
```

```
In [42]: def add_and_sum(x, y):
    added = x + y
    summed = added.sum(axis=1)
    return summed
```

```
In [43]: def call_function():
    x = randn(1000, 1000)
    y = randn(1000, 1000)
    return add_and_sum(x, y)
```

- add\_and\_sum fonksiyonunun performansını anlamak istersek %prun bize şunu verir:

```
In [44]: %run prof_mod
```

```
In [45]: x = randn(3000, 3000)
```

```
In [46]: y = randn(3000, 3000)
```

```
In [47]: %prun add_and_sum(x, y)
```

```
7 function calls in 0.092 seconds
```

Ordered by: internal time

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.062	0.062	0.083	0.083	589675368.py:1(add_and_sum)
1	0.020	0.020	0.020	0.020	{method 'reduce' of 'numpy.ufunc' objects}
1	0.009	0.009	0.092	0.092	<string>:1(<module>)
1	0.000	0.000	0.092	0.092	{built-in method builtins.exec}
1	0.000	0.000	0.020	0.020	{method 'sum' of 'numpy.ndarray' objects}
1	0.000	0.000	0.020	0.020	_methods.py:47(_sum)
1	0.000	0.000	0.000	0.000	{method 'disable' of '_lsprof.Profiler' object}
s}					

- line\_profiler IPython extension etkinleştirildiğinde, yeni bir %lprun komutu kullanılabilir. Kullanımdaki tek farkımız %lprun'a hangi fonksiyon veya fonksiyonların profilini çıkarmak istediğimizi bildirmeliyiz. Genel sözdizimi şöyledir:

```
In [48]: %lprun -f func1 -f func2 statement_to_profile
```

- Bu durumda add\_and\_sum'un profilini çıkarmak istiyoruz, bu yüzden şunu çalıştırıyoruz:

```
In [573]: %lprun -f add_and_sum add_and_sum(x, y)
Timer unit: 1e-06 s
File: prof_mod.py
Function: add_and_sum at line 3
Total time: 0.045936 s
Line #      Hits          Time  Per Hit   % Time  Line Contents
=====
3           1     36510  36510.0    79.5
4           1      9425  9425.0    20.5
5           1          1     1.0     0.0
6           1
def add_and_sum(x, y):
    added = x + y
    summed = added.sum(axis=1)
    return summed
```

- Bunu yorumlamak çok daha kolay olabilir. Bu durumda, açıklamada kullandığımız fonksiyonun profilini çıkardık. Önceki modül koduna bakarak şunu çağırabiliriz: call\_function ve profile ile add\_and\_sum'un yanı sıra kodun performansının tam bir resmini elde etmek:

```

In [574]: %lprun -f add_and_sum -f call_function call_function()
Timer unit: 1e-06 s
File: prof_mod.py
Function: add_and_sum at line 3
Total time: 0.005526 s
Line #    Hits         Time  Per Hit   % Time  Line Contents
=====
3          1      4375   4375.0    79.2      def add_and_sum(x, y):
4          1      1149   1149.0    20.8      added = x + y
5          1          2      2.0      0.0      summed = added.sum(axis=1)
6          1          2      2.0      0.0      return summed
File: prof_mod.py
Function: call_function at line 8
Total time: 0.121016 s
Line #    Hits         Time  Per Hit   % Time  Line Contents
=====
8          1     57169  57169.0    47.2      def call_function():
9          1     58304  58304.0    48.2      x = randn(1000, 1000)
10         1          5543   5543.0    4.6      y = randn(1000, 1000)
11         1          5543   5543.0    4.6      return add_and_sum(x, y)

```

- Genel bir kural olarak, "makro" profil oluşturma için %prun (cProfile) ve "mikro" profil oluşturma için %lprun (line\_profiler) tercih edilebilir. Her iki aracı da iyi anlamak faydalı olacaktır.

## B.4 IPython Kullanarak Üretken Kod Geliştirmeye Yönelik İpuçları

- Kodu geliştirmeyi, hata ayıklamayı ve sonuçta etkileşimli olarak kullanmayı kolaylaştıracak şekilde yazmak, birçok kullanıcı için bir paradigma değişikliği olabilir. Kodlama stiliyle ilgili endişelerin yanı sıra bazı ayarlamalar gerektirebilecek kodun yeniden yüklenmesi.
- Bu nedenle, bu bölümde anlatılan stratejilerin ve sizin için etkili olan Python kodunu yazmanın bir yolunu belirlemek için sizin tarafınızdan bazı deneyler yapılması gerekecektir. Sonuçta kodunuza tekrar tekrar kullanmayı kolaylaştıracak ve bir program veya işlevi çalıştırmanın sonuçlarını mümkün olduğunda zahmetszizce keşfedebilecek şekilde yapılandırmak istiyorsunuz. Yalnızca bağımsız komut satırı uygulaması olarak çalıştırılması amaçlanan koda göre, IPython ile çalışmanın daha kolay olduğu düşünülerek tasarlanmış bir yazılım keşfedildi. Bu özellikle bir şeyler ters gittiğinde ve sizin veya bir başkasının aylar veya yıllar önce yazmış olabileceği bir kod hatasını teşhis etmeniz gerektiğinde önemlidir.

## Modül Bağımlılıklarını Yeniden Yükleme

- Python'da, import some\_lib yazdığınızda, some\_lib'deki kod yürütülür ve içinde tanımlanan tüm değişkenler, işlevler ve içe aktarmalar, yeni oluşturulansome\_lib modül ad alanında saklanır. Bir sonraki sefer import some\_lib yazdığınızda, mevcut modül ad alanına bir referans alacaksınız. Etkileşimli IPython kod geliştirmedeki potansiyel zorluk, örneğin değişiklik yapmış olabileceğiniz başka bir modüle bağlı olan bir script'i %çalıştırdığınızda ortaya çıkar. Diyelim ki test\_script.py'de aşağıdaki kod vardı:

```
In [53]: import some_lib  
  
x = 5  
y = [1, 2, 3, 4]  
result = some_lib.get_answer(x, y)
```

- %run test\_script.py dosyasını çalıştırıcasanız bir dahaki sefere some\_lib.py dosyasını değiştirebilirsiniz. %run test\_script.py dosyasını çalıştırıldığınızda Python'un "bir kez yükle" modül sistemi nedeniyle hala some\_lib.py'nin eski sürümünü alacaksınız. Bu davranış, kod değişikliklerini otomatik olarak yayan MATLAB gibi diğer bazı veri analizi ortamlarından farklıdır.<sup>1</sup> Bununla başa çıkmak için birkaç seçenekiniz vardır. İlk yol, standart kütüphanedeki importlib modülündeki yeniden yükleme işlevi:

```
In [54]: import some_lib  
import importlib  
  
importlib.reload(some_lib)
```

- Bu, test\_script.py'yi her çalıştırıldığınızda some\_lib.py dosyasının yeni bir kopyasını alacağınızı garanti eder. Açıktası, eğer bağımlılıklar daha derine inerse, bunu yapmak biraz zor olabilir. Bu soruna yönelik olarak IPython, modüllerin "derin" (özyinelemeli) yeniden yüklenmesi için özel bir reload işlevine (magic bir işlev değil) sahiptir. Some\_lib.py'yi çalıştırıp ardından reload(some\_lib) yazılırsa, some\_lib'i ve tüm bağımlılıklarını yeniden yüklemeye çalışacaktır. Ne yazık ki bu her durumda işe yaramayacaktır, ancak işe yaradığında IPython'u yeniden başlatmak zorunda kalmaktan daha kullanışlıdır.

## Kod Tasarımı İpuçları

- Bunun basit bir tarifi yok ama bu çalışmada etkili bulunan bazı üst düzey ilkeler.

## İlgili Nesneleri ve Verileri Canlı Tutun

- Aşağıdaki önemsiz örneğe benzer bir yapıya sahip, komut satırı için yazılmış bir program görmek alışılmadık bir durum değildir:

```
In [56]: from my_functions import g
```

```
In [57]: def f(x, y):
    return g(x + y)

def main():
    x = 6
    y = 7.5
    result = x + y

if __name__ == '__main__':
    main()
```

- Bu programı IPython'da çalıştırılırsa bazı şeyler yanlış gidebilir. Bu işlem tamamlandıktan sonra, ana fonksiyonda tanımlanan hiçbir sonuç veya nesneye IPython kabuğundan erişilmeyecektir. Daha iyi bir yol, mainexecute'da bulunan kodun doğrudan modülün genel ad alanında (veya modülün içe aktarılabilir olmasını istiyorsanız if \_\_name\_\_ == '\_\_main\_\_': bloğunda) bulunmasıdır. Bu şekilde, kodu çalıştırığınızda (%run), main'de tanımlanan tüm değişkenlere bakabileceksiniz. Bu, Jupyter not defterindeki hücrelerdeki üst düzey değişkenleri tanımlamaya eşdeğerdir.

## Düz İç İçe Olmaktan Daha İyidir

- Derinlemesine iç içe geçmiş kod, bir soğanın birçok katmanının düşünülmesi sağlanır. Bir işlevi test ederken veya hata ayıklarken, ilgilenilen koda ulaşmak için soğanın kaç katmanını soymanız gereklidir? "Düz olanın iç içe olmaktan daha iyi olduğu" fikri Python'un Zen'inin bir parçasıdır ve genellikle etkileşimli kullanıma yönelik kod geliştirme için de geçerlidir. İşlevleri ve sınıfları olabildiğince ayrık ve modüler hale getirmek, bunların test edilmesini (birim testleri yazıyorsanız), hata ayıklamayı ve etkileşimli olarak kullanılmasını kolaylaştırır.

## Daha Uzun Dosya Korkusunun Üstesinden Gelin

- Java (veya buna benzer başka bir dil) geçmişinden geliyorsanız, dosyaları kısa tutmanız söylemiş olabilir. Birçok dilde bu, sağlam bir tavsiyedir; uzun uzunluk genellikle kötü bir "kod kokusudur" ve yeniden düzenlemenin veya yeniden düzenlemenin gerekli olabileceğini gösterir. Bununla birlikte, IPython kullanarak kod geliştirirken, 10 küçük ancak birbirine bağlı dosyayla (her biri 100 satırın altında) çalışmak, genel olarak iki veya üç uzun dosyadan daha fazla baş ağrısına neden olacaktır. Daha az dosya, yeniden yüklenerek daha az modül ve düzenleme sırasında dosyalar arasında daha az geçiş anlamına gelir. Her biri yüksek iç uyuma sahip daha büyük modüllerin bakımının çok daha kullanışlı ve Pythonic olduğu fark edildi. Bir çözüme doğru ilerledikten sonra bazen büyük dosyaları daha küçük dosyalara dönüştürmek mantıklı olabilir.

- Açıkçası, bu argümanın aşırı uçlara götürülmesin desteklenmiyor; bu, tüm kodu tek bir devasa dosyaya koymak olacaktır. Büyük bir kod tabanı için mantıklı ve sezgisel bir modül ve paket yapısı bulmak çoğu zaman biraz çalışma gerektirir, ancak ekiplere doğru şekilde uyum sağlamak özellikle önemlidir. Her modül kendi içinde uyumlu olmalı ve her işlevsellik alanından sorumlu işlevlerin ve sınıfların nerede bulunacağı mümkün olduğunda açık olmalıdır.

## B.5 Gelişmiş IPython Özellikleri

- IPython sisteminden tam olarak yararlanmak, kodunuzu biraz farklı bir şekilde yazmanıza veya yapılandırmayı derinlemesine incelemenize yol açabilir.

### Kendi Sınıflarınızı IPython Dostu Haline Getirme

- IPython, incelediğiniz herhangi bir nesnenin konsol dostu dize temsilini görüntülemek için her türlü çabayı gösterir. Dicteler, listeler ve tuple'lar gibi birçok nesne için güzel biçimlendirmeyi yapmak üzere yerleşik pprint modülü kullanılır. Ancak kullanıcı tanımlı sınıflarda istediğiniz dize çıktısını kendiniz oluşturmanız gereklidir. Aşağıdaki basit sınıfı sahip olduğumuzu varsayıyalım:

```
In [58]: class Message:
    def __init__(self, msg):
        self.msg = msg
```

- Bunu yazdığınız, sınıfınızın varsayılan çıktısının pek de hoş olmadığını keşfederseniz hayal kırıklığına uğrarsınız:

```
In [59]: x = Message('I have a secret')
```

```
In [60]: x
```

```
Out[60]: <__main__.Message at 0x1d8d6f10c50>
```

- IPython, `__repr__`magic yönteminin döndürdüğü dizeyi alır (`output = repr(obj)` yaparak) ve bunu konsola yazdırır. Böylece, daha yararlı bir çıktı elde etmek için önceki sınıfı basit bir `__repr__`yöntemi ekleyebiliriz:

```
In [61]: class Message:
    def __init__(self, msg):
        self.msg = msg
    def __repr__(self):
        return 'Message: %s' % self.msg
```

```
In [62]: x = Message('I have a secret')
```

```
In [63]: x
```

```
Out[63]: Message: I have a secret
```

## Profiller ve Yapılandırma

- IPython ve Jupyter ortamlarının görünümünün (renkler, bilgi istemi, satırlar arasındaki boşluklar vb.) ve davranışının çoğu yönü, kapsamlı bir yapılandırma sistemi aracılığıyla yapılandırılabilir. Yapılandırma yoluyla yapabileceğiniz bazı şeyler şunlardır:

- Renk düzenini değiştirin
- Giriş ve çıkış istemlerinin görünüşünü değiştirin veya Out'tan sonraki ve sonraki In isteminden önceki boş satırı kaldırın
- Rastgele bir Python ifadeleri listesi yürütün (örneğin, her zaman kullandığınız içe aktarmalar veya IPython'u her başlattığınızda olmasını istediğiniz herhangi bir şey)
- line\_profiler'daki %lprun magic gibi her zaman açık IPython uzantılarını etkinleştirin
- Jupyter uzantılarını etkinleştirme
- Kendi büyülerinizi veya sistem takma adlarınızı tanımlayın
- IPython kabuğunun yapılandırmaları özel ipython\_config.py dosyalarında belirtilir, bunlar genellikle kullanıcı ana dizinizdeki .ipython/ dizininde bulunur. Yapılandırma belirli bir profile göre gerçekleştirilir. IPython'u normal şekilde başlattığınızda, varsayılan olarak profile\_default dizininde saklanan varsayılan profili yüklersiniz. Bu nedenle, Linux işletim sisteminde varsayılan IPython yapılandırma dosyamın tam yolu şöyledir:

```
/home/wesm/.ipython/profile_default/ipython_config.py
```

- Bu dosyayı sisteminizde başlatmak için terminalde çalıştırın:

```
ipython profile create
```

- Ek bir kullanıcı özellik, birden fazla profile sahip olmanın mümkün olmasıdır. Özel olarak alternatif bir IPython yapılandırmasına sahip olmak istedığınızı varsayılm. Belirli bir uygulama veya proje için. Yeni bir profil oluşturmak, aşağıdakine benzer bir şe yzmak kadar basittir:

```
ipython profile create secret_project
```

- Bunu yaptıktan sonra, yeni oluşturulan profile\_secret\_project dizinindeki yapılandırma dosyalarını düzenleyin ve ardından IPython'u şu şekilde başlatın:

```
$ ipython --profile=secret_project
Python 3.5.1 | packaged by conda-forge | (default, May 20 2016, 05:22:56)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

IPython profile: secret_project
```

- Jupyter'ın yapılandırması biraz farklı çalışır çünkü not defterlerini Python dışındaki dillerle kullanabilirsiniz. Benzer bir Jupyter yapılandırma dosyası oluşturmak için şunu çalıştırın:

```
jupyter notebook --generate-config
```

- Bu, ana dizinizdeki `.jupyter/jupyter_notebook_config.py` dizinine varsayılan bir yapılandırma dosyası yazar. Bunu ihtiyaçlarınıza uyacak şekilde düzenledikten sonra, aşağıdaki gibi farklı bir dosyayla yeniden adlandırabilirsiniz:

```
$ mv ~/.jupyter/jupyter_notebook_config.py ~/.jupyter/my_custom_config.py
```

- Jupyter'ı başlatırken `--config` argümanını ekleyebilirsiniz:

```
jupyter notebook --config=~/jupyter/my_custom_config.py
```