

```
# A) Ön bilgiler
```

Ne Tür Veriler?

"Veri" dediğimiz zaman tam olarak neyi kastediyoruz? Birincil odak noktası, aşağıdakiler gibi birçok farklı ortak veri biçimini kapsayan kasıtlı olarak belirsiz bir terim olan yapılandırılmış veriler üzerinedir

- 1- Her sütunun farklı bir türde (dize, sayısal, tarih veya başka türlü) olab ileceği tablo veya elektronik tablo benzeri veriler. Bu, ilişkisel veritabanlarında veya sekme veya virgülle sınırlanmış metin dosyalarında yaygın olarak depolanan çoğu veri türünü içerir.
- 2- Çok boyutlu diziler (matrisler).
- 3- Anahtar sütunlarla (bir SQL kullanıcısı için birincil veya yabancı anahtarlar olabilir) birbiriyle ilişkili birden çok veri tablosu.
- 4- Eşit veya düzensiz aralıklı zaman serileri.

Veri Analizi İçin Neden Python?

1991'deki ilk ortaya çıkışından bu yana, Python Perl, Ruby ve diğerleri ile birlikte en popüler yorumlanmış programlama dillerinden biri haline geldi. Bu tür diller, küçük programları veya diğer görevleri otomatikleştirmek için komut dosyalarını hızlı bir şekilde yazmak için kullanılabildikleri için genellikle komut dosyası dilleri olarak adlandırılır.

Python, çeşitli tarihsel ve kültürel nedenlerle yorumlanan diller arasında geniş ve aktif bir bilimsel hesaplama ve veri analizi topluluğu geliştirmiştir. Son 10 yılda, Python, akademi ve endüstride veri bilimi, makine öğrenimi ve genel yazılım geliştirme için en önemli dillerden biri haline gelmiştir. Python'un genel amaçlı yazılım mühendisliği için genel gücü ile birleştiğinde, veri uygulamaları oluşturmak için birincil dil olarak mükemmel bir seçenektedir.

Neden Python Değil?

Python, birçok türde analitik uygulama ve genel amaçlı sistem oluşturmak için mükemmel bir ortam olsa da, Python'un daha az uygun olabileceği bir dizi kullanım vardır.

Python yorumlanmış bir programlama dili olduğundan, genel olarak çoğu Python kodu, Java veya C++ gibi derlenmiş bir dilde yazılmış koddan önemli ölçüde daha yavaş çalışacaktır.

Programcı zamanı genellikle CPU zamanından daha değerli olduğundan, çoğu kişi bu değişim tokusu yapmaktan mutluluk duyar. Bununla birlikte, çok düşük gecikme süresi veya zorlu kaynak kullanımı gereksinimleri olan bir uygulamada (örneğin, yüksek frekanslı bir ticaret sistemi), mümkün olan maksimum performansı elde etmek için C++ gibi daha düşük seviyeli (ancak aynı zamanda dah

a düşük verimli) bir dilde programlama için harcanan zaman iyi harcanan zaman olabilir.

Python, yüksek düzeyde eşzamanlı, çok iş parçacıklı uygulamalar, özellikle birçok CPU'ya bağlı iş parçacığı içeren uygulamalar oluşturmak için zorlu bir dil olabilir.

Temel Python Kütüphaneleri

1-NumPy

Numerical Python'un kısaltması olan NumPy, uzun zamandır Python'da sayısal hesaplamalarda kullanılır. Python'da sayısal verileri içeren çoğu bilimsel uygulama için gereken veri yapılarını, algoritmaları sağlar. NumPy, diğer şeylerin yanı sıra şunları içerir:

- Hızlı ve verimli bir çok boyutlu dizi nesnesi
- Diziler veya matematiksel işlemler ile eleman bazlı hesaplamalar yapmak için işlevler
 - Dizi tabanlı veri kümelerini okumak ve diske yazmak için araçlar
 - Doğrusal cebir işlemleri, Fourier dönüşümü ve rastgele sayı üretimi
 - Python uzantılarının ve yerel C veya C ++ kodunun NumPy'nin veri yapılarına ve hesaplama tesislerine

erişmesini sağlayan olgun bir C API

NumPy'nin Python'a eklediği hızlı dizi işleme yeteneklerinin ötesinde, veri analizindeki birincil kullanımlarından biri, algoritmalar ve kütüphaneler arasında veri aktarımını sağlar. Sayısal veriler için NumPy dizileri, verileri depolamak ve işlemek için diğer yerleşik Python veri yapılarına göre daha etkilidir.

2- Pandas

Pandas, yapılandırılmış veya tablo şeklindeki verilerle çalışmayı hızlı, kolay ve anlamlı hale getirmek için tasarlanmış üst düzey veri yapıları ve işlevleri sağlar

Pandas, NumPy'nin yüksek performanslı, dizi hesaplama fikirlerini elektronik tabloların ve ilişkisel veritabanlarının (SQL gibi) esnek veri işleme yetenekleriyle ilişkilidir. Yeniden şekillendirmeyi, dilimlemeyi ve parçalara ayırmayı, toplamaları gerçekleştirmeyi ve veri alt kümelerini seçmeyi kolaylaştırılmak için gelişmiş işlemleri sağlar. Veri manipülasyonundan bu yana, veri analizinde hazırlık ve temizlik çok önemli bir beceridir.

3- Matplotlib

Matplotlib, grafikler ve diğer iki boyutlu veri görselleştirmeleri üretmek için kullanılan Python kütüphanesidir.

4- Scipy

SciPy, bilimsel hesaplamada bir dizi farklı standart problem alanını ele alan bir paketleri barındıran bir kütüphanedir. Örnek verecek olursak:

a)scipy.integrate

Sayısal entegrasyon rutinleri ve diferansiyel denklem çözüçüler

b)scipy.linalg

NumPy.linalg'de sunulanların ötesine uzanan doğrusal cebir rutinleri ve localhost:8888/notebooks/tasarim_rapor.ipynb

matris ayırtmaları

c)scipy.optimize

İşlev iyileştirciler (küçülticiler) ve kök bulma algoritmaları

d)scipy.signal

Sinyal işleme araçları

e)scipy.sparse

Seyrek matrisler ve seyrek doğrusal sistem çözücüleri

f)scipy.special

Gama işlevi gibi birçok yaygın matematiksel işlevi uygulayan bir Fortran kitaplığı olan SPECFUN'un etrafına sarıcı

g)scipy.stats

Standart sürekli ve ayrık olasılık dağılımları (yoğunluk fonksiyonları, örnekleyiciler, sürekli dağılım fonksiyonları), çeşitli istatistiksel testler ve daha açıklayıcı istatistikler

5- Scikit-Learn

- Classification: SVM, en yakın komşular, rastgele orman, lojistik regresyon vb.
- Regression: Kement, sırt gerilemesi vb.
- Clustering: k-means, spektral kümeleme, vb.
- Dimensionality reduction: PCA, özellik seçimi, matris çarpanlara ayırma vb.
- Model selection: Izgara arama, çapraz doğrulama, metrikler
- Preprocessing: Özellik çıkarma, normalleştirme

Python Paketlerini Yükleme veya Güncelleme

conda install paket_adi

Bu işe yaramazsa, paketi pip paketini kullanarak da kurabilirsiniz. pip install paket_adi Conda update komutunu kullanarak paketleri güncelleyebilirsiniz : Conda update package_name pip ayrıca --upgrade flag kullanan yükseltmeleri de destekler : pip install --upgrade paket_adi

Python Kütüphaneleri İçe Aktarma Kuralları

- import numpy as np
- import matplotlib.pyplot as plt
- import pandas as pd
- import seaborn as sns
- import statsmodels as sm

B) Python Dilinin Temelleri

Python Yorumlayıcı

Python yorumlanmış bir dildir. Python yorumlayıcısı, her seferinde bir ifade çalıştırarak bir programı çalıştırır. Standart etkileşimli Python yorumlayıcısı, python komutuyla komut satırından çağrılabılır

Değişken tanımlamak ve onu ekrana yazdırınmak için aşağıdaki komutlar çalıştırılır.

In [2]:

```
a=5  
print(a)
```

5

Değişkeni ekrana yazdırınmak için print() fonksiyonu kullanılır.

In [3]:

```
print('Hello World')
```

Hello World

Jupyter'da var olan bir Python dosyasını çalıştırınmak için *run dosyaismi* komutu kullanılır.(dosya mevcut çalışma terminalinizde olmalıdır)

In [12]:

```
run /Users/veyseldogan/Downloads/tasarim_rapor.py
```

5
Hello World

İsteğe bağlı Python ifadelerini yazıp Enter tuşuna basarak çalıştırabilirsiniz

In [13]:

```
a=5
```

In [14]:

```
a
```

Out[14]:

5

Alt taraftaki kod bloğunda rastgele sayılar üretmek için *satır 12*'de numpy kütüphanesini çalışma ortamımıza dahil ettik ve daha sonra np olarak ismini kısalttık

Satır 14'te data değişkenine numpy kütüphanesinin random.rand() fonksiyonunu kullanarak bizim için 7 tane rastgele değerler üretmesini sağladık. Bir alt satırda üretilen değerleri görebilirsiniz.

In [16]:

```
import numpy as np
data = {i : np.random.randn() for i in range(7)}
data
```

Out[16]:

```
{0: -1.3838656170876116,
 1: -0.7333500242760296,
 2: 0.5653514843382652,
 3: -0.17815368324275532,
 4: 0.0856441824930693,
 5: 0.20658474841089627,
 6: 1.6412016455359335}
```

Birçok Python nesnesi türü, normal baskıdan farklı olarak, daha okunabilir veya güzel basılmış olacak şekilde biçimlendirilmiştir. Yukarıdaki veri değişkenini standart Python yorumlayıcısında yazdırırsanız, çok daha az okunabilir olacaktır

In [17]:

```
from numpy.random import rand
print(data)
```

```
{0: -1.3838656170876116, 1: -0.7333500242760296, 2: 0.565351484338265
2, 3: -0.17815368324275532, 4: 0.0856441824930693, 5: 0.20658474841089
627, 6: 1.6412016455359335}
```

Jupyter Notebook İle Çalışmak

Yeni bir defteri oluşturmak için Yeni düğmesine tıklayın ve "Python 3" veya "conda [varsayılan]" seçeneğini seçin. İlk kez yapıyorsanız, " hücre" boş koduna tıklayıp bir Python kodu satırı girmeyi deneyin. Ardından yürütmek için Shift-Enter tuşlarına basın.

Not defterini kaydettiğinizde, .ipynb uzantılı bir dosya oluşturulur. Bu, şunda not defterinde bulunan tüm içeriği (değerlendirilen kod çıktıları dahil) içeren bağımsız bir dosya biçimidir. Bunlar diğer Jupyter kullanıcıları tarafından yüklenebilir ve düzenlenebilir.

Sekme Tamamlama

İfadeler girerken, Tab tuşuna basmak, şimdije kadar yazdığınız karakterlerle eşleşen tüm değişkenler (nesneler, işlevler vb.) için ad alanında arama yapacaktır.

In [101]:

```
from IPython.display import Image  
Image(filename='img/Picture1.png')
```

Out[101]:

In [21]: an_apple=27
an_example=42

In [20]: an

In []: an_apple
an_example
any

Bu örnekte, tanımlanan iki değişkeni hem de Python anahtar sözcüğünü ve yerleşik işlevi herhangi birini gösterdiğine dikkat edin. Bir nokta yazdıktan sonra herhangi bir nesne üzerindeki yöntemleri ve nitelikleri tamamlayabilirsiniz.

In [9]:

```
Image(filename='img/Picture2.png')
```

Out[9]:

In [22]: b=[1,2,3]

In []: b.

In []: append
clear
copy
count
extend
index
insert
pop
remove
reverse

Aynısı modüller için de geçerlidir.

In [10]:

Image(filename='img/Picture3.png')

Out[10]:

```
In [23]: import datetime

In [ ]: datetime.

In [ ]: 
```

date
datetime
datetime_CAPI
MAXYEAR
MINYEAR
sys
time
timedelta
timezone
tzinfo

Sekme tamamlama, etkileşimli ad alanı aramanın ve nesne veya modül niteliklerini tamamlamanın dışında birçok bağlamda çalışır. Dosya yoluna benzeyen herhangi bir şey yazarken (bir Python dizesinde bile), Sekme tuşuna basmak bilgisayarınızın dosya sistemindeki her şeyi yazdıklarınızla eşleşen tamamlayacaktır.

In [11]:

Image(filename='img/Picture4.png')

Out[11]:

```
In [7]: datasets/movielens/<Tab>
datasets/movielens/movies.dat    datasets/movielens/README
datasets/movielens/ratings.dat   datasets/movielens/users.dat
```

```
In [7]: path = 'datasets/movielens/<Tab>
datasets/movielens/movies.dat    datasets/movielens/README
datasets/movielens/ratings.dat   datasets/movielens/users.dat'
```

Sekme tamamlamanın zaman kazandırdığı başka bir alan, işlev anahtar kelime argümanları (ve = işaretti dahil!)

In [33]:

```
def func_with_keywords(abra=1, abbra=2, abbbra=3):
    return abra, abbra, abbbra
```

In [12]:

```
Image(filename='img/Picture5.png')
```

Out[12]:

```
In [ ]: func_with_keywords(ab)
In [ ]: abbbra=
In [ ]: abbra=
In [ ]: abra=
In [ ]: abs
```

Nesne İç Gözlemi

Bir değişkenden önce veya sonra bir soru işaretçi (?) Kullanmak, nesne hakkında bazı genel bilgileri gösterir.

In [36]:

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

In [37]:

```
b?
```

In [5]:

```
from IPython.display import Image
Image(filename='img/Picture6.png')
```

Out[5]:

```
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.
```

In [40]:

```
print?
```

In [13]:

Image(filename='img/Picture7.png')

Out[13]:

```

Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type: builtin_function_or_method

```

Nesne iç gözleme(as object introspection), nesne bir işlev veya örnek yöntem ise, tanımlanmışsa, docstring de gösterilecektir.

In [43]:

```

def add_numbers(a,b):
    """
    Add two numbers together
    Returns
    -----
    the_sum : type of arguments
    """
    return a+b

```

In [44]:

add_numbers?

In [14]:

Image(filename='img/Picture8.png')

Out[14]:

```

Signature: add_numbers(a, b)
Docstring:
"
Add two numbers together
Returns
-----
the_sum : type of arguments
File:      ~/<ipython-input-43-c442b52e1239>
Type:      function

```

?? kullanmak aynı zamanda kaynak kodunu da bilgi olarak verecektir.

In [46]:

```
add_numbers??
```

In [15]:

```
Image(filename='img/Picture9.png')
```

Out[15]:

```
Signature: add_numbers(a, b)
Source:
def add_numbers(a,b):
    """
    Add two numbers together
    Returns
    -----
    the_sum : type of arguments
    """
    return a+b
File:      ~/<ipython-input-43-c442b52e1239>
Type:      function
```

Joker karakterle (*) birleştirilen birkaç karakter, joker ifadesiyle eşleşen tüm adları gösterecektir. Örneğin, load içeren top-level NumPy namespace tüm işlevlerin bir listesini alabiliriz.

In [48]:

```
np.*load*?
```

In [16]:

```
Image(filename='img/Picture10.png')
```

Out[16]:

```
np.__loader__
np.load
np.loads
np.loadtxt
```

%run Komutu

%Run komutunu kullanarak herhangi bir dosyayı bir Python programı olarak çalıştırabilirsiniz

run__command.ipynb dosyasında aşağıdaki komutları içeren bir yapı olduğunu varsayıyalım.
(run__command.ipynb dosyasında resimdeki kodlara erişebilirsiniz.)

In [17]:

Image(filename='img/Picture11.png')

Out[17]:

```
In [1]: def f(x,y,z):
    return(x+y)/z
```

```
In [2]: a=5
b=6
c=7.5
result = f(a,b,c)
```

```
In [ ]:
```

In []:

run run_command.ipynb ##artık run_command dosyası içindeki değişkenlere ulaşabilir

In [55]:

c

Out[55]:

7.5

In [56]:

result

Out[56]:

1.4666666666666666

Panodan Kod Yürütme

Jupyter not defterinde, kodu herhangi bir kod hücresine kopyalayıp yapıştırabilir ve çalıştırabilirsiniz.

En kusursuz yöntemler, % paste ve % cpaste büyü işlevleridir. %paste panodaki metni alır ve onu tek bir blok olarak yürütür.

Terminal Klavye Kısıyolları

Ctrl-P veya yukarı-ok = Ö anda girilen metinle başlayan komutlar için komut geçmişinde geriye doğru arama
 Ctrl-N veya aşağı ok = Ö anda girilen metinle başlayan komutlar için komut g
eçmişinde ileriye doğru arama
 Ctrl-R = Read line tarzı ters geçmiş arama (kısmi eşleme)
 Ctrl-Shift-V = Panodaki metni yapıştır
 Ctrl-C = Şu anda yürütülen kodu kes
 Ctrl-A = İmleci satırın başına taşı
 Ctrl-E = İmleci satırın sonuna taşı
 Ctrl-K = İmleçten satır sonuna kadar metni sil
 Ctrl-U = Geçerli satırdaki tüm metni sil
 Ctrl-F = İmleci bir karakter ileri taşı
 Ctrl-B = İmleci bir karakter geri taşı
 Ctrl-L = Ekranı temizle

Magic Commands Hakkında

Bunlar, genel görevleri kolaylaştırmak ve sisteminin davranışını kolayca kontrol etmenizi sağlamak için tasarlanmıştır. Bir magic command, yüzde sembolü ile başlayan herhangi bir komuttur. Örneğin, matris çarpımı gibi herhangi bir Python ifadesinin yürütme zamanını %timeitmagic işlevini kullanarak kontrol edebilirsiniz.

In [61]:

```
a = np.random.randn(100, 100)
```

In [62]:

```
%timeit np.dot(a, a)
```

```
20.5 µs ± 802 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [63]:

```
%debug?
```

In [64]:

```
%pwd
```

Out[64]:

```
'/Users/veyseldogan'
```

Söz konusu Magic Command ile aynı adı taşıyan hiçbir değişken tanımlanmadığı sürece, Magic Command varsayılan olarak yüzde işaretini olmadan kullanılabilir. Bu özelliğe automagic denir ve % automagic ile etkinleştirilebilir veya devre dışı bırakılabilir.

In [65]:

foo=%pwd

In [66]:

foo

Out[66]:

'/Users/veyseldogan'

Sık kullanılan bazı Magic Command'lar

```
%quickref = IPython Hızlı Referans Kartını görüntüleyin
%magic = Mevcut tüm magic commands için ayrıntılı belgeleri görüntüleyin
%debug = Son istisna takibinin altındaki etkileşimli hata ayıklayıcıyı girin
%hist = Komut girişi (ve istege bağlı olarak çıktı) geçmişini yazdır
%pdb = Herhangi bir istisnadan sonra otomatik olarak hata ayıklayıcıyı girin
%paste = Panodan önceden biçimlendirilmiş Python kodunu yürütün
%cpaste = Çalıştırılacak Python kodunu manuel olarak yapıştırmak için özel bir istem açın
%reset = Etkileşimli ad alanında tanımlanan tüm değişkenleri / adları silin
%page OBJECT = Nesneyi güzelce yazdırın ve bir çağrı cihazıyla görüntüleyin
%run script.py = IPython içinde bir Python dosyası çalıştırın
%prun statement = İfadeyi cProfile ile yürütün ve profil oluşturucu çıktısını rapor edin
%time statement = Tek bir ifadenin yürütme zamanını bildirin
%timeit statement = Topluluk ortalama yürütme süresini hesaplamak için bir döymeli birden çok kez çalıştırın; çok kısa yürütme süreli zamanlama kodu için kullanılabilir
%who, %who_ls, %whos = Çeşitli bilgi / ayrıntı düzeyleriyle etkileşimli ad alanında tanımlanan değişkenleri görüntüleyin
%xdel variable = Bir değişkeni silin ve IPython dahili dosyalarında nesneye yapılan tüm başvuruları silmeye çalışın
```

Matplotlib Entegrasyonu

%Matplotlib Magic Command, IPython kabuğu veya Jupyter dizüstü bilgisayar ile entegrasyonunu yapılandırır. Bu önemlidir, aksi takdirde oluşturduğunuz grafikler görünmez (not defteri) veya kapanana kadar (kabuk) oturumun kontrolünü ele geçirmez.

%matplotlib'i çalıştırın, entegrasyonu ayarlayın, böylece konsol oturumuna müdahele etmeden birden çok çizim penceresi oluşturabilirsiniz.

In [73]:

%matplotlib

Using matplotlib backend: MacOSX

In [74]:

```
%matplotlib inline
```

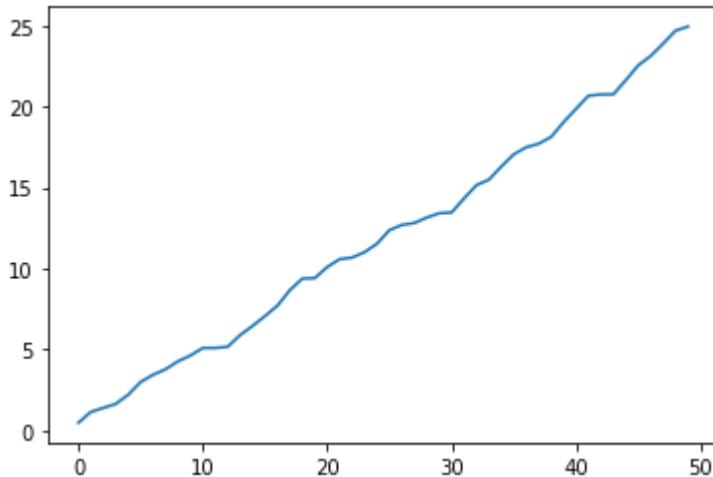
Jupyter satır içi matplotlib çizimi

In [75]:

```
import numpy as np
import matplotlib.pyplot as plt
plt.plot(np.random.rand(50).cumsum())
```

Out[75]:

```
[<matplotlib.lines.Line2D at 0x7fae8a4afc90>]
```



Python Dilinin Temelleri

1- Dil Anlambilimi

Python dil tasarımı, okunabilirlik, basitlik ve açıklığa vurgu yapmasıyla ayırt edilir. "çalıştırılabilir sözde kod" da denebilir.

2- Girintiler ile çalışır

Python, R, C++, Java ve Perl gibi diğer birçok dilde olduğu gibi parentez kullanmak yerine kodu yapılandırmak için boşluk (sekmeler veya boşluklar) kullanır. Örneğin:

```
for x in array:
    if x < pivot:
        less.append(x)
    else:
        greater.append(x)
```

İki nokta üst üste, girintili bir kod bloğunun başlangıcını belirtir, bundan sonra tüm kod bloğun sonuna kadar aynı miktarda girintilenmelidir.

Şimdiye kadar görebileceğiniz gibi, Python ifadelerinin de noktalı virgülle sonlandırılmasına gerek yoktur. Bununla birlikte, tek bir satırdaki birden

çok ifadeyi ayırmak için noktalı virgül kullanılabilir. Örneğin:

```
a = 5; b = 6; c = 7
```

Ancak bu yanlış bir kullanımıdır. Çünkü, kodu daha az okunaklı hale getirir.

3- Her şey bir nesnedir

Python dilinin önemli bir özelliği, nesne modelinin tutarlılığıdır. Her sayı, dizi, veri yapısı, işlev, sınıf, modül ve benzeri, Python yorumlayıcıında, Python nesnesi olarak adlandırılır. Her nesnenin ilişkili bir türü ve dahili verileri vardır. Pratikte bu, dilin çok esnek olmasını sağlar, çünkü işlevler bile başka herhangi bir nesne gibi ele alınabilir.

4- Yorum Satırları

Önünde kare işaretti (#) bulunan herhangi bir metin Python derleyicisi tarafından yok sayılır. Bu genellikle koda yorum eklemek için kullanılır. Bazen belirli kod bloklarını silmeden hariç tutmak istenebilir. Bu yüzden kod bloğu yorum satırı haline getirilebilir. Örnek aşağıda belirtilmiştir.

```
•results = []
for line in file_handle:
    # şimdilik boş satırları koru
    # if len(line) == 0:
    #     devam
    results.append(line.replace('foo', 'bar'))

•print("Reached this line") # Basit durum raporu
```

In [6]:

```
print("Reached this line") # Basit durum raporu
```

Reached this line

5- Fonksiyon ve Nesne Metod Çağrısını

Parantez kullanarak ve sıfır veya daha fazla argüman ileterek, istege bağlı olarak döndürülen değeri bir değişkene atayarak değişkenleri çağırırsınız. Örneğin:

```
result = f(x, y, z)
g()
```

Fonksiyonlar hem konumsal hem de anahtar kelime argümanlarını alabilir.

```
result = f(a, b, c, d=5, e='foo')
```

6- Değişkenler ve Argümanlar

Python'da bir değişken (veya isim) atarken, eşittir işaretinin sağ tarafındaki nesneye bir referans oluşturursunuz.

In [11]:

```
a=[1,2,3]
```

a'yi yeni bir değişken olan b'ye atayalım

In [12]:

```
b=a
```

Python'da, a ve b artık aynı nesneye, orijinal listeye [1, 2, 3] atıfta bulunur. Bunu, a'ya bir öğe ekleyerek ve ardından b'yi inceleyerek kanıtlayabilirsiniz.

In [13]:

```
a.append(4)
```

In [14]:

```
b
```

Out[14]:

```
[1, 2, 3, 4]
```

Göründüğü üzere a değişkenine 4 elemanını eklenmesine rağmen, b değişkeni orijinal listeye atıfta bulunduğu için, b değişkeninin yeni listesi [1, 2, 3, 4] oldu.

Yeni bir nesneyi bir fonksiyonun içindeki değişkene atarsanız, bu değişiklik üst kapsama yansıtılmayacaktır. Bu nedenle, değişimlerin içsel özelliklerini değiştirmek mümkündür.

In [15]:

```
def append_element(some_list, element): some_list.append(element)
```

In [16]:

```
data = [1, 2, 3]
```

In [17]:

```
append_element(data, 4)
```

In [18]:

data

Out[18]:

[1, 2, 3, 4]

7- Dinamik Referanslar, Güçlü Tipler

Değişkenler, belirli bir nesnelerin adlarıdır; tür bilgisi(int, str...) nesnenin kendisinde saklanır

In [19]:

a = 5

In [20]:

type(a)

Out[20]:

int

In [21]:

a = 'foo'

In [22]:

type(a)

Out[22]:

str

Aynı tipe sahip olmayan nesnelerde, matematik operatörleriyle işlem yapılamaz. Eğer '5'+5 toplamaya çalışırsak aşağıdaki hata çıktısı görünecektir

In [18]:

```
from IPython.display import Image
Image(filename='img/Picture12.png')
```

Out[18]:

In [24]: '5'+5

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-24-e84694abbbbf> in <module>  
----> 1 '5'+5  
  
TypeError: can only concatenate str (not "int") to str
```

Python, güçlü bir şekilde yazılmış bir dil olarak kabul edilir, yani her nesnenin belirli bir türü (veya sınıfı) vardır ve dönüştürmeler yalnızca aşağıdakiler gibi belirli belli durumlarda gerçekleşir.

In [30]:

```
a = 4.5
```

In [31]:

```
b = 2
```

In [32]:

```
print('a is {}, b is {}'.format(type(a), type(b)))
```

```
a is <class 'float'>, b is <class 'int'>
```

In [33]:

```
a/b
```

Out[33]:

```
2.25
```

isinstance fonksiyonunu kullanarak bir nesnenin belirli bir türün örneği olup olmadığını kontrol edebilirsiniz.

In [34]:

```
a = 5
```

In [37]:

```
isinstance(a, int) # a değişkeninin tipi int olduğu için, bize True değeri döndürecek
```

Out[37]:

```
True
```

isinstance fonksiyonu ile birden fazla değişken tipini kontrol edebilirsiniz.

In [38]:

```
a=5;b=4.5
```

In [39]:

```
isinstance(a,(int, float))
```

Out[39]:

```
True
```

In [40]:

```
isinstance(b,(int, float))
```

Out[40]:

```
True
```

8- Attributler and Methodlar

Python'daki nesneler tipik olarak hem Attributlere (nesnenin "içinde" de polanan diğer Python nesnelerine) hem de Metodlara (nesnenin dahili verileri ne erişebilen bir nesneye ilişkili işlevler) sahiptir.

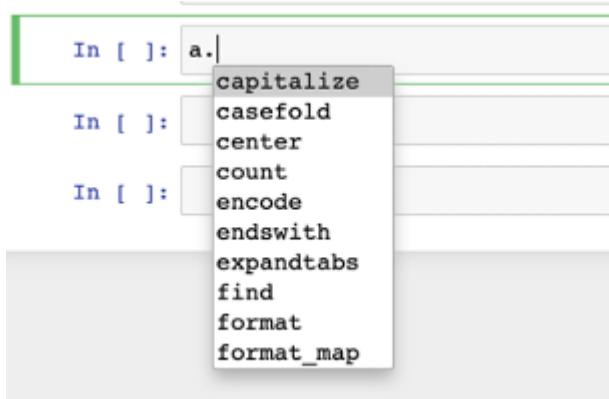
In [41]:

```
a = 'foo'
```

In [19]:

```
Image(filename='img/Picture13.png')
```

Out[19]:



a değişkeni için kullanılabilen Attributler'e a.Tab şeklinde ulaşılabilir.

a değişkeni için tüm Attributler aşağıda listelenmiştir.

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

Attributler ve Metodlara `getattr` fonksiyonu aracılığıyla da erişilebilir.

In [45]:

```
getattr(a, 'split')
```

Out[45]:

```
<function str.split(sep=None, maxsplit=-1)>
```

9- Duck Typing

Bilgisayar programlamasında Duck Typing bir yazılım kavramıdır, bir nesnenin belirli bir amaç için kullanılıp kullanılamayacağını belirlemek için bir uygulamadır. Kavramın adı Ördek testinden gelmektedir testin mantığı kısaca "Ördek gibi yürüyorsa ve ördek gibi vaklıyorsa, o zaman bir ördek olmalıdır".

r" cümlesiidir.

Örneğin, iterator(yineleme) protokolünü uygulayan bir nesnenin yinelenebilir olduğunu doğrulayabilirsiniz. Birçok nesne için bu, `__iter__` "magic method" olduğu anlamına gelir, ancak kontrol etmenin alternatif ve daha iyi bir yolu iter fonksiyonunu kullanmaktır.

In [52]:

```
def is iterable(obj):
    try:
        iter(obj)
        return True
    except TypeError: # not iterable
        return False # Bu fonksiyon dizeler için ve çoğu Python koleksiyonu türü için
```

In [48]:

```
is iterable('a string')
```

Out[48]:

True

In [49]:

```
is iterable([1, 2, 3])
```

Out[49]:

True

In [50]:

```
is iterable(5)
```

Out[50]:

False

Bu fonksiyon birden çok türde girişi kabul edebilen fonksiyonlar yazmak için kullanılabilir. Önce nesnenin bir liste (veya bir NumPy dizisi) olup olmadığıını kontrol edebilir ve değilse, onu bir olacak şekilde dönüştürebilirsiniz.

Örneğin:

```
if not isinstance(x, list) and is iterable(x):
    x = list(x)
```

10- Imports

Python'da bir modül, Python kodunu içeren .py uzantısına sahip bir dosyadır. Aşağıdaki modüle sahip olduğumuzu varsayıyalım(# some_module.ipynb dosyası çalışma ağacı altında mevcuttur).

```
# some_module.ipynb
```

Some_module.py içinde tanımlanan değişkenlere ve işlevlere erişmek istersek import kodunu kullanabiliriz.

```
import some_module
result = some_module.f(5)
pi = some_module.PI
```

Veya

```
from some_module import f, g, PI
result = g(5, PI)
```

Veya as anahtarı ile farklı bir isim vererek

```
import some_module as sm
from some_module import PI as pi, g as gf

r1 = sm.f(pi)
r2 = gf(6, pi)
```

11- İkili Operatörler ve Karşılaştırmalar

In [73]:

```
5 - 7
```

Out[73]:

-2

In [71]:

```
12 + 21.5
```

Out[71]:

33.5

In [72]:

```
5 <= 2
```

Out[72]:

False

İki referansın aynı nesneye atıfta bulunup bulunmadığını(a = b) kontrol etmek için *is* anahtar sözcüğünü kullanın

In [74]:

```
a = [1, 2, 3]
```

In [75]:

```
b = a # b değişkenini a değişkenine eşitleyerek, aynı nesneye atıfta bulunmasını sağla
```

In [77]:

```
c = list(a)
```

In [78]:

```
a is b # aynı nesneye atıfta bulundukları için bize True değerini döndürecek tır.
```

Out[78]:

True

In [82]:

```
a is not c # c değişkeni, a değişkenini kopyaladığı için, doğrudan a değişkenine atı
```

Out[82]:

True

list her zaman yeni bir Python listesi (yani bir kopya) oluşturduğundan, c'nin a'dan farklı olduğundan emin olabiliriz. is ile karşılaştırmak == operatörü ile aynı şey değildir.

In [83]:

```
a == c
```

Out[83]:

True

None değişkeni, değişkenin yok olup olmadığını kontrol etmek için kullanılır.

In [84]:

```
a = None
```

In [85]:

```
a is None
```

Out[85]:

True

İkili operatörler

a+b = a ya b ekle
 a-b = a dan b çıkar
 a*b = a ile b'yi çarp
 a/b = a yi b ye böl
 a // b = Herhangi bir kesirli kalanı düşürerek yere bölme
 a ** b = a nin b kuvveti
 a & b = Hem a hem de b Doğru ise doğrudur tamsayılar için bitsel AND'yi alın
 a | b = A veya b True ise doğrudur; tamsayılar için bitsel OR
 a^b = Boole'lar için, a veya b Doğru ise Doğru, ancak ikisi birden değil; tamsayılar için bitsel EXCLUSIVE-OR

a == b = a, b'ye eşitse doğrudur
 a != b = a, b'ye eşit değilse doğrudur
 a <= b, a < b = a, b'den küçükse (küçük veya eşitse) doğrudur
 a > b, a >= b = a, b'den büyükse (büyük veya eşitse) doğrudur
 a is b = a ve b aynı Python nesnesine referans gösteriyorsa doğrudur
 a is not b = a ve b aynı Python nesnesine referans göstermiyorsa doğrudur

12- Değişebilir ve değişemez nesneler

Python'daki listeler, NumPy dizileri ve kullanıcı tanımlı türler (sınıflar) gibi nesnelerin çoğu değiştirilebilir. Bu, içerdikleri nesne veya değerlerin değiştirileceği anlamına gelir

In [86]:

```
a_list = ['foo', 2, [4, 5]]
```

In [87]:

```
a_list[2] = (3, 4)
```

In [88]:

```
a_list
```

Out[88]:

```
['foo', 2, (3, 4)]
```

In [89]:

```
a_tuple = (3, 5, (4, 5))
```

In [20]:

Image(filename='img/Picture14.png')

Out[20]:

```
In [90]: a_tuple[1] = 'four'

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-90-23fe12dalba6> in <module>
----> 1 a_tuple[1] = 'four'

TypeError: 'tuple' object does not support item assignment
```

Bir fonksiyon yazarken, herhangi bir yan etki, işlevin dokümantasyonunda veya yorumlarında kullanıcıya açıkça bildirilmelidir. Mümkünse, değişken nesneler olsa bile, yan etkilerden kaçınmak ve değişmezliği desteklemek gereklidir.

13- Skaler Türler

Python, standart kütüphanesi birlikte sayısal verileri, dizeleri, boolean (Doğru veya Yanlış) değerleri ve tarih ve saat'i işlemek için küçük bir yerlesik tür kümesine sahiptir. Bu "tek değer" türlerine bazen skaler türler denir.

Standart Python skaler türleri

```
None = Python "boş" değeri
str = Dize türü; Unicode (UTF-8 kodlu) dizelerini tutar
bytes = Ham ASCII baytları (veya bayt olarak kodlanmış Unicode)
float = Çift duyarlıklı (64-bit) ondalık sayı
bool = Doğru veya Yanlış değer
int = Tamsayı
```

14- Sayısal türler

Sayılar için birincil Python türleri int ve float'tır. Bir int, rastgele büyük sayıları depolayabilir.

In [92]:

ival = 17239871

In [93]:

ival ** 6

Out[93]:

26254519291092456596965462913230729701102721

Ondalık sayılar, Python float türü ile temsil edilir. Her biri çift hassasiyetli (64 bit) bir değerdir. Ayrıca bilimsel gösterimle de ifade edilebilirler.

```
r
```

In [94]:

```
fval = 7.243
```

In [95]:

```
fval2 = 6.78e-5
```

Tam sayı ile sonuçlanmayan tamsayı bölümü her zaman ondalık sayı verir.

In [96]:

```
3/2
```

Out[96]:

```
1.5
```

Tam sayı bölme elde etmek için // operatörü kullanılır.

In [97]:

```
3//2
```

Out[97]:

```
1
```

15- Strings

In [98]:

```
a = 'one way of writing a string'  
b = "another way"
```

Satır sonu içeren çok satırlı dizeler için, '''veya "" " olmak üzere üçlü tırnak kullanabilirsiniz.

In [101]:

```
c = """  
This is a longer string that  
spans multiple lines  
"""
```

c dizesi 4 satırlık metin içermektedir. "" " ve sonraki satırlar dizeye dahil edilir.

In [102]:

```
c.count('\n')
```

Out[102]:

```
3
```

Python dizeleri değişmezdir. Bir dizeyi değiştiremezsiniz.

In [103]:

```
a = 'this is a string'
```

In [21]:

```
Image(filename='img/Picture15.png')
```

Out[21]:

```
In [104]: a[10] = 'f'
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-104-2151a30ed055> in <module>
      1 a[10] = 'f'
----> 1 TypeError: 'str' object does not support item assignment
```

In [106]:

```
b = a.replace('string', 'longer string')
```

In [107]:

```
b
```

Out[107]:

```
'this is a longer string'
```

Bu işlemden sonra, a değişkeni değişmiş gibi gözükebilir, ancak değişimmemiştir.

In [108]:

```
a
```

Out[108]:

```
'this is a string'
```

Birçok Python nesnesi, str islevi kullanılarak bir dizeye dönüştürülebilir.

In [109]:

```
a=5.6
```

In [110]:

```
s = str(a)
```

In [111]:

```
print(s)
```

5.6

Dizeler, Unicode karakterlerinin bir dizisidir ve bu nedenle, listeler ve tuples gibi diğer diziler gibi ele alınabilir.

In [112]:

```
s = 'python'
```

In [113]:

```
list(s)
```

Out[113]:

```
['p', 'y', 't', 'h', 'o', 'n']
```

In [114]:

```
s[:3] # S[:3] sözdizimi parçalama olarak adlandırılır ve birçok Python dizisi türü i
```

Out[114]:

```
'pyt'
```

Ters eğik çizgi karakteri \ bir çıkış karakteridir, yani satırsonu \n veya Unicode karakterleri gibi özel karakterleri belirtmek için kullanılır. Ters eğik çizgi içeren bir dize yazmak için onlardan kaçmanız gereklidir.

In [115]:

```
s = '12\\34'
```

In [116]:

```
print(s)
```

12\34

Çok fazla ters eğik çizgi içeren bir dizeniz varsa ve hiçbir özel karakterin iz yoksa, dizenin başındaki alıntıya r ile başlayabilirisiniz, bu da karakterlerin şu şekilde yorumlanması gerektiği anlamına gelir

In [119]:

```
s = r'this\has\no\special\characters' # R, ham anlamına gelir.  
# İki dizeyi birbirine eklemek onları birleştirir ve yeni bir dize oluşturur.
```

In [120]:

s

Out[120]:

'this\\has\\no\\special\\characters'

In [121]:

a = 'this is the first half '

In [122]:

b = 'and this is the second half'

In [123]:

a+b

Out[123]:

'this is the first half and this is the second half'

Dize nesneleri, biçimlendirilmiş bağımsız değişkenleri dizeye değiştirmek için kullanılabilen ve yeni bir dize üreten bir biçim yöntemine sahiptir.

In [124]:

template = '{0:.2f} {1:s} are worth US\${2:d}'

- {0: .2f}, ilk bağımsız değişkeni iki ondalık basamaklı bir sayı olarak biçimlendirmek anlamına gelir.
- {1: s}, ikinci bağımsız değişkeni bir dizge olarak biçimlendirmek anlamına gelir.
- {2: d}, üçüncü bağımsız değişkeni tam bir tam sayı olarak biçimlendirmek anlamına gelir.

In [125]:

template.format(4.5560, 'Argentine Pesos', 1)

Out[125]:

'4.56 Argentine Pesos are worth US\$1'

Python'un eski sürümlerinde, dizelerin tümü, açık Unicode kodlaması olmadan baytı. Karakter kodlaması ile Unicode'a dönüştürebilirsiniz.

In [126]:

```
val = "español"
```

In [127]:

```
val
```

Out[127]:

```
'español'
```

Bu Unicode dizesini UTF-8 bayt gösterimine şu komutu kullanarak dönüştürebiliriz.

In [128]:

```
val_utf8 = val.encode('utf-8')
```

In [129]:

```
val_utf8
```

Out[129]:

```
b'espa\xc3\xb1ol'
```

In [130]:

```
type(val_utf8)
```

Out[130]:

```
bytes
```

Decode yöntemini kullanarak geri dönebilirsiniz.

In [131]:

```
val_utf8.decode('utf-8')
```

Out[131]:

```
'español'
```

Herhangi bir kodlama için UTF-8 kullanılması tercih edilse de, herhangi bir sayıda farklı kodlamada verilerle karşılaşabilirsiniz.

In [132]:

```
val.encode('latin1')
```

Out[132]:

```
b'espa\xf1ol'
```

In [133]:

```
val.encode('utf-16')
```

Out[133]:

```
b'\xff\xfee\x00s\x00p\x00a\x00\xf1\x00o\x001\x00'
```

In [134]:

```
val.encode('utf-16le')
```

Out[134]:

```
b'e\x00s\x00p\x00a\x00\xf1\x00o\x001\x00'
```

Bir dizeyi b ile önceden sabitleyerek kendi bayt değişmezlerinizi tanımlayabilirsiniz.

In [135]:

```
bytes_val = b'this is bytes'
```

In [136]:

```
bytes_val
```

Out[136]:

```
b'this is bytes'
```

In [137]:

```
decoded = bytes_val.decode('utf8')
```

In [138]:

```
decoded # artık str tipinde (Unicode)
```

Out[138]:

```
'this is bytes'
```

17- Booleans

Python'daki iki boole değeri True ve False olarak yazılır. Karşılaştırma lar ve diğer koşullu ifadeler Doğru veya Yanlış olarak değerlendirilir. Boolean değerleri, ve ve veya anahtar sözcükleriyle birleştirilir.

In [139]:

```
True and True
```

Out[139]:

```
True
```

In [140]:

```
False or True
```

Out[140]:

```
True
```

18- Tip Dönüşümü

Str, bool, int ve float türleri de bu türlere değer atamak için kullanılabilen işlevlerdir.

In [141]:

```
s = '3.14159'
```

In [142]:

```
fval = float(s)
```

In [143]:

```
type(fval)
```

Out[143]:

```
float
```

In [144]:

```
int(fval)
```

Out[144]:

```
3
```

In [145]:

```
bool(fval)
```

Out[145]:

```
True
```

In [146]:

```
bool(0)
```

Out[146]:

```
False
```

19- None

None, Python boş değer türüdür. Bir fonksiyon açıkça bir değer döndürmezse, None döndürür.

In [147]:

```
a = None
```

In [148]:

```
a is None
```

Out[148]:

```
True
```

In [149]:

```
b=5
```

In [150]:

```
b is not None
```

Out[150]:

```
True
```

None ayrıca fonksiyon bağımsız değişkenleri için ortak bir varsayılan değerdir.

In [151]:

```
def add_and_maybe_multiply(a, b, c=None):
    result=a+b

    if c is not None:
        result = result * c

    return result
```

In [152]:

```
type(None)
```

Out[152]:

```
NoneType
```

20- Dates ve times

Python datetime modülü, ve en yaygın kullanılan modüldür. Tarih saat, tarih ve saat türleri sağlar. Tarih saat türü, tarih ve saatte depolanan bilgileri birleştirir.

In [153]:

```
from datetime import datetime, date, time
```

In [157]:

```
dt = datetime(2011, 10, 29, 20, 30, 21) # 29 Ekim 2011 saat 20.30:21
```

In [155]:

```
dt.day # günü verecektir.
```

Out[155]:

29

In [156]:

```
dt.minute # dakikayı verecektir.
```

Out[156]:

30

In [159]:

```
dt.date() #tam tarihi verecektir.
```

Out[159]:

datetime.date(2011, 10, 29)

In [161]:

```
dt.time() # tam saati verecektir.
```

Out[161]:

datetime.time(20, 30, 21)

Strftime yöntemi, bir tarih saatini bir dize(string) olarak biçimlendirir.

In [162]:

```
dt.strftime('%m/%d/%Y %H:%M')
```

Out[162]:

'10/29/2011 20:30'

Dizeler strftime fonksiyonu ile datetime nesnelerine dönüştürülebilir.

In [163]:

```
datetime.strptime('20091031', '%Y%m%d')
```

Out[163]:

```
datetime.datetime(2009, 10, 31, 0, 0)
```

Zaman serisi verilerini toplarken veya başka şekilde grupperlendirmek zaman zaman bir dizi veri zamanının zaman alanlarını değiştirmek gerekecektir.

In [164]:

```
dt.replace(minute=0, second=0)
```

Out[164]:

```
datetime.datetime(2011, 10, 29, 20, 0)
```

Datetime.datetime değişmez bir tür olduğundan, bunun gibi yöntemler her zaman yeni nesneler üretir.

İki datetime nesnesinin farkı bir datetime.timedelta türü oluşturur.

In [165]:

```
dt2 = datetime(2011, 11, 15, 22, 30)
```

In [166]:

```
delta = dt2 - dt
```

In [167]:

```
delta
```

Out[167]:

```
datetime.timedelta(days=17, seconds=7179)
```

In [168]:

```
type(delta)
```

Out[168]:

```
datetime.timedelta
```

Bir tarih saatine bir timedelta eklemek yeni bir ileri tarih saati üretir.

In [169]:

dt

Out[169]:

datetime.datetime(2011, 10, 29, 20, 30, 21)

In [170]:

dt + delta

Out[170]:

datetime.datetime(2011, 11, 15, 22, 30)

Tarih ve saat biçimini belirtimi

%Y = Dört basamaklı yıl

%y = İki basamaklı yıl

%m = İki basamaklı ay [01, 12]

%d = İki basamaklı gün [01, 31]

%H = Saat (24 saatlik zaman) [00, 23]

%I = Saat (12 saatlik zaman) [01, 12]

%M = İki basamaklı dakika [00, 59]

%S = Saniye [00, 61] (saniye 60, 61 artık saniyeler için hesap)

%w = Tamsayı olarak hafta içi gün [0 (Pazar), 6]

%U = Yılın hafta sayısı [00, 53]; Pazar, haftanın ilk günü olarak kabul edilir ve yılın ilk Pazar gününden önceki günler "0. hafta" olarak kabul edilir

%W = Yılın hafta sayısı [00, 53] Pazartesi, haftanın ilk günü olarak kabul edilir ve yılın ilk Pazartesi gününden önceki günler "0. hafta" olarak kabul edilir

%z = UTC saat dilimi farkı + HHMMor-HHMM

%F = % Y-% m-% d için kısayol (ör. 2012-4-18)

%D = % M /% d /% y için kısayol (ör. 04/18/12)

Kontrol Akışı (Control Flow)

Python, koşullu mantık, döngüler ve diğer programlama dillerinde bulunan diğer standart kontrol akışı kavramları için birkaç yerleşik anahtar kelimeye sahiptir.

1- if, elif, and else

İf ifadesi, en iyi bilinen kontrol akışı ifadesi türlerinden biridir. Doğru ise, altındaki bloktaki kodu değerlendiren bir koşulu kontrol eder.

Bir if ifadesinin ardından istege bağlı olarak bir veya daha fazla elif bloğu gelebilir:

```
if x < 0:  
    print('Its negative')
```

```

        elif x == 0:
            print('Equal to zero')
        elif 0 < x < 5:
            print('Positive but smaller than 5')
        else:
            print('Positive and larger than or equal to 5')

```

Koşullardan herhangi biri Doğru ise, başka bir elif veya başka bloğa ulaşım ayacaktır. Ve veya veya kullanılarak bileşik bir koşulla, koşullar soldan sağa değerlendirilir.

In [177]:

```
a=5;b=7
```

In [178]:

```
c=8;d=4
```

In [181]:

```

if a < b or c > d:
    print('Made it')

```

Made it

Bu örnekte, $c > d$ karşılaştırması hiçbir zaman değerlendirilmez çünkü ilk karşılaştırma doğru.

Karşılaştırmaları zincirlemek de mümkündür.

In [182]:

```
4>3>2>1
```

Out[182]:

True

2- for loops

for döngüler, bir koleksiyon üzerinde veya bir yineleyici üzerinde yinelenme yapmak içindir. Bir for döngüsü için standart sözdizimi şöyledir:

```

for value in collection:
    # çalıştırılacak kod bloğu

```

Continue anahtar sözcüğünü kullanarak bloğun geri kalanını atlayarak bir sonraki yinelemeye bir for döngüsü ilerletebilirsiniz.

In [184]:

```
sequence = [1, 2, None, 4, None, 5]
```

In [185]:

```
total = 0
```

In [187]:

```
for value in sequence:
    if value is None:
        continue
    total += value # Bu kod, 5'e ulaşılana kadar listenin öğelerini toplar.
```

In [192]:

```
sequence=[1,2,0,4,6,5,2,1]
total_until_5 = 0
for value in sequence:
    if value == 5:
        break
    total_until_5 += value # Break anahtar sözcüğü yalnızca en içteki for döngüsünü
```

In [195]:

```
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)
```

3- while Döngüleri

While döngüsü, koşul False olarak değerlendirilene veya döngü açıkça break ile sonlandırılana kadar yürütülecek bir koşul ve kod bloğunu belirtir.

In [197]:

```
x=256
total = 0
while x > 0:
    if total > 500:
        break
    total += x
    x=x//2
```

4- pass

pass, Python'daki "işlemsiz" ifadesidir. Herhangi bir işlemin yapılmayacağı bloklarda kullanılabilir.

In [198]:

```
if x < 0:
    print('negative!')
elif x == 0:
    # herhangi bir şey
    pass
else:
    print('positive!')
```

positive!

5- range

Range işlevi, eşit aralıklı tamsayılar dizisi veren bir yineleyici döndürür.

In [199]:

range(10)

Out[199]:

range(0, 10)

In [200]:

list(range(10))

Out[200]:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [201]:

list(range(0, 20, 2))

Out[201]:

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

In [202]:

```
list(range(5, 0, -1))
```

Out[202]:

```
[5, 4, 3, 2, 1]
```

range, uç noktaya kadar olan ancak dahil olmayan tamsayılar üretir.

In [204]:

```
seq=[1,2,3,4]
for i in range(len(seq)):
    val = seq[i]
```

Range tarafından üretilen tüm tam sayıları başka bir veri yapısında saklamak için list gibi fonksiyonlar kullanılabilir.

In [207]:

```
sum = 0
for i in range(100000):
    # % is the modulo operator
    if i % 3 == 0 or i % 5 == 0:
        sum+=i # 0 ile 99.999 arasındaki 3 veya 5'in katları olan tüm sayıları topla
```

6- Üçlü ifadeler (Ternary expressions)

Üçlü ifade (Ternary expression), bir değer üreten bir if-else bloğunu tek bir satır veya ifadede birleştirmenize izin verir. Yazılışı şöyledir:

```
value = true-expr if condition else false-expr
```

```
Burada, true-expr ve false-expr herhangi bir Python ifadesi olabilir
if condition:
    value = true-expr
else:
    value = false-expr
```

In [211]:

```
x=5
```

In [212]:

```
'Non-negative' if x >= 0 else 'Negative' # If-else bloklarında olduğu gibi, ifadeler
```

Out[212]:

```
'Non-negative'
```

!! üçlü ifadeler (Ternary expressions) kullanmak kolay olsa da, koşulun yanı sıra doğru ve yanlış ifadeler çok karmaşıksa okunabilirliği azaltacaktır.

C) Veri Yapıları, Fonksiyonları ve Dosyaları

1- Veri Yapıları ve Dizileri

• Tuple

Tuple, sabit uzunlukta, değişmez bir Python nesneleri dizisidir. Bir tane oluşturmanın en kolay yolu, virgülle ayrılmış değerler dizisi kullanmaktır.

In [27]:

```
tup = 4, 5, 6
```

In [28]:

```
tup
```

Out[28]:

```
(4, 5, 6)
```

Daha karmaşık ifadeleri Tuple ile tanımlarken parantez içine almak gereklidir.

In [29]:

```
nested_tup = (4, 5, 6), (7, 8)
```

In [30]:

```
nested_tup
```

Out[30]:

```
((4, 5, 6), (7, 8))
```

Tuple'ı çağırarak herhangi bir diziyi(sequence) veya yineleyiciyi(iterator) bir Tuple'a dönüştürebilirsiniz.

In [31]:

```
tuple([4, 0, 2])
```

Out[31]:

```
(4, 0, 2)
```

In [32]:

```
tup = tuple('string')
```

In [33]:

```
tup
```

Out[33]:

```
('s', 't', 'r', 'i', 'n', 'g')
```

İndislere, köşeli parantez [] ile erişilebilir.

In [34]:

```
tup[0]
```

Out[34]:

```
's'
```

Tuple nesneleri kendi başlarına değiştirilebilir olsa da, Tuple oluşturulduktan sonra hangi nesnenin saklanacağını değiştirmek mümkün değildir.

In [35]:

```
tup = tuple(['foo', [1, 2], True])
```

In [22]:

```
from IPython.display import Image
Image(filename='img/Picture16.png')
```

Out[22]:

```
In [36]: tup[2] = False
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-36-b89d0c4ae599> in <module>
      1 tup[2] = False
----> 1 TypeError: 'tuple' object does not support item assignment
```

Bir Tuple içindeki bir nesne, liste gibi değiştirilebilirse, onu değiştirebilirsiniz

In [40]:

```
tup[1].append(3)
```

In [41]:

```
tup
```

Out[41]:

```
('foo', [1, 2, 3], True)
```

Daha uzun Tuple oluşturmak için + operatörünü kullanarak Tuple'ları birleştirebilirsiniz.

In [42]:

```
(4, None, 'foo') + (6, 0) + ('bar',)
```

Out[42]:

```
(4, None, 'foo', 6, 0, 'bar')
```

Tuple'ları tam sayılar ile çarpabilirsiniz. Tuple kendini çarpım sayısı kadar tekrar edecektir.

!! Unutmayın nesneler kendini kopyalamayacaktır. Sadece onlara yapılan referans kopyalanacaktır.

In [43]:

```
('foo', 'bar') * 4
```

Out[43]:

```
('foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'bar')
```

1- Unpacking Tuples

Değişkenlerin Tuple benzeri bir ifadeye atamaya çalışırsanız, eşittir işaretiinin sağ tarafındaki değer unpack edilir.

In [44]:

```
tup=(4,5,6)
```

In [45]:

```
a,b,c=tup
```

In [50]:

```
a
```

Out[50]:

```
4
```

In [46]:

```
b
```

Out[46]:

5

In [51]:

```
c
```

Out[51]:

6

İç içe geçmiş Tuple dizilerde de bunu uygulamak mümkündür.

In [47]:

```
tup=4,5,(6,7)
```

In [48]:

```
a,b,(c,d)=tup
```

In [52]:

```
a
```

Out[52]:

4

In [53]:

```
b
```

Out[53]:

5

In [54]:

```
c
```

Out[54]:

6

In [55]:

```
d
```

Out[55]:

7

yeni bir değişkene ihtiyaç olmadan kolayca değiştirebilirsiniz.

In [57]:

```
tmp=a  
a=b  
b=tmp
```

Tuple ile değiştirme şu şekilde yapılabilir.

In [58]:

```
a,b=1,2
```

In [59]:

```
a
```

Out[59]:

```
1
```

In [60]:

```
b
```

Out[60]:

```
2
```

In [64]:

```
b,a=a,b # a değişkeni b değerini, b değişkeni a değerini alacaktır.
```

In [65]:

```
a
```

Out[65]:

```
1
```

In [66]:

```
b
```

Out[66]:

```
2
```

Tuple veya liste dizileri üzerinde yineleme yapabilirsiniz.

In [67]:

```
seq=[(1,2,3),(4,5,6),(7,8,9)]
```

In [68]:

```
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
```

Rest ile bir Tuple'ı başından birkaç öğeyi kesebilir, kesilen değerleri başk a bir Tuple a atayabilirsiniz.

In [69]:

```
values=1,2,3,4,5
```

In [70]:

```
a, b, *rest = values
```

In [71]:

```
a, b
```

Out[71]:

```
(1, 2)
```

In [72]:

```
rest
```

Out[72]:

```
[3, 4, 5]
```

rest yerine *_ kullanabilirsiniz.

In [73]:

```
a, b, *_ = values
```

2- Tuple Metodları
count ile dizide, elemanın sayısını görebilirsiniz.

In [74]:

```
a=(1,2,2,2,3,4,2)
```

In [75]:

```
a.count(2)
```

Out[75]:

```
4
```

•List

Tuple'ların aksine, listeler değişken uzunluktadır ve nesnelerin yerleri değiştirilebilir. Bunları köşeli parantez [] veya liste türü işleviyle tanımlayabilirsiniz.

In [76]:

```
a_list = [2, 3, 7, None]
```

In [77]:

```
tup = ('foo', 'bar', 'baz')
```

In [78]:

```
b_list = list(tup)
```

In [79]:

```
b_list
```

Out[79]:

```
['foo', 'bar', 'baz']
```

In [80]:

```
b_list[1] = 'peekaboo'
```

In [81]:

```
b_list
```

Out[81]:

```
['foo', 'peekaboo', 'baz']
```

Listeler ve Tuple'lar anlamsal olarak benzerdir (tuplelar değiştirilemez) ve birçok işlevde birbirinin yerine kullanılabilir.

List fonksiyonu, bir değer aralığı alarak veri üretmek için de kullanılır.

In [82]:

```
gen = range(10)
```

In [83]:

```
gen
```

Out[83]:

```
range(0, 10)
```

In [84]:

```
list(gen)
```

Out[84]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

1- Eleman ekleme ve çıkarma

Ekleme yöntemi ile elemanlar listenin sonuna eklenebilir.

In [85]:

```
b_list.append('dwarf')
```

In [86]:

```
b_list
```

Out[86]:

```
['foo', 'peekaboo', 'baz', 'dwarf']
```

insert metodu kullanarak listedeki belirli bir konuma bir eleman ekleyebilirsiniz.

In [89]:

```
b_list.insert(1, 'red') # Ekleme dizini 0 ile liste uzunluğu arasında olmalıdır.
```

In [90]:

```
b_list
```

Out[90]:

```
['foo', 'red', 'red', 'peekaboo', 'baz', 'dwarf']
```

!! insert, append'a kıyasla hesaplama açısından zordur, çünkü yeni öğeye yer açmak için sonraki öğelere yapılan referanslar kaydırılmalıdır. Bir dizinin hem başına hem de sonuna eleman eklemeniz gerekiyorsa, collections.deque'yi kullanabilirsiniz.

pop ile dizideki herhangi bir elemanı kaldırabilirsiniz.

In [93]:

```
b_list.pop(2)
```

Out[93]:

```
'peekaboo'
```

In [94]:

```
b_list
```

Out[94]:

```
['foo', 'red', 'baz', 'dwarf']
```

Dizinizde 2 veya daha fazla aynı ada sahip eleman varsa remove ile öncelik olarak önde olan indis kaldırılacaktır.

In [95]:

```
b_list.append('foo')
```

In [96]:

```
b_list
```

Out[96]:

```
['foo', 'red', 'baz', 'dwarf', 'foo']
```

In [98]:

```
b_list.remove('foo')
```

In [99]:

```
b_list
```

Out[99]:

```
['red', 'baz', 'dwarf', 'foo']
```

in anahtar sözcüğünü kullanarak bir listenin bir değer içerip içermediğini kontrol edebilirsiniz.

not in için de geçerlidir.

In [100]:

'dwarf' in b_list

Out[100]:

True

In [101]:

'dwarf' not in b_list

Out[101]:

False

2- Listeleri Birleştirme(Concatenating and combining lists)

Tuple'lara benzer şekilde, + ile birlikte onları birleştirebilirsiniz.

In [102]:

[4, None, 'foo'] + [7, 8, (2, 3)]

Out[102]:

[4, None, 'foo', 7, 8, (2, 3)]

Önceden tanımlanmış bir listeniz varsa, extend method kullanarak birden çok öğe ekleyebilirsiniz.

In [103]:

x = [4, None, 'foo']

In [104]:

x.extend([7, 8, (2, 3)])

In [105]:

x

Out[105]:

[4, None, 'foo', 7, 8, (2, 3)]

Elemanları mevcut bir listeye eklemek için, özellikle de büyük bir liste oluşturuyorsanız extend kullanabilirsiniz.

```
everything = []
for chunk in list_of_lists:
    everything.extend(chunk)
```

! bitiştirme alternatifinden daha hızlıdır. !

```
everything = []
for chunk in list_of_lists:
    everything = everything + chunk
```

3- Sıralama (Sorting)

Sort ile dizideki elemanları sıralayabilirsiniz.

In [107]:

```
a=[7,2,5,1,3]
```

In [109]:

```
a.sort()
```

In [110]:

```
a
```

Out[110]:

```
[1, 2, 3, 5, 7]
```

Sort ile string ifadeleri kelime uzunluklarına göre sıralayabilirsiniz.

In [112]:

```
b = ['saw', 'small', 'He', 'foxes', 'six']
```

In [113]:

```
b.sort(key=len)
```

In [114]:

```
b
```

Out[114]:

```
['He', 'saw', 'six', 'small', 'foxes']
```

4- İkili arama ve sıralı bir liste tutma

bisect.bisect, bir öğenin sıralanması için yerleştirilmesi gereken yeri bulur, bisect.insort ise öğeyi o konuma ekler.

In [115]:

```
import bisect
```

In [116]:

```
c=[1,2,2,2,3,4,7]
```

In [117]:

```
bisect.bisect(c, 2)
```

Out[117]:

4

In [118]:

```
bisect.bisect(c, 5)
```

Out[118]:

6

In [119]:

```
bisect.insort(c, 6)
```

In [120]:

```
c
```

Out[120]:

[1, 2, 2, 2, 3, 4, 6, 7]

!! bisect ile bunları sıralanmamış bir liste ile kullanmak hatasız bir şekilde sonuçlanır olur ancak yanlış sonuçlara yol açabilir.

5- Slicing(Dilimleme)

Dizide herhangi bir aralık alabilirsiniz.

In [121]:

```
seq=[7,2,3,7,5,6,0,1]
```

In [122]:

```
seq[1:5]
```

Out[122]:

[2, 3, 7, 5]

İndis belirterek sırayla o indise eleman atayabilirsiniz.

In [123]:

```
seq[3:4] = [6, 3]
```

In [124]:

```
seq
```

Out[124]:

```
[7, 2, 3, 6, 3, 5, 6, 0, 1]
```

seq[:değer] kullanarak baştan o değer indisine kadar yazdırmayı sağlayabilirsiniz.

seq[değer:] kullanarak değer indisinden başlayarak sona kadar olan elemanları yazdırabilirsiniz.

In [125]:

```
seq[:5]
```

Out[125]:

```
[7, 2, 3, 6, 3]
```

In [126]:

```
seq[3:]
```

Out[126]:

```
[6, 3, 5, 6, 0, 1]
```

Negatif indeksler, diziyi sona göre böler.

In [127]:

```
seq[-4:]
```

Out[127]:

```
[5, 6, 0, 1]
```

In [128]:

```
seq[-6:-2]
```

Out[128]:

```
[6, 3, 5, 6]
```

Dizide aralıklı alarak eleman almak mümkündür.

In [129]:

seq[::-2]

Out[129]:

[7, 3, 3, 6, 1]

Diziyi ters çevirmek aşağıdaki gibi mümkündür.

In [130]:

seq[::-1]

Out[130]:

[1, 0, 6, 5, 3, 6, 3, 2, 7]

•Yerleşik Sıra Fonksiyonları(Built-in Sequence Functions)

1- enumerate

Bir dizi üzerinde yineleme yaparken, mevcut ögenin dizinini takip etmek istenir.

```
i=0
for value in collection:
    # herhangi bir değer
    i+=1

for i, value in enumerate(collection):
    # do something with value
```

Diziyi listelerken her eleman benzersiz bir değer alır.

In [132]:

some_list = ['foo', 'bar', 'baz']

In [133]:

mapping = {}

In [134]:

```
for i, v in enumerate(some_list):
    mapping[v] = i
```

In [135]:

```
mapping
```

Out[135]:

```
{'foo': 0, 'bar': 1, 'baz': 2}
```

2- sorted

Herhangi bir dizinin öğelerinden yeni bir sıralı liste döndürür.

In [136]:

```
sorted([7, 1, 2, 6, 0, 3, 2])
```

Out[136]:

```
[0, 1, 2, 2, 3, 6, 7]
```

In [137]:

```
sorted('horse race')
```

Out[137]:

```
[' ', 'a', 'c', 'e', 'e', 'h', 'o', 'r', 'r', 's']
```

3- zip

tuple'in veya diğer dizilerin öğelerini eşler.

In [138]:

```
seq1 = ['foo', 'bar', 'baz']
```

In [139]:

```
seq2 = ['one', 'two', 'three']
```

In [140]:

```
zipped = zip(seq1, seq2)
```

In [141]:

```
list(zipped)
```

Out[141]:

```
[('foo', 'one'), ('bar', 'two'), ('baz', 'three')]
```

zip, rastgele sayıda sıra alabilir ve ürettiği öğelerin sayısı en kısa sıraya göre belirlenir.

In [142]:

```
seq3 = [False, True]
```

In [143]:

```
list(zip(seq1, seq2, seq3))
```

Out[143]:

```
[('foo', 'one', False), ('bar', 'two', True)]
```

Çok yaygın bir zip kullanımı, aynı anda birden fazla sekans üzerinde yinelemeye yapmaktadır. enumerate ile birleşebilir.

In [144]:

```
for i, (a, b) in enumerate(zip(seq1, seq2)):
    print('{0}: {1}, {2}'.format(i, a, b))
```

```
0: foo, one
1: bar, two
2: baz, three
```

Ziplenmiş bir sıra verildiğinde, diziyi unziplemek için zip akıllıca uygulanabilir.

In [145]:

```
pitchers = [('Nolan', 'Ryan'), ('Roger', 'Clemens'), ('Schilling', 'Curt')]
```

In [146]:

```
first_names, last_names = zip(*pitchers)
```

In [147]:

```
first_names
```

Out[147]:

```
('Nolan', 'Roger', 'Schilling')
```

In [148]:

```
last_names
```

Out[148]:

```
('Ryan', 'Clemens', 'Curt')
```

4- reversed

reversed sıralı bir dizinin öğelerini tersine yineler.

In [149]:

```
list(reversed(range(10)))
```

Out[149]:

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

•dict

Python nesneleri esnek boyutlu bir anahtar-değer çiftleri koleksiyonudur. Ol usturmanın yolu, anahtarları ve değerleri ayırmak için küme parantezi {} ve iki nokta üst üste kullanmaktadır.

In [150]:

```
empty_dict = {}
```

In [151]:

```
d1 = {'a' : 'some value', 'b' : [1, 2, 3, 4]}
```

In [152]:

```
d1
```

Out[152]:

```
{'a': 'some value', 'b': [1, 2, 3, 4]}
```

Bir liste veya başlığın öğelerine erişimle aynı sözdizimini kullanarak öğe re erişebilir, ekleyebilir veya ayarlayabilirsiniz.

In [153]:

```
d1[7] = 'an integer'
```

In [154]:

```
d1
```

Out[154]:

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

In [155]:

```
d1['b']
```

Out[155]:

```
[1, 2, 3, 4]
```

Bir listenin bir değer içerip içermediğini, bir anahtar içerip içermediğini

kontrol edebilirsiniz.

In [156]:

```
'b' in d1
```

Out[156]:

```
True
```

Değerleri del sözcüğünü ya da pop yöntemini kullanarak silebilirsiniz.

In [157]:

```
d1[5] = 'some value'
```

In [158]:

```
d1
```

Out[158]:

```
{'a': 'some value', 'b': [1, 2, 3, 4], 7: 'an integer', 5: 'some value'}
```

In [159]:

```
d1['dummy'] = 'another value'
```

In [160]:

```
d1
```

Out[160]:

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

In [161]:

```
del d1[5]
```

In [162]:

```
d1
```

Out[162]:

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

In [163]:

```
ret = d1.pop('dummy')
```

In [164]:

```
ret
```

Out[164]:

```
'another value'
```

In [165]:

```
d1
```

Out[165]:

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

Anahtarlar ve değerler size sırasıyla anahtarlarının ve değerlerinin yineleyicilerini verir.

In [166]:

```
list(d1.keys())
```

Out[166]:

```
['a', 'b', 7]
```

In [167]:

```
list(d1.values())
```

Out[167]:

```
['some value', [1, 2, 3, 4], 'an integer']
```

Update yöntemini kullanarak bir dikteyi diğeriyile birleştirebilirsiniz.

In [168]:

```
d1.update({'b': 'foo', 'c': 12})
```

In [169]:

```
d1
```

Out[169]:

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

!! Update yöntemi güncellenen verilerdeki mevcut anahtarların eski değerleri atılır.

1- Dizilerden dikteler oluşturma

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

In [171]:

```
mapping = dict(zip(range(5), reversed(range(5))))
```

In [172]:

```
mapping
```

Out[172]:

```
{0: 4, 1: 3, 2: 2, 3: 1, 4: 0}
```

2- Varsayılan değerler

```
if key in some_dict:
    value = some_dict[key]
else:
    value = default_value
```

Böylece, dict yöntemleri get ve pop döndürülecek varsayılan bir değer alabilir, böylece yukarıdaki if-else bloğu basitçe şu şekilde yazılabilir:

```
value = some_dict.get(key, default_value)
```

varsayılan olarak get, anahtar yoksa None döndürür, pop ise bir istisna oluşturur

In [173]:

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

In [174]:

```
by_letter = {}
```

In [175]:

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

In [176]:

by_letter

Out[176]:

```
{'a': ['apple', 'atom'], 'b': ['bat', 'bar', 'book']}
```

Setdefault dict yöntemi tam da bu amaç içindir. Önceki for döngüsü şu şekilde yeniden yazılabilir:

In [177]:

```
for word in words:
    letter = word[0]
    by_letter.setdefault(letter, []).append(word)
```

built-in collections, bunu daha da kolaylaştırın kullanışlı bir sınıfı, defaultdict'e sahiptir.

In [178]:

```
from collections import defaultdict
by_letter = defaultdict(list)
for word in words:
    by_letter[word[0]].append(word)
```

3- Geçerli dikte anahtar türleri

Bir diktenin değerleri herhangi bir nesne olabilirken, anahtarlar genellikle skaler türler (int, float, string) veya tuples (demetteki tüm nesnelerin de değişmez olması gereklidir) gibi değişmez nesneler olmalıdır. Buradaki teknik terim, hashability'dir. Hash fonksiyonuyla bir nesnenin hashable olmadığını kontrol edebilirsiniz.

In [179]:

hash('string')

Out[179]:

```
-24849086933898132
```

In [180]:

hash((1, 2, (2, 3)))

Out[180]:

```
-9209053662355515447
```

In [23]:

Image(filename='img/Picture17.png')

Out[23]:

In [182]: hash([1, 2, [2, 3])) # fails because lists are mutable

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-182-473c35a62c0b> in <module>  
----> 1 hash([1, 2, [2, 3])) # fails because lists are mutable  
  
TypeError: unhashable type: 'list'
```

Bir listeyi anahtar olarak kullanmak için, listeyi bir demete dönüştürmek gereklidir.

In [184]:

d={}

In [185]:

d[tuple([1, 2, 3])] = 5

In [186]:

d

Out[186]:

{(1, 2, 3): 5}

•set

Küme, benzersiz öğelerden oluşan sırasız bir koleksiyondur. Anahtarlar, değerleri yoktur. Bir küme iki şekilde oluşturulabilir: küme fonksiyon aracılığıyla veya küme parantezleri aracılığıyla.

In [187]:

set([2, 2, 2, 1, 3, 3])

Out[187]:

{1, 2, 3}

In [189]:

```
{2, 2, 2, 1, 3, 3}
```

Out[189]:

```
{1, 2, 3}
```

Kümeler, birleşim, kesişim, fark ve simetrik fark gibi matematiksel küme işlevlerini destekler.

In [190]:

```
a={1, 2, 3, 4, 5}
```

In [192]:

```
b={3, 4, 5, 6, 7, 8}
```

Bu iki kümenin birleşimi, her iki kümede de meydana gelen farklı öğeler kümesidir. Bu, birleşim yöntemi or ifadesini tam olarak açıklar.

In [193]:

```
a.union(b)
```

Out[193]:

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [198]:

```
a | b
```

Out[198]:

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

Kesişim, her iki sette de meydana gelen öğeleri içerir.

In [195]:

```
a.intersection(b)
```

Out[195]:

```
{3, 4, 5}
```

In [197]:

a & b

Out[197]:

{3, 4, 5}

In [24]:

Image(filename='img/Picture18.png')

Out[24]:

Table 3-1. Python set operations

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 both 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 not both
a.symmetric_difference_update(b)	a ^= b	Set a to contain the elements in either a or b but not both
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

Mantıksal küme işlemlerinin tümü, işlemin sol tarafındaki kümenin içeriğini sonuçla değiştirmenize olanak tanır.

In [200]:

c = a.copy()

In [203]:

c |= b

In [204]:

c

Out[204]:

{1, 2, 3, 4, 5, 6, 7, 8}

In [205]:

```
d = a.copy()
```

In [207]:

```
d &= b
```

In [208]:

```
d
```

Out[208]:

```
{3, 4, 5}
```

Dikteler gibi, set öğeleri de genellikle değişmez olmalıdır. Liste benzeri ögelere sahip olmak için, onu bir Tuple'a dönüştürmelisiniz.

In [209]:

```
my_data = [1, 2, 3, 4]
```

In [210]:

```
my_set = {tuple(my_data)}
```

In [211]:

```
my_set
```

Out[211]:

```
{(1, 2, 3, 4)}
```

Ayrıca bir kümenin başka bir kümenin bir alt kümesi mi (içinde bulunduğu) ve ya bir üst kümesi mi (tüm öğelerini içeren) olup olmadığını da kontrol edebilirsiniz.

In [212]:

```
a_set = {1, 2, 3, 4, 5}
```

In [213]:

```
{1, 2, 3}.issubset(a_set)
```

Out[213]:

```
True
```

In [214]:

```
a_set.issuperset({1, 2, 3})
```

Out[214]:

True

Kümeler ancak ve ancak içerikleri eşitse eşittir.

In [216]:

```
{1,2,3} == {3,2,1}
```

Out[216]:

True

•List, Set, ve Dict Comprehensions

Bir koleksiyonun öğelerini filtreleyerek, filtreyi geçen öğeleri kısa ve öz bir ifadede dönüştürerek yeni bir liste oluşturmanıza olanak tanır.

```
[expr for val in collection if condition]
```

Bu, aşağıdaki for döngüsü ile aynıdır

```
result = []
for val in collection:
    if condition:
        result.append(expr)
```

Dizelerin bir listesi verildiğinde, uzunluğu 2 veya daha az olan dizeleri filtreleyebilir ve bunları aşağıdaki gibi büyük harfe dönüştürebiliriz.

In [217]:

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

In [218]:

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

Out[218]:

```
['BAT', 'CAR', 'DOVE', 'PYTHON']
```

Bir dikt comprehension şuna benzer

```
dict_comp = {key-expr : value-expr for value in collection
            if condition}
```

Küme comprehension, köşeli parantezler yerine küme parantezleri dışında eşdeğer liste anlayışına benzer.

```
set_comp = {expr for value in collection if condition}
```

Liste comprehensions çoğunlukla kolaylık sağlar, kodu hem yazmayı hem de okumayı kolaylaştırır.

In [219]:

```
unique_lengths = {len(x) for x in strings}
```

In [220]:

```
unique_lengths
```

Out[220]:

```
{1, 2, 3, 4, 6}
```

map fonksiyonunu kullanarak da ifade edebiliriz.

In [221]:

```
set(map(len, strings))
```

Out[221]:

```
{1, 2, 3, 4, 6}
```

Dizelerin listedeki konumlarına ilişkin bir arama haritası oluşturabiliriz.

In [222]:

```
loc_mapping = {val : index for index, val in enumerate(strings)}
```

In [223]:

```
loc_mapping
```

Out[223]:

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

1- Nested list comprehensions

Bazı İngilizce ve İspanyolca isimleri içeren bir liste olduğunu varsayıalım.

In [225]:

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

İçinde iki veya daha fazla e harfi bulunan tüm isimleri içeren tek bir liste elde etmeye çalışalım.

In [226]:

```
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)
```

Yukarıdaki işlemleri daha kısa olarak aşağıdaki şekilde yazabilirsiniz.

In [227]:

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

In [228]:

```
result
```

Out[228]:

```
['Steven']
```

Başka bir örnek

In [232]:

```
some_tuples = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]
```

In [233]:

```
flattened = [x for tup in some_tuples for x in tup]
```

In [234]:

```
flattened
```

Out[234]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Bir list comprehension yerine iç içe geçmiş bir for döngüsü yazarsanız, for ifadelerinin sırasının aynı olacağını unutmayın.

```
flattened = []
```

```
for tup in some_tuples: for x in tup: flattened.append(x)
```

Ya da

In [236]:

```
[[x for x in tup] for tup in some_tuples] # Bu, tüm iç öğelerin düzleştirilmiş bir liste oluşturur.
```

Out[236]:

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

2- Fonksiyonlar

Fonksiyonlar, kod düzenleme ve yeniden kullanım için kullanılır. Aynı veya çok benzer kodu birden çok kez tekrarlıyorsanız kullanılabilir.

Fonksiyonlar def anahtar sözcüğü ile bildirilir ve return anahtar sözcüğü ile döndürülür.

In [29]:

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

Önceki fonksiyonda, x ve y konumsal argümanlar iken z bir anahtar kelime argumentür. Bu, işlevin aşağıdaki yollardan herhangi biriyle çağrılabileceği anlamına gelir.

In [31]:

```
my_function(5, 6, z=0.7)
my_function(3.14, 7, 3.5)
my_function(10, 20)
```

Out[31]:

45.0

Konumsal argümanları iletmek için anahtar kelimeler kullanmak da mümkündür.

Önceki örnekte şunu da yazabilirdik:

Daha okunabilir olacaktır.

In [33]:

```
my_function(x=5, y=6, z=7)
my_function(y=6, x=5, z=7)
```

Out[33]:

77

Namespaces, Scope, ve Local Fonksiyonlar

Fonksiyonlar değişkenlere iki farklı kapsamda erişebilir: genel ve yerel. Bir fonksiyon içinde varsayılan olarak atanan tüm değişkenler yerel ad alanına atanır. Yerel isim alanı, fonksiyon çağrılığında ve fonksiyonun argümanları tarafından hemen doldurulduğunda oluşturulur. İşlev tamamlandıktan sonra, yerel ad alanı yok edilir.

In [34]:

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

Yukarıdaki örnekte Func () çağrılığında, boş bir a listesi oluşturulur, beş öğe eklenir ve ardından fonksiyondan çıkışlığında a listesi yok edilir.

In [35]:

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

Değişkenleri fonksiyonun kapsamı dışında atamak mümkün değildir, ancak bu değişkenler global olarak tanımlanmalıdır.

In [36]:

```
a = None
```

In [38]:

```
def bind_a_variable():
    global a
    a=[]
bind_a_variable()
```

In [39]:

```
print(a)
```

None

Birden Çok Değer Döndürme

Bir fonksiyondan birden çok değer döndürebilirsiniz.

In [40]:

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

In [41]:

```
a,b,c=f()
```

In [42]:

```
return_value = f()
```

In [43]:

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

Fonksiyonlar Nesnelerdir

Fonksiyonlar nesneler olduğundan, diğer dillerde yapılması zor olan birçok y^{apı} kolaylıkla ifade edilebilir.

In [44]:

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

Bu dizeler listesini analize hazır hale getirmek için boşlukları kaldırma^k, noktalama işaretlerini kaldırma^k ve uygun büyük harf kullanımının standartla^şştırmak. Bunu yapmanın bir yolu, normal ifadeler için yeniden standart kütü^phane modülüyle birlikte built-in string methods kullanmaktır

In [45]:

```
import re
```

In [46]:

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

Sonuç şuna dönüşecektir:

In [47]:

```
clean_strings(states)
```

Out[47]:

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

In [48]:

```
def remove_punctuation(value):
    return re.sub('[!#?]', '', value)
```

In [49]:

```
clean_ops = [str.strip, remove_punctuation, str.title]
```

In [53]:

```
def clean_strings(strings, ops):
    result = []
    for value in strings:
        for function in ops:
            value = function(value)
        result.append(value)
    return result
```

Sonuç şuna dönüşecektir:

In [54]:

```
clean_strings(states, clean_ops)
```

Out[54]:

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

Clean_strings fonksiyonu de artık daha okunabilir ve geneldir.

Bir fonksiyonu bir tür diziye uygulayan yerleşik map fonksiyonu gibi diğer fonksiyonlar için bağımsız değişkenler olarak kullanabilirsiniz:

In [55]:

```
for x in map(remove_punctuation, states):
    print(x)
```

```
Alabama
Georgia
Georgia
georgia
FLORIDA
south carolina
West virginia
```

Anonim (Lambda) Fonksiyonları

Tek bir ifadeden oluşan fonksiyonları yazmanın bir yolu olan anonim veya lambda fonksiyonları sonucu dönüş değerini olur. Bunlar lambda anahtar sözcüğü ile tanımlanırlar.

In [58]:

```
def short_function(x):
    return x * 2
```

In [59]:

```
equiv_anon = lambda x: x * 2
```

Veri analizinde kullanışlıdırılar çünkü, veri dönüştürme fonksiyonlarının bağımsız değişken olarak alacağı birçok durum vardır.

In [60]:

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

In [61]:

```
ints=[4,0,1,5,6]
```

In [63]:

```
apply_to_list(ints, lambda x: x * 2)
```

Out[63]:

```
[8, 0, 2, 10, 12]
```

`[x * 2 for x in ints]` yazabilirdiniz, ancak burada kısa ve öz bir özel fonksiyon olan `apply_to_list` e geçtik.

Başka bir örnek olarak, bir dizeler listesini her dizedeki farklı harflerin sayısına göre sıralamak istediğiniz varsayıyalım.

In [64]:

```
strings = ['foo', 'card', 'bar', 'aaaa', 'abab']
```

Burada, listenin sıralama yöntemine bir lambda fonksiyonu yazabilirsiniz.

In [65]:

```
strings.sort(key=lambda x: len(set(list(x))))
```

In [66]:

```
strings
```

Out[66]:

```
['aaaa', 'foo', 'abab', 'bar', 'card']
```

Currying: Kısmi Argüman Uygulaması

Currying, kısmi argüman uygulamasıyla mevcut fonksiyonlardan yeni fonksiyonlar türetmek anlamına gelen bilgisayar bilimleri jargonudur.

In [67]:

```
def add_numbers(x, y):
    return x + y
```

Bu fonksiyonu kullanarak, argümanına 5 ekleyen, add_five adlı tek değişkenli yeni bir fonksiyon türetebilirsiniz.

In [69]:

```
add_five = lambda y: add_numbers(5, y)
```

Mevcut bir fonksiyonu çağırın yeni bir fonksiyon tanımlandı. functools modülünü kullanarak bu işlemi basitleştirebilirsiniz.

In [70]:

```
from functools import partial
```

In [71]:

```
add_five = partial(add_numbers, 5)
```

Generators

Bir listedeki nesneler veya bir dosyadaki satırlar gibi dizileri yinelemek için bir özelliktir. Bu, nesneleri yinelenebilir hale getirmek iterator aracılarıyla sağlanır.

In [72]:

```
some_dict = {'a': 1, 'b': 2, 'c': 3}
```

In [73]:

```
for key in some_dict:  
    print(key)
```

a
b
c

Some_dict ilk olarak some_dict'ten bir iterator oluşturmaya çalışır.

In [74]:

```
dict_iterator = iter(some_dict)
```

In [75]:

dict_iterator

Out[75]:

<dict_keyiterator at 0x7fe6bc58a770>

Liste veya liste benzeri bir nesne bekleyen çoğu yöntem, iterable nesnelerde kabul eder. Bu, min, max ve sum tuple gibi tür oluşturucuları içerir.

In [76]:

list(dict_iterator)

Out[76]:

['a', 'b', 'c']

Generator, yeni bir yinelenebilir nesne oluşturmanın özlü bir yoludur. Normal işlevler bir seferde tek bir sonucu çalıştırıp geri döndürürken, bu birden çok sonuç dizisi döndürür, her birinden sonra bir sonraki talep edilene kadar duraklar. Generator oluşturmak için fonksiyonda return yerine yield anahtar sözcüğünü kullanın.

In [79]:

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

Generator çağırığınız zaman, hiçbir kod hemen çalıştırılmaz

In [80]:

gen = squares()

In [81]:

gen

Out[81]:

<generator object squares at 0x7fe6bc635820>

Öge talep etmeniz gereklidir.

In [82]:

```
for x in gen:
    print(x, end=' ')
```

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

1- Generator ifadeleri

In [83]:

```
gen = (x ** 2 for x in range(100))
```

In [84]:

```
gen
```

Out[84]:

```
<generator object <genexpr> at 0x7fe6bbb582e0>
```

Yukarıdaki ifade, aşağıdaki daha ayrıntılı ifadeye tamamen eşdeğerdır.

In [85]:

```
def _make_gen():
    for x in range(100):
        yield x ** 2
gen = _make_gen()
```

çoğu durumda fonksiyon argumentleri olarak list comprehensions yerine kullanılabılır.

In [86]:

```
sum(x ** 2 for x in range(100))
```

Out[86]:

```
328350
```

In [87]:

```
dict((i, i ** 2) for i in range(5))
```

Out[87]:

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

2- Itertools modülü

`itertools` modülü, birçok ortak veri algoritması için bir generator koleksiyonudur.

iyonuna sahiptir. Örneğin, groupby herhangi bir sırayı ve bir işlevi alır ve sıradaki ardışık öğeleri işlevin dönüş değerine göre grupperdir.

In [88]:

```
import itertools
```

In [89]:

```
first_letter = lambda x: x[0]
```

In [90]:

```
names = ['Alan', 'Adam', 'Wes', 'Will', 'Albert', 'Steven']
```

In [91]:

```
for letter, names in itertools.groupby(names, first_letter):
    print(letter, list(names)) # isimler bir generator
```

```
A ['Alan', 'Adam']
W ['Wes', 'Will']
A ['Albert']
S ['Steven']
```

Bazı yararlı itertools işlevleri

combinations(iterable, k) = Yinelenebilir, sırayı göz arı ederek ve değiştirmeden tüm olası k-tuples elemanlarının bir dizisini oluşturur
permutations(iterable, k) = Yinelenebilir, sıraya göre öğelerin olası tüm k-demetlerinin bir dizisini oluşturur
groupby(iterable[, keyfunc]) = Benzersiz for each anahtarı üretir
product(*iterables, repeat=1)= Yuvalanmış for döngüsüne benzer şekilde, giriş yinelemelerinin kartezyen çarpımını tuple olarak oluşturur

Errors and Exception Handling

Veri analizi uygulamalarında, birçok fonksiyon yalnızca belirli tür girdiler üzerinde çalışır. Örnek olarak, Python'un float fonksiyonu bir dizeyi ondalık sayıya çevirebilir, ancak uygun olmayan girdilerde ValueError ile başarısız olur.

In [92]:

```
float('1.2345')
```

Out[92]:

```
1.2345
```

In [96]:

```
# float('something')
Image(filename='img/Picture19.png')
```

Out[96]:

In [93]: `float('something')`

```
-----
ValueError                                                 Traceback (most recent call last)
<ipython-input-93-2649e4ade0e6> in <module>
----> 1 float('something')

ValueError: could not convert string to float: 'something'
```

try / exclude bloğunda float çağrısını içeren bir fonksiyon yazarak kendi yazdığınız hatayı döndürebilirsiniz.

In [97]:

```
def attempt_float(x):
    try:
        return float(x)
    except:
        return x
```

In [98]:

`attempt_float('1.2345')`

Out[98]:

1.2345

In [99]:

`attempt_float('something')`

Out[99]:

'something'

Float ValueError dışında istisnalar oluşturabilir.

In [101]:

```
Image(filename='img/Picture20.png')
# float((1, 2))
```

Out[101]:

In [100]: `float((1, 2))`

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-100-82f777b0e564> in <module>
      1 float((1, 2))
----> 1 TypeError: float() argument must be a string or a number, not 'tuple'
```

Bir `TypeError` programınızda geçerli bir hatayı gösterebileceğinden, yalnızca `ValueError`'ı bastırmak isteyebilirsiniz.

In [102]:

```
def attempt_float(x):
    try:
        return float(x)
    except ValueError:
        return x
```

In [104]:

```
Image(filename='img/Picture21.png')
# attempt_float((1, 2))
```

Out[104]:

In [103]: `attempt_float((1, 2))`

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-103-8b0026e9e6b7> in <module>
      1 attempt_float((1, 2))
----> 1 TypeError: float() argument must be a string or a number, not 'tuple'

<ipython-input-102-6209ddecdb5> 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 number, not 'tuple'
```

Bunun yerine, istisna türlerinden oluşan bir tuple yazarak birden çok istisna türünü yakalayabilirsiniz (parantezler gereklidir).

In [111]:

```
def attempt_float(x):
    try:
        return float(x)
    except (TypeError, ValueError):
        return x
```

Bazı durumlarda, bir istisnayı bastırmak istemeyebilirsiniz, ancak try bloğu ndaki kodun başarılı olup olmadığına bakılmaksızın bazı kodların yürütülmesi ni isteyebilirsiniz.

```
f = open(path, 'w')

try:
    write_to_file(f)
finally:
    f.close()
```

Burada, dosya tanıtıcısı f her zaman kapatılacaktır. Benzer şekilde, yalnızca try: bloğu else kullanmayı başarırsa çalıştırın bir koda sahip olabilirsiniz.

```
f = open(path, 'w')
try:
    write_to_file(f)
except:
    print('Failed')
else:
    print('Succeeded')
finally:
    f.close()
```

IPython'daki istisnalar

Herhangi bir ifadeyi çalıştırırken bir istisna ortaya çıkarsa, IPython varsa yılan olarak yığındaki her noktada konumun etrafında birkaç satırlık bağlam ile tam bir çağrı yığını izi (izleme) yazdıracaktır.

In [113]:

Image(filename='img/Picture22.png')

Out[113]:

```
In [10]: %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:
```

3- Dosyalar ve İşletim Sistemi

Bir dosyayı okumak veya yazmak için , built-in open fonksiyonunu veya bir dosya yolu ile kullanabilirsiniz.

In [10]:

path = 'segismundo.txt'

In [11]:

f = open(path)

Varsayılan olarak, dosya salt okunur mod 'r' ile açılır. Daha sonra f dosya tutamacına bir liste gibi davranabilir ve bu şekilde satırlar üzerinde yineleme yapabilirsiniz.

In [12]:

```
for line in f:
    pass
```

Satırlar, satır sonu (End Of Line) işaretçileri bozulmadan dosyadan çıkar, bunedenle bir dosyada end of line içermeyen satır listesi almak için aşağıda ki kodu sıkılıkla görürsünüz.

In [13]:

```
lines = [x.rstrip() for x in open(path)]
```

In [14]:

```
lines
```

Out[14]:

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

Dosya nesneleri oluşturmak ile ilgili işiniz bittiğinde dosyayı kapatmanız önemlidir. Dosyayı kapatmak, kaynaklarını işletim sistemine geri gönderir.

In [15]:

```
f.close()
```

Açık dosyaları temizlemeyi kolaylaştırmının yollarından biri with ifadesini kullanmaktadır.

In [16]:

```
with open(path) as f:
    lines = [x.rstrip() for x in f]
```

Bu, bloktan çıkışken f dosyasını otomatik olarak kapatacaktır.

Okunabilir dosyalar için en sık kullanılan yöntemlerden bazıları okuma, arama ve anlatmadır. `read` dosyadan belirli sayıda karakter döndürür. Bir "karakteri" neyin oluşturduğu, dosyanın kodlaması (ör. UTF-8) veya dosya ikili moda açılırsa yalnızca ham baytlar tarafından belirlenir.

In [17]:

```
f = open(path)
```

In [18]:

```
f.read(10)
```

Out[18]:

```
'Sueña el r'
```

In [19]:

```
f2 = open(path, 'rb') # Binary mode
```

In [21]:

```
f2.read(10)
```

Out[21]:

```
b'Sue\xc3\xb1la el '
```

Okuma yöntemi, dosya tutamacının konumunu okunan bayt sayısına göre ilerletir. `tell` size mevcut konumu verir.

In [22]:

```
f.tell()
```

Out[22]:

```
11
```

In [24]:

```
f2.tell()
```

Out[24]:

```
10
```

Dosyadan 10 karakter okumamıza rağmen, konum 11'dir çünkü varsayılan kodlamaayı kullanarak 10 karakterin kodunu çözmek bu kadar bayt gerektirdi.

In [25]:

```
import sys
```

In [26]:

```
sys.getdefaultencoding()
```

Out[26]:

```
'utf-8'
```

seek dosya konumunu dosyada belirtilen bayta değiştirir.

In [27]:

```
f.seek(3)
```

Out[27]:

```
3
```

In [28]:

```
f.read(1)
```

Out[28]:

```
'ñ'
```

In [29]:

```
f.close()
```

In [30]:

```
f2.close()
```

Python dosya modları

```
r = Salt okunur modu
w = Yalnızca yazma modu; yeni bir dosya oluşturur (aynı ada sahip herhangi bir dosyanın verilerini silerek)
x = Yalnızca yazma modu; yeni bir dosya oluşturur, ancak dosya yolu zaten mevcutsa başarısız olur
a = Mevcut dosyaya ekle (zaten mevcut değilse dosyayı oluşturun)
r+ = Oku ve yaz
b = İkili dosyalar için moda ekle (yani, 'rb'or'wb')
t = Dosyalar için metin modu (baytları otomatik olarak Unicode'a çözer).
Belirtilmemezse bu varsayılandır. Bunu kullanmak için diğer modlara t ekleyin
(yani, 'rt'or'xt')
```

Bir dosyaya metin yazmak için dosyanın yazma veya yazma yöntemlerini kullanı-

bilirsiniz. Örneğin, böyle boş satırlar olmadan `prof_mod.py`'nin bir sürümünü oluşturabiliriz.

In [31]:

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

In [32]:

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

In [33]:

```
lines
```

Out[33]:

```
['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']
```

Önemli Python dosya yöntemleri veya öznitelikleri

`read([size])` = Dosyadaki verileri, okunacak bayt sayısını belirten isteğe bağlı boyut bağımsız değişkeni ile bir dize olarak döndürür

`readlines([size])` = İsteğe bağlı boyut bağımsız değişkeniyle dosyadaki satırların listesini döndür

`write(str)` = Dosyaya geçirilen dizeyi yaz

`writelines(strings)` = Dosyaya geçirilen dizge dizilerini yaz

`close()` = işlemi kapat

`flush()` = Dahili I/O arabelleğini diske boşaltın

`seek(pos)` = Belirtilen dosya konumuna gitme (integer)

`tell()` = Mevcut dosya konumunu integer olarak döndür

`closed` = Dosya kapalıysa doğrudur

Dosyalar ile Bytes ve Unicode

Python dosyaları için varsayılan davranış (okunabilir veya yazılabilir) metin modudur, bu da Python dizeleriyle (yani Unicode) çalışmayı planladığınız anlamına gelir. Bu, dosya moduna b ekleyerek elde edebileceğiniz binary mode gelişir.

In [34]:

```
with open(path) as f:  
    chars = f.read(10)
```

In [35]:

```
chars
```

Out[35]:

```
'Sueña el r'
```

UTF-8 değişken uzunluklu bir Unicode kodlamasıdır, bu nedenle dosyadan bir kaç karakter istediğimde, Python dosyadan o kadar karakterin kodunu çözmek için yeterli bayt (en az 10 veya 40 bayt olabilir) okur. Dosyayı bunun yerine 'rb' modunda açarsam, read tam bayt sayısını ister.

In [36]:

```
with open(path, 'rb') as f:  
    data = f.read(10)
```

In [37]:

```
data
```

Out[37]:

```
b'Sue\xc3\xb1a el '
```

In [38]:

```
data.decode('utf8')
```

Out[38]:

```
'Sueña el '
```

In [42]:

```
from IPython.display import Image
Image(filename='img/Picture23.png')
# data[:4].decode('utf8')
```

Out[42]:

In [39]: data[:4].decode('utf8')

```
-----
UnicodeDecodeError                         Traceback (most recent call last)
<ipython-input-39-0ad9ad6a11bd> in <module>
----> 1 data[:4].decode('utf8')

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xc3 in position 3: unexpected end of data
```

Text mod, encoding seçeneği, bir Unicode kodlamadan diğerine dönüştürmek için uygun bir yol sağlar.

In [43]:

sink_path = 'sink.txt'

In [44]:

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

In [45]:

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

Sueña el r

Dosyaları ikili mod dışında herhangi bir modda açarken aramayı kullanmaya dikkat edin. Dosya konumu, bir Unicode karakterini tanımlayan baytların ortasına denk gelirse, sonraki okumalar bir hataya neden olur.

In [46]:

f = open(path)

In [47]:

f.read(5)

Out[47]:

'Sueña'

In [48]:

f.seek(4)

Out[48]:

4

In [50]:

Image(filename='img/Picture24.png')
f.read(1)

Out[50]:

In [49]: f.read(1)

```
-----
UnicodeDecodeError                         Traceback (most recent call last)
<ipython-input-49-5a354f952aa4> in <module>
      1 f.read(1)

-/opt/anaconda3/lib/python3.8/codecs.py in decode(self, input, final)
  320     # decode input (taking the buffer into account)
  321     data = self.buffer + input
--> 322     (result, consumed) = self._buffer_decode(data, self.errors, final)
  323     # keep undecoded input until the next call
  324     self.buffer = data[consumed:]

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb1 in position 0: invalid start byte
```

In [51]:

f.close()

D) NumPy Temelleri: Diziler ve Vektörlü Hesaplama

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

- NumPy'de bulacağınız şeylerden bazıları şunlardır:

Tüm veri dizilerinde hızlı işlemler için matematiksel işlevler döngüler yazmak.

- Dizi verilerini diske okumak / yazmak ve bellek eşlemeli dosyalarla çalışmak için araçlar.
- Doğrusal cebir, rastgele sayı üretimi ve Fourier dönüşümü yetenekleri.
- NumPy'yi C, C ++ veya FORTRAN ile yazılmış kitaplıklara bağlamak için bir C API.

NumPy kendi başına modelleme veya bilimsel işlevsellik sağlamazken, NumPy dizilerini ve dizi yönelik hesaplamayı anlamanız, pandas gibi dizi odaklı analamlara sahip araçları çok daha etkili bir şekilde kullanmanıza yardımcı olacaktır.

Çoğu veri analizi uygulaması için odaklanılacak temel işlev alanları şunlardır:

- Veri işleme ve temizleme, alt küme oluşturma ve filtreleme, dönüştürme

ve diğer her türlü hesaplama için hızlı vektörleştirilmiş dizi işlemleri

- Sıralama, benzersiz ve set işlemleri gibi ortak dizi algoritmaları
- Verimli tanımlayıcı istatistikler ve verileri toplama / özetleme
- Heterojen veri kümelerini birleştirmek ve bir araya getirmek için veri hizalama ve ilişkisel veri manipülasyonları
 - Koşullu mantığı, if-elif- else dalları ile döngüler yerine dizi ifadeleri olarak ifade etme
 - Grup bazında veri manipülasyonları (toplama, dönüştürme, işlev uygulaması)

Pandas, NumPy'de bulunmayan, zaman serisi manipülasyonu gibi daha fazla alan'a özgü fonksiyonlar sağlar.

NumPy'nin Python'daki sayısal hesaplamalar için bu kadar önemli olmasının ne denlerinden biri, geniş veri dizileri üzerinde verimlilik için tasarlanmış olmasıdır.

NumPy dizileri Python dizilerinden çok daha az bellek kullanır.

NumPy işlemleri, döngüler için Python'a ihtiyaç duymadan tüm dizilerde karmaşık hesaplamalar gerçekleştirir.

Bir milyon tamsayıdan oluşan bir NumPy dizisini ve eşdeğer Python listesini düşünün.

In [1]:

```
import numpy as np
```

In [2]:

```
my_arr = np.arange(1000000)
```

In [3]:

```
my_list = list(range(1000000))
```

Şimdi her diziyi 2 ile çarpalım.

In [4]:

```
@time for _ in range(10): my_arr2 = my_arr * 2
```

```
CPU times: user 16.7 ms, sys: 9.42 ms, total: 26.2 ms
Wall time: 24.9 ms
```

In [5]:

```
%time for _ in range(10): my_list2 = [x * 2 for x in my_list]
```

```
CPU times: user 602 ms, sys: 160 ms, total: 762 ms
Wall time: 764 ms
```

NumPy tabanlı algoritmalar 10 ila 100 kat daha hızlıdır (veya daha fazla) ve önemli ölçüde daha az bellek kullanır.

1- NumPy ndarray: Çok Boyutlu Bir Dizi Nesnesi

Küçük bir rastgele veri dizisi oluşturalım.

In [6]:

```
import numpy as np
```

In [7]:

```
data = np.random.randn(2, 3)
```

In [8]:

```
data
```

Out[8]:

```
array([[-1.30499323, -0.67655629,  1.07808656],
       [-2.31033129, -1.24564909,  0.69877416]])
```

Daha sonra verilerle matematiksel işlemler yazalım.

In [9]:

```
data * 10
```

Out[9]:

```
array([[-13.04993227,  -6.76556291,   10.78086564],
       [-23.10331294, -12.4564909 ,    6.98774158]])
```

In [10]:

```
data + data
```

Out[10]:

```
array([[-2.60998645, -1.35311258,  2.15617313],
       [-4.62066259, -2.49129818,  1.39754832]])
```

Bir ndarray, veriler için tüm öğeler aynı türde olmalıdır. Her dizinin bir şecli, her boyutun boyutunu belirten bir tuple ve dizinin veri türünü açıklayıp bir nesne olan bir dtype vardır.

In [11]:

```
data.shape
```

Out[11]:

```
(2, 3)
```

In [12]:

```
data.dtype
```

Out[12]:

```
dtype('float64')
```

•Ndarrays oluşturma

In [13]:

```
data1 = [6, 7.5, 8, 0, 1]
```

In [14]:

```
arr1 = np.array(data1)
```

In [15]:

```
arr1
```

Out[15]:

```
array([6., 7.5, 8., 0., 1.])
```

Eşit uzunluklu listelerin bir listesi gibi iç içe geçmiş diziler çok boyutlu bir diziye dönüştürülecektir.

In [16]:

```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
```

In [17]:

```
arr2 = np.array(data2)
```

In [18]:

```
arr2
```

Out[18]:

```
array([[1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

data2 bir liste listesi olduğundan, NumPy dizi arr2'nin veriden çıkarılan şekli ile iki boyutu vardır.

In [19]:

```
arr2.ndim
```

Out[19]:

```
2
```

In [20]:

```
arr2.shape
```

Out[20]:

```
(2, 4)
```

Veri türü, özel bir dtype meta veri nesnesinde saklanır. Örneğin:

In [21]:

```
arr1.dtype
```

Out[21]:

```
dtype('float64')
```

In [22]:

```
arr2.dtype
```

Out[22]:

```
dtype('int64')
```

Np.array ek olarak, yeni diziler oluşturmak için bir dizi başka işlev vardı. Örnek olarak, sıfırlar ve birler, belirli bir uzunluk veya şekilde sırasıyla 0 veya 1 dizileri oluşturur. empty, değerlerini belirli bir değere başlatmadan bir dizi oluşturur.

In [23]:

np.zeros(10)

Out[23]:

array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

In [24]:

np.zeros((3, 6))

Out[24]:

array([[0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0.]])

In [26]:

np.empty((2, 3, 2))

Out[26]:

array([[[2.68156159e+154, -2.68679870e+154],
 [2.96439388e-323, 0.00000000e+000],
 [0.00000000e+000, 1.16095484e-028]],

 [[6.97283618e+228, 3.68008723e-110],
 [6.48224638e+170, 3.67145870e+228],
 [1.28986845e+295, 8.34404819e-309]]])

Np.empty'nin tamamen sıfırlardan oluşan bir dizi döndüreceği kesin değildir.
 Bazı durumlarda Yukarıdaki gibi "çöp" değerler döndürebilir.

arange, ile belirli bir aralıkta dizi oluşturabilirsiniz.

In [27]:

np.arange(15)

Out[27]:

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

Dizi oluşturma fonksiyonları

array = Giriş verilerini (liste, tuple, dizi veya diğer dizi türü) bir d type çıkararak veya açıkça bir dtype belirterek ndarray'e dönüştürür, giriş verilerini varsayılan olarak kopyalar

asarray = Giriş ndarray'e dönüştür, ancak girdi zaten bir ndarray ise k oyalamayın

arange = inrangebut gibi liste yerine ndarray döndürür

`ones, ones_like` = Verilen şekil ve dtype ile tüm 1'lerin bir dizisini oluşturun; `ones_like` başka bir dizi alır ve aynı şekil ve tipte bir birler dizisi üretir

`zeros, zeros_like` = `ones` ve `ones_like` gibi fakat 0'ların dizilerini üretiyor

`empty, empty_like` = Yeni bellek ayırarak yeni diziler oluşturun, ancak birler ve sıfırlar gibi değerlerle doldurmayın

`full, full_like` = Tüm değerleri belirtilen "dolgu değerine" ayarlanmış şekilde verilen şekil ve dtype dizisini üretin `full_like` başka bir dizi alır ve aynı şekil ve dtype ile doldurulmuş bir dizi oluşturur

`eye, identity` = Kare bir $N \times N$ kimlik matrisi oluşturun (köşegende 1'ler ve başka yerlerde 0'lar)

•Ndarrays için Veri Türleri

Veri türü veya dtype, ndarray'in bir bellek yığısını belirli bir veri türü olarak yorumlamak için ihtiyaç duyduğu bilgileri (veya meta verileri, veriler hakkındaki verileri) içeren özel bir nesnedir.

In [28]:

```
arr1 = np.array([1, 2, 3], dtype=np.float64)
```

In [29]:

```
arr2 = np.array([1, 2, 3], dtype=np.int32)
```

In [30]:

```
arr1.dtype
```

Out[30]:

```
dtype('float64')
```

In [31]:

```
arr2.dtype
```

Out[31]:

```
dtype('int32')
```

`dtypes`, NumPy'nin diğer sistemlerden gelen verilerle etkileşim kurma esnekliğiinin bir kaynağıdır. Çoğu durumda, doğrudan temeldeki bir diske veya bellek sunumuna bir eşleme sağlarlar, bu da ikili veri akışlarını diske okumayı ve yazmayı ve ayrıca C veya Fortran gibi düşük seviyeli bir dilde yazılmış kod

a bağlanmayı kolaylaştırır. Sayısal tipler aynı şekilde adlandırılır: float veya int gibi bir tür adı, ardından eleman başına bit sayısını gösteren bir sayı gelir. Standart bir ondalık değeri 8 bayt veya 64 bit kaplar. Bu nedenle, bu tür NumPy'de float64 olarak bilinir.

NumPy veri tipleri

In [34]:

```
from IPython.display import Image
Image(filename='img/Picture25.png')
```

Out[34]:

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,	c8, c16,	Complex numbers represented by two 32, 64, or 128 floats, respectively
complex128,		
complex256	c32	
bool	?	Boolean type storing True and False values
object	0	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')

Ndarray'ın astype yöntemini kullanarak bir diziyi bir dtype'den diğerine açı kçı dönüştürebilir veya çevirebilirsiniz.

In [35]:

```
arr = np.array([1, 2, 3, 4, 5])
```

In [36]:

```
arr.dtype
```

Out[36]:

```
dtype('int64')
```

In [37]:

```
float_arr = arr.astype(np.float64)
```

In [38]:

```
float_arr.dtype
```

Out[38]:

```
dtype('float64')
```

Bu örnekte, tamsayılar ondalık sayılaraya dönüştürülmüştür. Bazı ondalık sayıları, tamsayı dtype olacak şekilde kullanırsanız, ondalık kısım kesilecektir.

In [39]:

```
arr = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
```

In [40]:

```
arr
```

Out[40]:

```
array([ 3.7, -1.2, -2.6,  0.5, 12.9, 10.1])
```

In [41]:

```
arr.astype(np.int32)
```

Out[41]:

```
array([ 3, -1, -2,  0, 12, 10], dtype=int32)
```

Sayılarla ilgili bir diziniz varsa, dönüştürmek için astype kullanabilirsiniz.

In [42]:

```
numeric_strings = np.array(['1.25', '-9.6', '42'], dtype=np.string_)
```

In [43]:

```
numeric_strings.astype(float)
```

Out[43]:

```
array([ 1.25, -9.6 , 42. ])
```

Eğer çevrim herhangi bir nedenle başarısız olursa (float64'e dönüştürülemeyen bir dize gibi), bir ValueError ortaya çıkar. NumPy, Python türlerini kendi eşdeğer veri tiplerine değiştirir.

Ayrıca başka bir dizinin `dtype` attribute'ünü de kullanabilirsiniz.

In [44]:

```
int_array = np.arange(10)
```

In [46]:

```
calibers = np.array([.22, .270, .357, .380, .44, .50], dtype=np.float64)
```

In [47]:

```
int_array.astype(calibers.dtype)
```

Out[47]:

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

Bir `dtype`'a başvurmak için shorthand türü kod dizeleri vardır.

In [48]:

```
empty_uint32 = np.empty(8, dtype='u4')
```

In [49]:

```
empty_uint32
```

Out[49]:

```
array([ 0, 1075314688, 0, 1075707904, 0,
       1075838976, 0, 1072693248], dtype=uint32)
```

NumPy Dizileriyle Aritmetik

Diziler önemlidir, çünkü herhangi bir `for` döngüsü yazmadan veriler üzerinde toplu işlemleri ifade etmenize olanak tanırlar. NumPy kullanıcıları buna ve körleştirmeye adını verir. Eşit boyutlu diziler arasındaki tüm aritmetik işlemler, işlem ögesi bazında uygulanır.

In [50]:

```
arr = np.array([[1., 2., 3.], [4., 5., 6.]])
```

In [51]:

```
arr
```

Out[51]:

```
array([[1., 2., 3.],
       [4., 5., 6.]])
```

In [52]:

```
arr * arr
```

Out[52]:

```
array([[ 1.,  4.,  9.],
       [16., 25., 36.]])
```

In [53]:

```
arr - arr
```

Out[53]:

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [55]:

```
1 / arr
```

Out[55]:

```
array([[1.          , 0.5          , 0.33333333],
       [0.25        , 0.2          , 0.16666667]])
```

In [56]:

```
arr ** 0.5
```

Out[56]:

```
array([[1.          , 1.41421356, 1.73205081],
       [2.          , 2.23606798, 2.44948974]])
```

In [57]:

```
arr2 = np.array([[0., 4., 1.], [7., 2., 12.]])
```

In [58]:

```
arr2
```

Out[58]:

```
array([[ 0.,  4.,  1.],
       [ 7.,  2., 12.]])
```

In [59]:

```
arr2 > arr
```

Out[59]:

```
array([[False,  True, False],
       [ True, False,  True]])
```

Temel İndeksleme ve Dilimleme

In [60]:

```
arr = np.arange(10)
```

In [61]:

```
arr
```

Out[61]:

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

In [62]:

```
arr[5]
```

Out[62]:

```
5
```

In [63]:

```
arr[5:8]
```

Out[63]:

```
array([5, 6, 7])
```

In [64]:

```
arr[5:8] = 12
```

In [65]:

```
arr
```

Out[65]:

```
array([ 0,  1,  2,  3,  4, 12, 12, 12,  8,  9])
```

Gördüğünüz gibi, dizi `[5: 8] = 12`'de olduğu gibi bir dilime skaler bir değer atarsanız, değer tüm seçime yayılır (veya bundan sonra yayınlanır). Python'un yerleşik listelerinden önemli bir ilk ayrim, dizi dilimlerinin orijinal dizideki görünümler olmasıdır. Bu, verilerin kopyalanmadığı ve görünümde yapılacak herhangi bir değişikliğin kaynak diziye yansıtılacağı anlamına gelir.

Buna bir örnek vermek gereklirse:

In [66]:

```
arr_slice = arr[5:8]
```

In [67]:

```
arr_slice
```

Out[67]:

```
array([12, 12, 12])
```

arr_slice'daki değerleri değiştirdiğiniz zaman dizi, dizi içinde yansıtılır.

In [68]:

```
arr_slice[1] = 12345
```

In [69]:

```
arr
```

Out[69]:

```
array([ 0, 1, 2, 3, 4, 12, 12345, 12, 8,
```

"bare" slice [:] bir dizideki tüm değerlere atayacaktır.

In [70]:

```
arr_slice[:] = 64
```

In [71]:

```
arr
```

Out[71]:

```
array([ 0, 1, 2, 3, 4, 64, 64, 64, 8, 9])
```

Daha yüksek boyutlu dizilerle çok daha fazla seçeneğiniz vardır. İki boyutlu bir dizide, her dizindeki öğeler artık skaler değil, tek boyutlu dizilerdir.

In [72]:

```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

In [73]:

```
arr2d[2]
```

Out[73]:

```
array([7, 8, 9])
```

Böylece, tek tek öğelere özyinelemeli olarak erişilebilir. Ancak bu biraz fa

zla iştir, bu nedenle tek tek öğeleri seçmek için virgülle ayrılmış bir indeks listesi kullanabilirsiniz.

In [74]:

```
arr2d[0][2]
```

Out[74]:

3

In [75]:

```
arr2d[0, 2]
```

Out[75]:

3

NumPy dizisindeki elemanların indexlenmesi

In [78]:

```
Image(filename='img/Picture26.png')
```

Out[78]:

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

Çok boyutlu dizilerde, sonraki indisleri atlarsanız, döndürülen nesne, yüksek boyutlar boyunca tüm verileri içeren daha düşük boyutlu bir ndarray olacaktır. Yani $2 \times 2 \times 3$ dizisinde arr3d:

In [79]:

```
arr3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
```

In [80]:

```
arr3d
```

Out[80]:

```
array([[ [ 1,  2,  3],  
        [ 4,  5,  6]],  
  
       [[ 7,  8,  9],  
        [10, 11, 12]]])
```

arr3d [0] bir 2×3 dizisidir.

In [81]:

```
arr3d[0]
```

Out[81]:

```
array([[1, 2, 3],  
       [4, 5, 6]])
```

Hem skaler değerler hem de diziler arr3d [0] 'a atanabilir.

In [82]:

```
old_values = arr3d[0].copy()
```

In [83]:

```
arr3d[0] = 42
```

In [84]:

```
arr3d
```

Out[84]:

```
array([[ [42, 42, 42],  
        [42, 42, 42]],  
  
       [[ 7,  8,  9],  
        [10, 11, 12]]])
```

In [85]:

```
arr3d[0] = old_values
```

In [86]:

```
arr3d
```

Out[86]:

```
array([[ [ 1,  2,  3],  
        [ 4,  5,  6]],  
  
       [[ 7,  8,  9],  
        [10, 11, 12]])
```

Benzer şekilde, arr3d [1, 0] size indisleri (1, 0) ile başlayan tüm değerler i verir.

In [87]:

```
arr3d[1, 0]
```

Out[87]:

```
array([7, 8, 9])
```

Bu ifade, dizine eklemiş olduğumuzla aynıdır.

In [88]:

```
x = arr3d[1]
```

In [89]:

```
x
```

Out[89]:

```
array([[ 7,  8,  9],  
       [10, 11, 12]])
```

In [90]:

```
x[0]
```

Out[90]:

```
array([7, 8, 9])
```

Dizinin alt bölmelerinin seçildiği tüm bu durumlarda döndürülen diziler görü numberlerdir.

1- Dilimlerle indeksleme

Python listeleri gibi tek boyutlu nesneler gibi, ndarray'ler de bilinen sözdizimi ile dilimlenebilir.

In [91]:

```
arr
```

Out[91]:

```
array([ 0,  1,  2,  3,  4, 64, 64, 64,  8,  9])
```

In [92]:

```
arr[1:6]
```

Out[92]:

```
array([ 1,  2,  3,  4, 64])
```

Önceki iki boyutlu diziyi düşünün, arr2d. Bu diziyi dilimlemek biraz farklıdır.

In [93]:

```
arr2d
```

Out[93]:

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

In [94]:

```
arr2d[:2]
```

Out[94]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

Gördüğünüz gibi, birinci eksen olan 0 eksenin boyunca dilimlenmiştir. Bu nedenle bir dilim, bir eksen boyunca bir dizi öğeyi seçer. Arr2d [: 2] ifadesini "arr2d'nin ilk iki satırını seçin" olarak okumak faydalı olabilir.

Birden çok dizini geçirebildiğiniz gibi birden çok dilimi geçirebilirsiniz.

In [95]:

```
arr2d[:2, 1:]
```

Out[95]:

```
array([[2, 3],
       [5, 6]])
```

Bu şekilde dilimleme yaparken, her zaman aynı sayıda boyuta sahip dizi görün

ümeleri elde edersiniz. Tamsayı dizinleri ve dilimleri karıştırarak, daha düşük boyutlu dilimler elde edersiniz.

Örneğin, ikinci satırı seçebilirsiniz ancak yalnızca ilk iki sütunu şu şekilde seçin:

In [96]:

```
arr2d[1, :2]
```

Out[96]:

```
array([4, 5])
```

Benzer şekilde, üçüncü sütunu seçebilirsiniz, ancak ilk iki satır şöyle:

In [97]:

```
arr2d[:2, 2]
```

Out[97]:

```
array([3, 6])
```

In [98]:

```
arr2d[:, :1]
```

Out[98]:

```
array([[1],  
       [4],  
       [7]])
```

Elbette, bir dilim ifadesine atama, tüm seçimi etkiler.

In [99]:

```
arr2d[:2, 1:] = 0
```

In [100]:

```
arr2d
```

Out[100]:

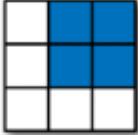
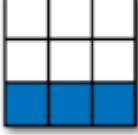
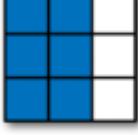
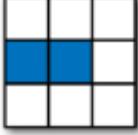
```
array([[1, 0, 0],  
       [4, 0, 0],  
       [7, 8, 9]])
```

iki boyutlu dizi dilimleme

In [102]:

Image(filename='img/Picture27.png')

Out[102]:

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

Boolean İndeksleme

Bir dizide bazı verilerin ve kopyaları olan bir dizi adının olduğu düşünün.

Burada numpy.random dosyasındaki randn işlevini rasgele normal dağıtılmış veriler oluşturmak için kullanacağız.

In [2]:

```
import numpy as np
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
```

In [3]:

```
data = np.random.randn(7, 4)
```

In [4]:

```
names
```

Out[4]:

```
array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')
```

In [5]:

data

Out[5]:

```
array([[ 0.22455152,  0.96433122, -0.0187979 ,  0.66933448],
       [-0.72645195, -1.62929013,  0.26105116,  0.92474528],
       [-2.01531438, -1.7051189 ,  2.44733507,  0.2832372 ],
       [ 1.47876094, -0.87256886, -0.75490499, -1.77170265],
       [ 0.34446199,  1.46530125,  0.1422849 , -0.61847951],
       [ 0.3160729 ,  1.15947519,  1.61562803,  0.05580487],
       [ 1.11143652, -0.11187688,  0.11198104,  1.28528034]])
```

Her bir ismin veri dizisindeki bir satıra karşılık geldiğini varsayıalım ve biz de karşılık gelen 'Bob' ismine sahip tüm satırları seçmek istedik. Aritmetik işlemler gibi, dizilerle karşılaşmalar (== gibi) da vektörleştirilir. Bu nedenle, isimleri 'Bob' dizesiyle karşılaşmak bir boole dizisi verir.

In [6]:

names == 'Bob'

Out[6]:

```
array([ True, False, False,  True, False, False, False])
```

Bu boole dizisi, dizi indekslenirken geçilebilir.

In [7]:

data[names == 'Bob']

Out[7]:

```
array([[ 0.22455152,  0.96433122, -0.0187979 ,  0.66933448],
       [ 1.47876094, -0.87256886, -0.75490499, -1.77170265]])
```

Bu örneklerde, isimlerin == 'Bob' olduğu satırlardan seçim yapıp sütunları indeksliyoruz.

In [8]:

data[names == 'Bob', 2:]

Out[8]:

```
array([[-0.0187979 ,  0.66933448],
       [-0.75490499, -1.77170265]])
```

In [9]:

```
data[names == 'Bob', 3]
```

Out[9]:

```
array([ 0.66933448, -1.77170265])
```

'Bob' dışındaki her şeyi seçmek için != Kullanabilir veya ~ ile koşulu reddedebilirsiniz.

In [10]:

```
names != 'Bob'
```

Out[10]:

```
array([False,  True,  True, False,  True,  True,  True])
```

In [11]:

```
data[~(names == 'Bob')]
```

Out[11]:

```
array([[ -0.72645195, -1.62929013,  0.26105116,  0.92474528],
       [-2.01531438, -1.7051189 ,  2.44733507,  0.2832372 ],
       [ 0.34446199,  1.46530125,  0.1422849 , -0.61847951],
       [ 0.3160729 ,  1.15947519,  1.61562803,  0.05580487],
       [ 1.11143652, -0.11187688,  0.11198104,  1.28528034]])
```

~ Operatörü, bir durumu tersine çevirmek istediğiniz zaman kullanılır.

In [12]:

```
cond = names == 'Bob'
```

In [13]:

```
data[~cond]
```

Out[13]:

```
array([[ -0.72645195, -1.62929013,  0.26105116,  0.92474528],
       [-2.01531438, -1.7051189 ,  2.44733507,  0.2832372 ],
       [ 0.34446199,  1.46530125,  0.1422849 , -0.61847951],
       [ 0.3160729 ,  1.15947519,  1.61562803,  0.05580487],
       [ 1.11143652, -0.11187688,  0.11198104,  1.28528034]])
```

Birden çok koşulunu birleştirmek için & (ve) ve | gibi aritmetik sembollerini kullanın.

In [14]:

```
mask = (names == 'Bob') | (names == 'Will')
```

In [15]:

```
mask
```

Out[15]:

```
array([ True, False,  True,  True,  True, False, False])
```

In [16]:

```
data[mask]
```

Out[16]:

```
array([[ 0.22455152,  0.96433122, -0.0187979 ,  0.66933448],
       [-2.01531438, -1.7051189 ,  2.44733507,  0.2832372 ],
       [ 1.47876094, -0.87256886, -0.75490499, -1.77170265],
       [ 0.34446199,  1.46530125,  0.1422849 , -0.61847951]])
```

Python anahtar sözcükleri and ve or boolean dizileriyle çalışmaz. & (Ve) ve | (veya) ile çalışır.

Verilerdeki tüm negatif değerleri 0 olarak ayarlamak için şunları yapmamız gereklidir.

In [17]:

```
data[data < 0] = 0
```

In [18]:

```
data
```

Out[18]:

```
array([[0.22455152, 0.96433122, 0.          , 0.66933448],
       [0.          , 0.          , 0.26105116, 0.92474528],
       [0.          , 0.          , 2.44733507, 0.2832372 ],
       [1.47876094, 0.          , 0.          , 0.          ],
       [0.34446199, 1.46530125, 0.1422849 , 0.          ],
       [0.3160729 , 1.15947519, 1.61562803, 0.05580487],
       [1.11143652, 0.          , 0.11198104, 1.28528034]])
```

Tek boyutlu bir boolean dizisi kullanarak tüm satırları veya sütunları ayarlamak da kolaydır.

In [19]:

```
data[names != 'Joe'] = 7
```

In [20]:

data

Out[20]:

```
array([[7.        , 7.        , 7.        , 7.        ],
       [0.        , 0.        , 0.26105116, 0.92474528],
       [7.        , 7.        , 7.        , 7.        ],
       [7.        , 7.        , 7.        , 7.        ],
       [7.        , 7.        , 7.        , 7.        ],
       [0.3160729 , 1.15947519, 1.61562803, 0.05580487],
       [1.11143652, 0.          , 0.11198104, 1.28528034]])
```

Fancy Indexing

NumPy tarafından tamsayı dizileri kullanarak indekslemeyi açıklamak için kullanılan bir terimdir.

In [21]:

```
arr = np.empty((8, 4))
```

In [22]:

```
for i in range(8):
    arr[i] = i
```

In [23]:

arr

Out[23]:

```
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.]])
```

Belirli bir sıradaki satırların bir alt kümesini seçmek için, istenen sırayı belirten bir liste veya tamsayı dizisi geçirebilirsiniz.

In [24]:

```
arr[[4, 3, 0, 6]]
```

Out[24]:

```
array([[4., 4., 4., 4.],
       [3., 3., 3., 3.],
       [0., 0., 0., 0.],
       [6., 6., 6., 6.]])
```

Negatif indeksleri kullanmak, sondan satırları seçer.

In [25]:

```
arr[[-3, -5, -7]]
```

Out[25]:

```
array([[5., 5., 5., 5.],
       [3., 3., 3., 3.],
       [1., 1., 1., 1.]])
```

In [27]:

```
arr = np.arange(32).reshape((8, 4))
```

In [28]:

```
arr
```

Out[28]:

```
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 [29]:

```
arr[[1, 5, 7, 2], [0, 3, 1, 2]]
```

Out[29]:

```
array([ 4, 23, 29, 10])
```

Burada (1, 0), (5, 3), (7, 1) ve (2, 2) öğeleri seçildi.

In [30]:

```
arr[[1, 5, 7, 2]][:, [0, 3, 1, 2]]
```

Out[30]:

```
array([[ 4,  7,  5,  6],
       [20, 23, 21, 22],
       [28, 31, 29, 30],
       [ 8, 11,  9, 10]])
```

Dizilerin Transpoze Edilmesi ve Eksenlerin Değiştirilmesi

In [31]:

```
arr = np.arange(15).reshape((3, 5))
```

In [32]:

```
arr
```

Out[32]:

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

In [34]:

```
arr.T# matrisin transpozesi alınacak.
```

Out[34]:

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

In [36]:

```
arr = np.random.randn(6, 3)
```

In [37]:

```
arr
```

Out[37]:

```
array([[-1.79320564,  1.26766081,  0.03154781],
       [ 0.04173907,  0.58811464, -2.22301603],
       [ 0.87306049,  1.19121176,  0.78904739],
       [-0.90758799, -0.37132274, -1.06791571],
       [-0.76761008, -0.67115059, -0.38436709],
       [ 0.16063923,  0.4271646 ,  0.60795606]])
```

In [38]:

```
np.dot(arr.T, arr)
```

Out[38]:

```
array([[ 5.41830938, -0.28781982,  1.90146086],
       [-0.28781982,  4.14262152,  0.58673309],
       [ 1.90146086,  0.58673309,  7.22318391]])
```

Daha yüksek boyutlu diziler için, transpoze eksenleri mutalemek için bir dizî eksen numarası kabul eder.

In [39]:

```
arr = np.arange(16).reshape((2, 2, 4))
```

In [40]:

```
arr
```

Out[40]:

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

In [41]:

```
arr.transpose((1, 0, 2))
```

Out[41]:

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

Burada eksenler, önce ikinci eksen, birinci eksen ikinci ve son eksen değişim eden yeniden sıralanmıştır.

In [42]:

```
arr
```

Out[42]:

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

In [44]:

```
arr.swapaxes(1, 2)# iki farklı 2x4 lük matrisi 4x2 lik yapacak
```

Out[44]:

```
array([[[ 0,  4],
       [ 1,  5],
       [ 2,  6],
       [ 3,  7]],

      [[ 8, 12],
       [ 9, 13],
       [10, 14],
       [11, 15]]])
```

2- Evrensel Fonksiyonlar: Hızlı Eleman Bilge Dizisi Fonksiyonları

Evrensel fonksiyonlar veya ufunc, ndarray'lerdeki veriler üzerinde öğe bazlı işlemler gerçekleştiren bir işlevdir. Bunları, bir veya daha fazla skaler değer alan ve bir veya daha fazla skaler sonuç üreten basit işlevler için hızlı vektörleştirilmiş sarmalayıcılar olarak düşünülebilir.

In [45]:

```
arr = np.arange(10)
```

In [46]:

```
arr
```

Out[46]:

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

In [47]:

```
np.sqrt(arr)
```

Out[47]:

```
array([0.          , 1.          , 1.41421356, 1.73205081, 2.
     , 2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.
     ,])
```

In [48]:

```
np.exp(arr)
```

Out[48]:

```
array([1.00000000e+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])
```

Bunlara tekli ufunclar denir. Toplama veya maksimum gibi diğerleri iki dizide alır ve sonuç olarak tek bir dizi döndürür.

In [49]:

```
x = np.random.randn(8)
```

In [50]:

```
y = np.random.randn(8)
```

In [51]:

```
x
```

Out[51]:

```
array([ 0.93269738, -0.21198339,  0.05080235, -0.59504331,  0.0238234  
8,  
       0.71590276,  0.20048127, -0.44969299])
```

In [52]:

```
y
```

Out[52]:

```
array([-1.72461811,  0.91115767, -1.81061942, -0.92115761, -0.8832143  
6,  
       -0.74624247,  0.73958143,  1.47599279])
```

In [53]:

```
np.maximum(x, y)
```

Out[53]:

```
array([ 0.93269738,  0.91115767,  0.05080235, -0.59504331,  0.0238234  
8,  
       0.71590276,  0.73958143,  1.47599279])
```

Burada numpy.maximum, x ve y deki elemanların eleman bazında maksimumunu hesapladı.

bir ufunc birden çok dizi döndürebilir. modf bir örnektir, bir ondalık dizisinin kesirli ve integral kısımlarını döndürür.

In [54]:

```
arr = np.random.randn(7) * 5
```

In [55]:

```
arr
```

Out[55]:

```
array([ 1.5043928 , -7.80215357, -3.69220978, -0.49738135,  9.2044859
8,
      5.08368936, -1.44564904])
```

In [56]:

```
remainder, whole_part = np.modf(arr)
```

In [57]:

```
remainder
```

Out[57]:

```
array([ 0.5043928 , -0.80215357, -0.69220978, -0.49738135,  0.2044859
8,
      0.08368936, -0.44564904])
```

In [58]:

```
whole_part
```

Out[58]:

```
array([ 1., -7., -3., -0.,  9.,  5., -1.])
```

In [59]:

```
arr
```

Out[59]:

```
array([ 1.5043928 , -7.80215357, -3.69220978, -0.49738135,  9.2044859
8,
      5.08368936, -1.44564904])
```

In [60]:

```
np.sqrt(arr)
```

```
<ipython-input-60-b58949107b3d>:1: RuntimeWarning: invalid value encountered in sqrt
np.sqrt(arr)
```

Out[60]:

```
array([1.22653691,           nan,           nan,           nan,  3.03388958,
2.25470383,           nan])
```

In [61]:

```
np.sqrt(arr, arr)
```

```
<ipython-input-61-164954cb2c14>:1: RuntimeWarning: invalid value encountered in sqrt
  np.sqrt(arr, arr)
```

Out[61]:

```
array([1.22653691,          nan,          nan,          nan, 3.03388958,
       2.25470383,          nan])
```

In [62]:

```
arr
```

Out[62]:

```
array([1.22653691,          nan,          nan,          nan, 3.03388958,
       2.25470383,          nan])
```

Tek terimli ufuncalar

In [63]:

```
from IPython.display import Image
Image(filename='img/Picture28.png')
```

Out[63]:

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>inf</code> , non-NaN) 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

In [65]:

```
Image(filename='img/Picture29.png')
```

Out[65]:

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; fmax ignores NaN
minimum, fmin	Element-wise minimum; fmin 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 >, >=, <, <=, ==, !=)
logical_and, logical_or, logical_xor	Compute element-wise truth value of logical operation (equivalent to infix operators & , ^)

3- Dizilerle Dizi Odaklı Programlama

Açık döngüleri dizi ifadeleriyle değiştirme uygulamasına genellikle vektörleştirme denir. Genel olarak, vektörleştirilmiş dizi işlemleri genellikle saf Python eşdeğerlerinden bir veya iki (veya daha fazla) büyülü sırası daha hızlı olacaktır ve her türlü sayısal hesaplamada en büyük etkiye sahiptir.

Basit bir örnek olarak, `sqrt(x ^ 2 + y ^ 2)` fonksiyonunu normal bir değerler ızgarasında değerlendirmek istediğimizi varsayıyalım. Np.meshgrid işlevi iki 1D dizisi alır ve iki dizideki tüm (x, y) çiftlerine karşılık gelen iki 2D matris üretir.

In [66]:

```
points = np.arange(-5, 5, 0.01) # 1000 eşit aralıklı nokta
```

In [67]:

```
xs, ys = np.meshgrid(points, points)
```

In [68]:

```
ys
```

Out[68]:

```
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]])
```

Şimdi, fonksiyonu değerlendirmek, yazacağınız ifadenin aynısını iki nokta ile yazma meselesidir.

In [69]:

```
z=np.sqrt(xs**2+ys**2)
```

In [70]:

```
z
```

Out[70]:

```
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]])
```

Bu iki boyutlu dizinin görselleştirmelerini oluşturmak için matplotlib kullanacağız.

In [71]:

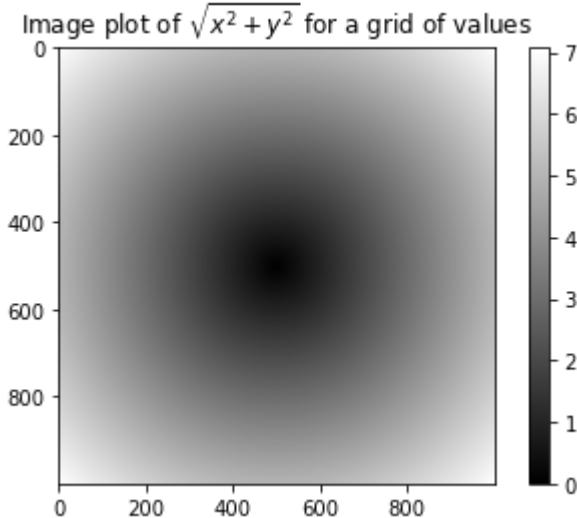
```
import matplotlib.pyplot as plt
```

In [74]:

```
plt.imshow(z, cmap=plt.cm.gray); plt.colorbar()
plt.title("Image plot of  $\sqrt{x^2 + y^2}$  for a grid of values")
```

Out[74]:

```
Text(0.5, 1.0, 'Image plot of  $\sqrt{x^2 + y^2}$  for a grid of value s')
```



Koşullu Mantığı Dizi İşlemleri Olarak İfade Etme

In [75]:

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

In [77]:

```
yarr = np.array([2.1, 2.2, 2.3, 2.4, 2.5])
```

In [79]:

```
cond = np.array([True, False, True, True, False])
```

Koşuldaki karşılık gelen değer True olduğunda xarr'den bir değer almak ve aksi takdirde yarr'dan değer almak istediğimizi varsayıyalım. Bunu yapan bir liste anlayışı şöyle görünebilir:

In [81]:

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

In [82]:

```
result
```

Out[82]:

```
[1.1, 2.2, 1.3, 1.4, 2.5]
```

Büyük diziler için çok hızlı olmayacağından, çok boyutlu dizilerle çalışmaya çalışır. Np. ile bunu çok kısaca yazabilirsiniz.

In [83]:

```
result = np.where(cond, xarr, yarr)
```

In [84]:

```
result
```

Out[84]:

```
array([1.1, 2.2, 1.3, 1.4, 2.5])
```

Np.'nin ikinci ve üçüncü argümanları dizi olmak zorunda değildir; bunlardan biri veya her ikisi skaler olabilir. Veri analizinin tipik bir kullanımı, başka bir diziye dayalı olarak yeni bir değerler dizisi üretmektir. Rastgele oluşturulmuş verilerden oluşan bir matrisiniz olduğunu ve tüm pozitif değerleri 2 ile ve tüm negatif değerleri -2 ile değiştirmek istediğiniz varsayılm. Bunu np.where ile yapmak çok kolaydır.

In [85]:

```
arr = np.random.randn(4, 4)
```

In [86]:

```
arr
```

Out[86]:

```
array([[ 0.45705752, -0.89545863, -0.7046009 , -1.13561517],
       [ 1.09352841,  0.37159746,  0.15903264, -0.74705254],
       [ 0.02068173,  0.0814741 , -0.06415624,  1.64816268],
       [ 0.67947968, -1.19104701, -1.50009779,  0.99851972]])
```

In [87]:

```
arr > 0
```

Out[87]:

```
array([[ True, False, False, False],
       [ True,  True,  True, False],
       [ True,  True, False,  True],
       [ True, False, False,  True]])
```

In [88]:

```
np.where(arr > 0, 2, -2)
```

Out[88]:

```
array([[ 2, -2, -2, -2],
       [ 2,  2,  2, -2],
       [ 2,  2, -2,  2],
       [ 2, -2, -2,  2]])
```

Np.where'i kullanırken skalerleri ve dizileri birleştirebilirsiniz. Örneğin, arr içindeki tüm pozitif değerleri, şu şekilde sabit 2 ile değiştirebilirim.

In [90]:

```
np.where(arr > 0, 2, arr) # sadece pozitif değerleri 2 olarak ayarla
```

Out[90]:

```
array([[ 2.          , -0.89545863, -0.7046009 , -1.13561517],
       [ 2.          ,  2.          ,  2.          , -0.74705254],
       [ 2.          ,  2.          , -0.06415624,  2.          ],
       [ 2.          , -1.19104701, -1.50009779,  2.          ]])
```

Matematiksel ve İstatistiksel Yöntemler

Bir dizinin tamamı veya bir eksen boyunca verilerle ilgili istatistikleri hesaplayan bir dizi matematiksel işlev, dizi sınıfının yöntemleri olarak erişilebilir. Toplam, ortalama ve std (standart sapma) gibi toplamaları (genellikle indirgeme olarak adlandırılır) ya dizi örneği yöntemini çağırarak ya da üst düzey NumPy işlevini kullanarak kullanabilirsiniz.

In [91]:

```
arr = np.random.randn(5, 4)
```

In [92]:

```
arr
```

Out[92]:

```
array([[ 1.0037799 , -1.07826314,  0.52308101, -1.18611347],
       [-0.7433188 ,  1.80998163, -1.03970367, -1.05382978],
       [-0.31090767, -0.44056377,  1.81178469,  0.7345561 ],
       [-0.0251316 , -1.17620804,  1.40443334, -0.48210486],
       [-0.44811163, -0.65747819,  0.68312086, -0.03346944]])
```

In [93]:

```
arr.mean()
```

Out[93]:

```
-0.03522332717118447
```

In [94]:

```
np.mean(arr)
```

Out[94]:

```
-0.03522332717118447
```

In [95]:

```
arr.sum()
```

Out[95]:

```
-0.7044665434236894
```

Ortalama ve toplam gibi fonksiyonlar, verilen eksen üzerindeki istatistiği hesaplayan isteğe bağlı bir eksen argümanı alır ve sonuçta daha az boyutlu bir dizi elde edilir.

In [96]:

```
arr.mean(axis=1)
```

Out[96]:

```
array([-0.18437892, -0.25671765,  0.44871734, -0.06975279, -0.1139846])
```

In [97]:

```
arr.sum(axis=0)
```

Out[97]:

```
array([-0.5236898 , -1.54253151,  3.38271623, -2.02096146])
```

Burada, arr.mean (1) "sütunların ortalamasını hesapla" anlamına gelir; burada arr.sum (0) "satırların toplamını hesapla" anlamına gelir.

Cumsum ve cumprod gibi diğer yöntemler bir araya gelmez, bunun yerine bir dizি ara sonuç üretir.

In [98]:

```
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7])
```

In [99]:

```
arr.cumsum()
```

Out[99]:

```
array([ 0,  1,  3,  6, 10, 15, 21, 28])
```

Çok boyutlu dizilerde, cumsum gibi toplama işlevleri aynı boyutta bir dizi döndürür, ancak her bir alt boyutlu dilime göre belirtilen eksen boyunca hesaplanan kısmi toplamalar ile döndürür.

In [100]:

```
arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
```

In [101]:

```
arr
```

Out[101]:

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

In [102]:

```
arr.cumsum(axis=0)
```

Out[102]:

```
array([[ 0,  1,  2],
       [ 3,  5,  7],
       [ 9, 12, 15]])
```

In [103]:

```
arr.cumprod(axis=1)
```

Out[103]:

```
array([[ 0,  0,  0],
       [ 3, 12, 60],
       [ 6, 42, 336]])
```

Temel dizi istatistiksel yöntemler

In [105]:

Image(filename='img/Picture30.png')

Out[105]:

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 Diziler için Yöntemler

Boolean değerleri, önceki yöntemlerde 1 (True) ve 0 (False) zorlanır. Bu nedenle, sum genellikle bir boolean dizisindeki True değerleri saymanın bir yolunu olarak kullanılır.

In [106]:

arr = np.random.randn(100)

In [107]:

(arr > 0).sum() # Pozitif değerlerin sayısı

Out[107]:

48

any bir dizideki bir veya daha fazla değerin True olup olmadığını test ederken all, her değerin True olup olmadığını kontrol eder.

In [108]:

bools = np.array([False, False, True, False])

In [109]:

```
bools.any()
```

Out[109]:

```
True
```

In [110]:

```
bools.all()
```

Out[110]:

```
False
```

Sıralama

In [111]:

```
arr = np.random.randn(6)
```

In [112]:

```
arr
```

Out[112]:

```
array([ 0.12509188,  1.69171022, -0.86209065, -0.01673704, -0.72651531,  
       -0.21287397])
```

In [113]:

```
arr.sort()
```

In [114]:

```
arr
```

Out[114]:

```
array([-0.86209065, -0.72651531, -0.21287397, -0.01673704,  0.12509188,  
      1.69171022])
```

Sıralamak için eksen numarasını ileterek, çok boyutlu bir dizideki değerleri n her bir boyutlu bölümünü bir eksen boyunca yerinde sıralayabilirsiniz.

In [115]:

```
arr = np.random.randn(5, 3)
```

In [116]:

arr

Out[116]:

```
array([[-3.26421805, -0.75524454, -0.86615591],
       [-0.54091939, -0.3264808 , -0.14022583],
       [-1.49432601,  1.00600517, -0.54349184],
       [-0.02670825, -0.18870903,  0.18060168],
       [ 0.47691174, -0.60343593, -0.27432372]])
```

In [117]:

arr.sort(1)

In [118]:

arr

Out[118]:

```
array([[-3.26421805, -0.86615591, -0.75524454],
       [-0.54091939, -0.3264808 , -0.14022583],
       [-1.49432601, -0.54349184,  1.00600517],
       [-0.18870903, -0.02670825,  0.18060168],
       [-0.60343593, -0.27432372,  0.47691174]])
```

np.sort, diziyi yerinde değiştirmek yerine dizinin sıralanmış bir kopyasını döndürür. Bir dizinin niceliklerini hesaplamanın hızlı ve kirli bir yolu, diziyi sıralamak ve belirli bir sıradaki değeri seçmektir.

In [119]:

large_arr = np.random.randn(1000)

In [120]:

large_arr.sort()

In [121]:

large_arr[int(0.05 * len(large_arr))] # 5% çeyreklik

Out[121]:

-1.641686377001675

Eşsiz ve Diğer Küme Mantığı

In [122]:

names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])

In [123]:

```
np.unique(names)
```

Out[123]:

```
array(['Bob', 'Joe', 'Will'], dtype='<U4')
```

In [124]:

```
ints = np.array([3, 3, 3, 2, 2, 1, 1, 4, 4])
```

In [125]:

```
np.unique(ints)
```

Out[125]:

```
array([1, 2, 3, 4])
```

In [126]:

```
sorted(set(names))
```

Out[126]:

```
['Bob', 'Joe', 'Will']
```

In [127]:

```
values = np.array([6, 0, 0, 3, 2, 5, 6])
```

In [128]:

```
np.in1d(values, [2, 3, 6])
```

Out[128]:

```
array([ True, False, False,  True,  True, False,  True])
```

Dizi kümesi işlemleri

In [130]:

```
Image(filename='img/Picture31.png')
```

Out[130]:

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

4- Dizilerle Dosya Girişi ve Çıkışı

NumPy, metin veya ikili formatta verileri kaydedebilir ve diske yükleyebilir.

`np.save` ve `np.load`, diskteki dizi verilerini verimli bir şekilde kaydetmek ve yüklemek için en çok kullanılan iki işlevdir. Diziler varsayılan olarak `.npy` dosya uzantısıyla sıkıştırılmamış ham ikili biçimde kaydedilir.

In [131]:

```
arr = np.arange(10)
```

In [132]:

```
np.save('some_array', arr)
```

Dosya yolu zaten `.npy` ile bitmiyorsa, uzantı eklenecektir. Diskteki dizi daha sonra `np.load` ile yüklenebilir.

In [133]:

```
np.load('some_array.npy')
```

Out[133]:

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

`np.savez` kullanarak ve dizileri anahtar kelime argümanları olarak ileterek sıkıştırılmamış bir arşive birden çok dizi kaydedersiniz.

In [134]:

```
np.savez('array_archive.npz', a=arr, b=arr)
```

Bir `.npz` dosyası yüklerken, tek tek dosyayı yükleyen dikteye benzer bir nesne alırsınız.

In [135]:

```
arch = np.load('array_archive.npz')
```

In [136]:

```
arch['b']
```

Out[136]:

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

Verileriniz iyi sıkıştırılırsa, bunun yerine `numpy.savez_compressed` kullanab

ilirsiniz.

In [137]:

```
np.savez_compressed('arrays_compressed.npz', a=arr, b=arr)
```

Lineer Cebir

iki iki boyutlu diziyi * ile çarpmak matris nokta çarpımı yerine eleman bazlı bir çarpımdır. Dolayısıyla, matris çarpımı için numpy ad alanında hem bir dizi yöntemi hem de bir işlev olan bir işlev noktası vardır.

In [138]:

```
x = np.array([[1., 2., 3.], [4., 5., 6.]])
```

In [139]:

```
y = np.array([[6., 23.], [-1, 7], [8, 9]])
```

In [141]:

```
x
```

Out[141]:

```
array([[1., 2., 3.],
       [4., 5., 6.]])
```

In [142]:

```
y
```

Out[142]:

```
array([[ 6., 23.],
       [-1., 7.],
       [ 8., 9.]])
```

In [144]:

```
x.dot(y) # x.dot (y), np.dot (x, y) ile eşdeğerdir:
```

Out[144]:

```
array([[ 28., 64.],
       [ 67., 181.]])
```

In [145]:

```
np.dot(x, y)
```

Out[145]:

```
array([[ 28., 64.],
       [ 67., 181.]])
```

İki boyutlu bir dizi ile uygun boyutlu tek boyutlu bir dizi arasındaki bir matris çarpımı, tek boyutlu bir dizi ile sonuçlanır.

In [146]:

```
np.dot(x, np.ones(3))
```

Out[146]:

```
array([ 6., 15.])
```

@ Sembolü ayrıca matris çarpımını gerçekleştiren bir infix operatörü olarak da çalışır.

In [147]:

```
x @ np.ones(3)
```

Out[147]:

```
array([ 6., 15.])
```

numpy.linalg, standart bir matris ayrıştırma kümесine ve ters ve determinant gibi şeylere sahiptir.

In [148]:

```
from numpy.linalg import inv, qr
```

In [149]:

```
X = np.random.randn(5, 5)
```

In [150]:

```
mat = X.T.dot(X)
```

In [151]:

```
inv(mat)
```

Out[151]:

```
array([[ 0.86702281, -0.52764206,  4.09153236,  1.05653846, -0.23634483],
       [-0.52764206,  0.44700047, -2.91944237, -0.71981258,  0.31601942],
       [ 4.09153236, -2.91944237, 28.14840772,  6.34152159, -1.92413454],
       [ 1.05653846, -0.71981258,  6.34152159,  2.12802299,  0.14394283],
       [-0.23634483,  0.31601942, -1.92413454,  0.14394283,  1.90675529]])
```

In [152]:

```
mat.dot(inv(mat))
```

Out[152]:

```
array([[ 1.00000000e+00, -4.59919063e-17,  5.59410789e-15,
       1.86486656e-16, -5.99889367e-17],
      [-7.68000606e-16,  1.00000000e+00, -5.83633059e-15,
       -8.42943492e-16,  6.96411557e-17],
      [-1.06480916e-16,  7.38917028e-17,  1.00000000e+00,
       5.63691749e-17,  1.71908411e-16],
      [-8.93158603e-17,  1.63778331e-16, -1.74472855e-15,
       1.00000000e+00, -1.84392208e-16],
      [ 1.38854810e-17,  3.06594696e-17, -7.87689604e-16,
       2.37855901e-18,  1.00000000e+00]])
```

In [153]:

```
q, r = qr(mat)
```

In [154]:

```
r
```

Out[154]:

```
array([[ -6.66148674, -9.48404403,  0.08250637, -0.16713821,  0.8585799
5],
       [ 0.          , -7.3696153 , -0.2284594 , -2.02476524,  1.2115277
7],
       [ 0.          ,  0.          , -0.52478655,  2.0169974 , -0.8595136
5],
       [ 0.          ,  0.          ,  0.          , -0.19273566, -0.3103062
3],
       [ 0.          ,  0.          ,  0.          ,  0.          ,  0.          ,
0.3647969
7]])
```

Yaygın olarak kullanılan numpy.linalg işlevleri

In [155]:

```
Image(filename='img/Picture32.png')
```

Out[155]:

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 $Ax = b$

6- Pseudorandom (Sözde Rasgele) Sayı Üretilimi

In [157]:

```
samples = np.random.normal(size=(4, 4))
```

In [158]:

```
samples
```

Out[158]:

```
array([[-5.29773034e-01, -1.80756156e-01, -4.16408367e-01,
       9.83040737e-01],
       [ 8.25428759e-01, -2.65529313e-01, -1.02223095e+00,
      -8.02727393e-01],
       [-8.80394238e-04, -1.34220032e+00,  8.87786069e-01,
       5.51799892e-01],
       [-1.64918727e+00,  1.01851868e-02,  4.61589399e-01,
      -5.92049671e-01]])
```

random modülü, bir seferde yalnızca bir değeri örnekler. Bu karşılaştırmadan da görebileceğiniz gibi, numpy.random, çok büyük örnekler oluşturmak için bir kat daha hızlıdır.

In [159]:

```
from random import normalvariate
```

In [160]:

```
N = 1000000
```

In [161]:

```
%timeit samples = [normalvariate(0, 1) for _ in range(N)]
```

582 ms ± 17.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

In [162]:

```
%timeit np.random.normal(size=N)
```

23.9 ms ± 1.65 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

Bunların sözde rasgele sayılar olduğunu söyleyiyoruz çünkü bunlar, rasgele sayı üreticisinin tohumuna dayalı deterministik davranışa sahip bir algoritma tarafından üretiliyorlar. NumPy'nin rastgele sayı oluşturma tohumunu `np.random.seed` kullanarak değiştirebilirsiniz.

In [163]:

```
np.random.seed(1234)
```

Numpy.random'daki veri oluşturma işlevleri, genel bir rastgele çekirdek kullanır. Global durumdan kaçınmak için, diğerlerinden izole edilmiş bir rastgele sayı oluşturucu oluşturmak için `numpy.random.RandomState`'i kullanabilirsiniz.

In [164]:

```
rng = np.random.RandomState(1234)
```

In [165]:

```
rng.randn(10)
```

Out[165]:

```
array([ 0.47143516, -1.19097569,  1.43270697, -0.3126519 , -0.72058873,
       0.88716294,  0.85958841, -0.6365235 ,  0.01569637, -2.24268495])
```

numpy.random fonksiyonları

In [166]:

Image(filename='img/Picture33.png')

Out[166]:

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

7- Örnek: Random Walks

Random Walks, dizi işlemlerinin kullanılmasının açıklayıcı bir uygulamasını sağlar. Önce 0'dan başlayıp 1 ve -1 adımlarıyla eşit olasılıkla gerçekleşen basit bir rastgele yürüyüşü düşünelim.

Yerleşik rastgele modülü kullanarak 1.000 adımlık tek bir rastgele yürüyüş gerçekleştirmenin yolu:

In [167]:

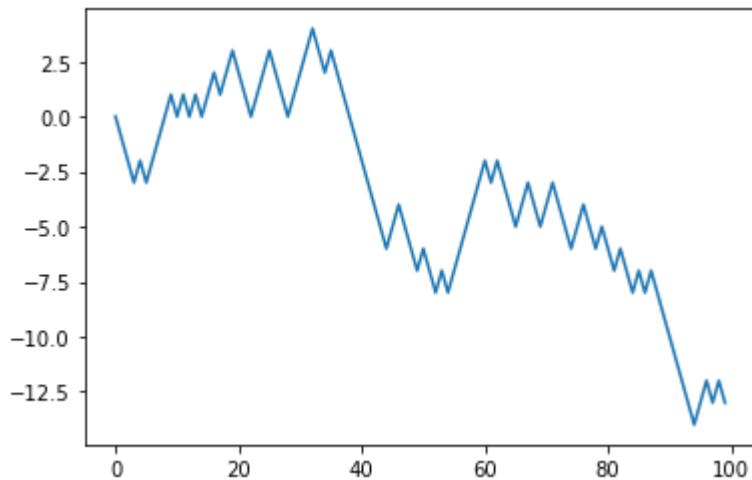
```
import random
position = 0
walk = [position]
steps = 1000
for i in range(steps):
    step = 1 if random.randint(0, 1) else -1
    position += step
    walk.append(position)
```

In [169]:

```
plt.plot(walk[:100]) # Random Walks ilk 100 değerin örnek bir çizimi
```

Out[169]:

```
[<matplotlib.lines.Line2D at 0x7fd65f6390a0>]
```



In [170]:

```
nsteps = 1000
```

In [171]:

```
draws = np.random.randint(0, 2, size=nsteps)
```

In [173]:

```
steps = np.where(draws > 0, 1, -1)
```

In [174]:

```
walk = steps.cumsum()
```

Bundan yürüyüşün yörüngesi boyunca minimum ve maksimum istatistikleri çıkarmaya başlayabiliriz.

In [175]:

```
walk.min()
```

Out[175]:

-9

In [176]:

```
walk.max()
```

Out[176]:

```
60
```

`np.abs(walk) >= 10` bize yürüme değerinin 10'a ulaştığı veya onu geçtiği yeri gösteren bir boolean dizisi verir, ancak biz ilk 10'un veya -10'un dizini ni istiyoruz. Bunu, boolean dizisindeki maksimum değerin ilk indeksini döndüren `argmax` kullanarak hesaplayabiliriz (True maksimum değerdir).

In [177]:

```
(np.abs(walk) >= 10).argmax()
```

Out[177]:

```
297
```

Aynı Anda Birçok Random Walks Simüle Etme

Amacınız birçok rastgele yürüyüşü simüle etmekse, mesela 5.000, önceki kodda küçük değişikliklerle tüm rastgele yürüyüşleri oluşturabilirsiniz.

In [2]:

```
nwalks = 5000
```

In [3]:

```
nsteps = 1000
```

In [5]:

```
import numpy as np
draws = np.random.randint(0, 2, size=(nwalks, nsteps)) # 0 ya da 1
```

In [6]:

```
steps = np.where(draws > 0, 1, -1)
```

In [7]:

```
walks = steps.cumsum(1)
```

In [8]:

```
walks
```

Out[8]:

```
array([[ 1,  2,  1, ..., -18, -19, -20],  
       [ 1,  0,  1, ..., -34, -35, -36],  
       [ 1,  0,  1, ...,  18,  19,  20],  
       ...,  
       [ 1,  2,  1, ...,  20,  19,  18],  
       [-1, -2, -3, ...,  18,  19,  18],  
       [-1,  0, -1, ..., -6, -5, -4]])
```

In [9]:

```
walks.max()
```

Out[9]:

```
124
```

In [10]:

```
walks.min()
```

Out[10]:

```
-133
```

Bu yürüyüşlerden minimum geçiş süresini 30 veya -30 olarak hesaplayalım. Bu biraz yaniltıcıdır, çünkü 5.000, 30'a ulaşmaz. Bunu herhangi bir yöntemi kullanarak kontrol edebiliriz.

In [11]:

```
hits30 = (np.abs(walks) >= 30).any(1)
```

In [12]:

```
hits30
```

Out[12]:

```
array([ True,  True,  True, ...,  True, False, False])
```

In [13]:

```
hits30.sum() # 30 veya -30'a ulaşan sayı
```

Out[13]:

```
3430
```

Mutlak geçiş sürelerini elde etmek için eksen 1 boyunca argmax çağırın.

In [14]:

```
crossing_times = (np.abs(walks[hits30]) >= 30).argmax(1)
```

In [15]:

```
crossing_times.mean()
```

Out[15]:

504.268804664723

In [16]:

```
steps = np.random.normal(loc=0, scale=0.25,
                         size=(nwalks, nsteps))
```

E) Pandas

Python'da veri temizleme ve analizini hızlı ve kolay hale getirmek için taraflanmış veri yapıları ve veri işleme araçları içerir. Pandalar genellikle NumPy ve SciPy gibi sayısal hesaplama araçları, istatistik modelleri ve scikit-learn gibi analitik kitaplıklar ve matplotlib gibi veri görselleştirme kitalıkları ile birlikte kullanılır. Pandalar, NumPy'nin deyimsel dizi tabanlı bilgi işlem tarzının önemli kısımlarını, özellikle dizi tabanlı işlevleri ve döngüler olmadan veri işleme tercihini benimser.

In [17]:

```
import pandas as pd
```

In [18]:

```
from pandas import Series, DataFrame
```

1- Pandas Veri Yapılarına Giriş

Series ve DataFrame.

Dizi

Bir Dizi, bir dizi değer (NumPy türlerine benzer türlerde) ve dizini adı verilen ilişkili bir veri etiketleri dizisi içeren tek boyutlu dizi benzeri bir nesnedir. En basit Seri, yalnızca bir veri dizisinden oluşur.

In [19]:

```
obj = pd.Series([4, 7, -5, 3])
```

In [20]:

```
obj
```

Out[20]:

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

Bir Serinin dize temsili, soldaki tanımlamayı ve sağdaki değerleri gösterir. Veriler için bir dizin belirtmediğimizden, 0 ile N - 1 arasındaki tam sayılar dan (burada N, verilerin uzunluğu) oluşan varsayılan bir dizin oluşturulur. Dizinin dizi gösterimini ve dizin nesnesini sırasıyla değerleri ve dizin nitelikleri aracılığıyla alabilirsiniz.

In [21]:

```
obj.values
```

Out[21]:

```
array([ 4,  7, -5,  3])
```

In [22]:

```
obj.index # range(4) gibi
```

Out[22]:

```
RangeIndex(start=0, stop=4, step=1)
```

Çoğu zaman, her veri noktasını bir etiketle tanımlayan bir dizine sahip bir Seri oluşturmak istenir.

In [23]:

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

In [24]:

```
obj2
```

Out[24]:

```
d    4  
b    7  
a   -5  
c    3  
dtype: int64
```

In [25]:

```
obj2.index
```

Out[25]:

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

NumPy dizileriyle karşılaşıldığında, tek bir değer veya bir değer kümesi seçerken dizindeki etiketleri kullanabilirsiniz.

In [26]:

```
obj2['a']
```

Out[26]:

```
-5
```

In [27]:

```
obj2['d'] = 6
```

In [28]:

```
obj2[['c', 'a', 'd']]
```

Out[28]:

```
c    3  
a   -5  
d    6  
dtype: int64
```

Burada ['c', 'a', 'd'], tamsayılar yerine dizeler içерse de, bir dizin liste si olarak yorumlanır.

Boolean dizisiyle filtreleme, skaler çarpma veya matematik fonksiyonları uyg ulama gibi NumPy fonksiyonları veya NumPy benzeri işlemleri kullanmak, dizin değerini koruyacaktır.

In [29]:

```
obj2[obj2 > 0]
```

Out[29]:

```
d    6  
b    7  
c    3  
dtype: int64
```

In [30]:

```
obj2 * 2
```

Out[30]:

```
d    12  
b    14  
a   -10  
c     6  
dtype: int64
```

In [31]:

```
np.exp(obj2)
```

Out[31]:

```
d    403.428793  
b   1096.633158  
a     0.006738  
c    20.085537  
dtype: float64
```

In [32]:

```
'b' in obj2
```

Out[32]:

```
True
```

In [33]:

```
'e' in obj2
```

Out[33]:

```
False
```

Bir Python dict'inde yer alan veriye sahipseniz, dict'i ileterek ondan bir S
eri oluşturabilirsiniz.

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 [39]:

```
obj4 = pd.Series(sdata, index=states)
```

In [40]:

```
obj4
```

Out[40]:

```
California      NaN
Ohio          35000.0
Oregon         16000.0
Texas          71000.0
dtype: float64
```

Burada, sdata'da bulunan üç değer uygun yerlere yerleştirilmiştir, ancak 'California' için bir değer bulunamadığından, pandas eksik veya NA değerlerini işaretlemek için düşünülen NaN (sayı değil) olarak görünür. 'Utah' eyaletle re dahil edilmemişinden, ortaya çıkan nesneden hariç tutulmuştur.

Pandas'daki isnull ve notnull fonksiyonları, eksik verileri tespit etmek içi n kullanılır.

In [41]:

```
pd.isnull(obj4)
```

Out[41]:

```
California    True
Ohio        False
Oregon       False
Texas        False
dtype: bool
```

In [42]:

```
pd.notnull(obj4)
```

Out[42]:

```
California    False
Ohio          True
Oregon         True
Texas          True
dtype: bool
```

In [43]:

```
obj4.isnull()
```

Out[43]:

```
California    True
Ohio          False
Oregon         False
Texas          False
dtype: bool
```

Birçok uygulama için kullanışlı bir Seri özelliği, dizine göre otomatik olarak hizalanmasıdır.

In [44]:

```
obj3
```

Out[44]:

```
Ohio      35000
Texas     71000
Oregon    16000
Utah      5000
dtype: int64
```

In [45]:

```
obj4
```

Out[45]:

```
California      NaN
Ohio        35000.0
Oregon      16000.0
Texas        71000.0
dtype: float64
```

In [46]:

```
obj3 + obj4
```

Out[46]:

```
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
```

Bir Serinin dizini, assignment yoluyla değiştirilebilir.

In [50]:

```
obj
```

Out[50]:

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

In [51]:

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

In [52]:

obj

Out[52]:

```
Bob      4
Steve    7
Jeff     -5
Ryan     3
dtype: int64
```

2- DataFrame

DataFrame, bir veri tablosunu temsil eder ve her biri farklı bir değer türü (sayısal, dize, boolean, vb.) Olabilen sıralı bir sütun içerir. DataFrame'in hem satır hem de sütun indeksi vardır. Hepsi aynı dizini paylaşan bir Seri dikteleri olarak düşünülebilir. Başlık altında, veriler bir liste, dikte veya tek boyutlu dizilerin başka bir koleksiyonu yerine bir veya daha fazla iki boyutlu blok olarak depolanır.

Bir DataFrame oluşturmanın birçok yolu vardır, ancak en yaygın olanlardan biri eşit uzunlukta listelerden veya NumPy dizilerinden oluşur.

In [54]:

```
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)
```

Sonuçta elde edilen DataFrame'in dizini Series ile olduğu gibi otomatik olarak atanır ve sütunlar sıralı olarak yerleştirilir.

In [55]:

frame

Out[55]:

	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

Büyük DataFrames için, head yöntemi yalnızca ilk beş satırı seçer.

In [56]:

```
frame.head()
```

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

Bir sütun dizisi belirtirseniz, DataFrame'in sütunları şu sırayla düzenlenir:

In [57]:

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

Out[57]:

	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

Eğer diktede yer almayan bir sütun olursa, sonuçta eksik değerlerle görünecektir.

In [58]:

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

In [59]:

```
frame2
```

Out[59]:

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

In [60]:

```
frame2.columns
```

Out[60]:

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

DataFrame'deki bir sütun, dikte benzeri gösterimle Seri olarak alınabilir.

In [61]:

```
frame2['state']
```

Out[61]:

```
one      Ohio
two      Ohio
three    Ohio
four     Nevada
five     Nevada
six     Nevada
Name: state, dtype: object
```

In [62]:

```
frame2.year
```

Out[62]:

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

!!Döndürülen Serilerin DataFrame ile aynı dizine sahip olduğunu ve ad öznite liğinin uygun şekilde ayarlandığını unutmayın.

Satırlar, özel loc niteliğiyle konum veya ada göre de alınabilir.

In [63]:

```
frame2.loc['three']
```

Out[63]:

```
year      2002
state    Ohio
pop       3.6
debt      NaN
Name: three, dtype: object
```

Sütunlar, atama ile değiştirilebilir. Örneğin, empty 'debt' sütununa skaler bir değer veya bir değerler dizisi atanabilir.

In [64]:

```
frame2['debt'] = 16.5
```

In [65]:

```
frame2
```

Out[65]:

	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 [66]:

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

In [67]:

frame2

Out[67]:

	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

Bir sütuna listeler veya diziler atarken, değerin uzunluğu DataFrame'in uzunluğuyla eşleşmelidir. Bir Seri atarsanız, etiketleri DataFrame'in dizinine tam olarak yeniden hizalanacak ve herhangi bir deliğe eksik değerler eklenecektir.

In [68]:

val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])

In [69]:

frame2['debt'] = val

In [70]:

frame2

Out[70]:

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2002	Nevada	2.9	-1.7
six	2003	Nevada	3.2	NaN

Var olmayan bir sütun atamak yeni bir sütun oluşturacaktır. Del anahtar sözcüğü sütunları olduğu gibi silecektir.

In [71]:

```
frame2['eastern'] = frame2.state == 'Ohio'
```

In [72]:

```
frame2
```

Out[72]:

	year	state	pop	debt	eastern
one	2000	Ohio	1.5	NaN	True
two	2001	Ohio	1.7	-1.2	True
three	2002	Ohio	3.6	NaN	True
four	2001	Nevada	2.4	-1.5	False
five	2002	Nevada	2.9	-1.7	False
six	2003	Nevada	3.2	NaN	False

In [73]:

```
del frame2['eastern']
```

In [74]:

```
frame2.columns
```

Out[74]:

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

Diğer bir yaygın veri biçimi, iç içe geçmiş bir diktedir.

In [76]:

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

İç içe geçmiş dikte DataFrame'e iletilirse, pandalar dış dikte anahtarlarını sütunlar olarak ve iç anahtarları satır dizinleri olarak yorumlar.

In [79]:

```
frame3 = pd.DataFrame(pop)
```

In [80]:

frame3

Out[80]:

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

DataFrame'i (satırları ve sütunları takas) benzer sözdizimiyle bir NumPy dizisine dönüştürebilirsiniz.

In [81]:

frame3.T

Out[81]:

	2001	2002	2000
Nevada	2.4	2.9	NaN
Ohio	1.7	3.6	1.5

In [82]:

pd.DataFrame(pop, index=[2001, 2002, 2003])

Out[82]:

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2003	NaN	NaN

In [83]:

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

In [84]:

pd.DataFrame(pdata)

Out[84]:

	Ohio	Nevada
2001	1.7	2.4
2002	3.6	2.9

Bir DataFrame'in dizini ve sütunlarının ad öznitelikleri ayarlanmışsa, bunlar da görüntülenir.

In [86]:

```
frame3.index.name = 'year'; frame3.columns.name = 'state'
```

In [87]:

```
frame3
```

Out[87]:

	state	Nevada	Ohio
year			
2001	2.4	1.7	
2002	2.9	3.6	
2000	NaN	1.5	

Series'de olduğu gibi, values özniteliği, DataFrame'de bulunan verileri iki boyutlu bir ndarray olarak döndürür.

In [88]:

```
frame3.values
```

Out[88]:

```
array([[2.4, 1.7],
       [2.9, 3.6],
       [nan, 1.5]])
```

DataFrame'in sütunları farklı type ise, değerler dizisinin dtype'ı, tüm sütunları barındıracak şekilde seçilir.

In [89]:

```
frame2.values
```

Out[89]:

```
array([[2000, 'Ohio', 1.5, nan],
       [2001, 'Ohio', 1.7, -1.2],
       [2002, 'Ohio', 3.6, nan],
       [2001, 'Nevada', 2.4, -1.5],
       [2002, 'Nevada', 2.9, -1.7],
       [2003, 'Nevada', 3.2, nan]], dtype=object)
```

DataFrame yapıcısına olası veri girişleri

In [110]:

```
from IPython.display import Image
Image(filename='img/Picture34.png')
```

Out[110]:

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 Objects

Pandas indeks nesneleri, eksen etiketlerini ve diğer meta verileri (eksen adı veya adları gibi) tutmaktan sorumludur. Bir Series veya DataFrame oluştururken kullandığınız herhangi bir dizi veya diğer etiket dizisi dahili olarak bir Dizine dönüştürülür.

In [91]:

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

In [92]:

```
index = obj.index
```

In [93]:

```
index
```

Out[93]:

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

In [94]:

```
index[1:]
```

Out[94]:

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

!!Dizin nesneleri değişmezdir ve bu nedenle kullanıcı tarafından değiştirilemez.

In [98]:

```
Image(filename='img/Picture35.png')
```

Out[98]:

```
In [95]: index[1] = 'd' # TypeError
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-95-1a15f4fda8a8> in <module>
----> 1 index[1] = 'd' # TypeError

-/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexes/base.py in __setitem__(self, key, value)
    3908
    3909     def __setitem__(self, key, value):
-> 3910         raise TypeError("Index does not support mutable operations")
    3911
    3912     def __getitem__(self, key):

TypeError: Index does not support mutable operations
```

Değişmezlik, Dizin nesnelerini veri yapıları arasında paylaşmayı daha güvenli hale getirir.

In [99]:

```
labels = pd.Index(np.arange(3))
```

In [100]:

```
labels
```

Out[100]:

```
Int64Index([0, 1, 2], dtype='int64')
```

In [102]:

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

In [103]:

```
obj2
```

Out[103]:

```
0    1.5
1   -2.5
2    0.0
dtype: float64
```

obj2.index is labels

Dizine benzer olmasının yanı sıra, bir Dizin ayrıca sabit boyutlu bir küme gibi davranır.

In [104]:

```
frame3
```

Out[104]:

```
state Nevada Ohio
year
2001    2.4   1.7
2002    2.9   3.6
2000    NaN   1.5
```

In [105]:

```
frame3.columns
```

Out[105]:

```
Index(['Nevada', 'Ohio'], dtype='object', name='state')
```

In [106]:

```
'Ohio' in frame3.columns
```

Out[106]:

```
True
```

In [107]:

```
2003 in frame3.index
```

Out[107]:

```
False
```

Python kümelerinden farklı olarak, bir Pandas Dizini yinelenen etiketler içerebilir.

In [108]:

```
dup_labels = pd.Index(['foo', 'foo', 'bar', 'bar'])
```

In [109]:

```
dup_labels
```

Out[109]:

```
Index(['foo', 'foo', 'bar', 'bar'], dtype='object')
```

Bazı Index yöntemleri ve özellikleri

In [128]:

Image(filename='img/Picture36.png')

Out[128]:

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

2- Temel İşlevsellik

Reindexing

Yeni bir dizine uyan verilerle, yeni bir nesne oluşturmak anlamına gelir.

In [112]:

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

In [113]:

obj

Out[113]:

```
d    4.5
b    7.2
a   -5.3
c    3.6
dtype: float64
```

Bu Seride reindex'in çağrılması, verileri yeni dizine göre yeniden düzenler ve herhangi bir dizin değeri mevcut değilse eksik değerleri ortaya çıkarır.

In [114]:

```
obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
```

In [115]:

```
obj2
```

Out[115]:

```
a    -5.3
b     7.2
c     3.6
d     4.5
e      NaN
dtype: float64
```

Zaman serileri gibi sıralı veriler için, yeniden indeksleme sırasında bazı değerlerin doldurulması istenebilir. method seçeneği, değerleri ileri dolduran ffill gibi bir method kullanarak bunu yapmamıza olanak tanır.

In [116]:

```
obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])
```

In [117]:

```
obj3
```

Out[117]:

```
0    blue
2   purple
4   yellow
dtype: object
```

In [118]:

```
obj3.reindex(range(6), method='ffill')
```

Out[118]:

```
0    blue
1    blue
2   purple
3   purple
4   yellow
5   yellow
dtype: object
```

DataFrame ile reindex, (satır) dizini, sütunları veya her ikisini de değiştirebilir. Yalnızca bir dizi iletildiğinde, sonuçtaki satırları yeniden dizine ekler.

In [119]:

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

In [120]:

frame

Out[120]:

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

In [121]:

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

In [122]:

frame2

Out[122]:

	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

Sütunlar, columns anahtar kelimesi ile yeniden dizine eklenebilir.

In [124]:

```
states = ['Texas', 'Utah', 'California']
```

In [125]:

```
frame.reindex(columns=states)
```

Out[125]:

	Texas	Utah	California
a	1	NaN	2
c	4	NaN	5
d	7	NaN	8

reindex fonksiyonu argumentleri

In [129]:

```
Image(filename='img/Picture37.png')
```

Out[129]:

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.

Dropping Entries from an Axis

Bir eksenden bir veya daha fazla girişi kaldırmak, bu girişler olmadan zaten bir dizin diziniz veya listeniz varsa kolaydır. Bu biraz munging ve ayar manlığı gerektirebileceğinden, drop yöntemi, belirtilen değer veya eksenden silinen değerlere sahip yeni bir nesne döndürür.

In [147]:

```
obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
```

In [148]:

```
obj
```

Out[148]:

```
a    0.0
b    1.0
c    2.0
d    3.0
e    4.0
dtype: float64
```

In [149]:

```
new_obj = obj.drop('c')
```

In [150]:

new_obj

Out[150]:

```
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64
```

In [151]:

obj.drop(['d', 'c'])

Out[151]:

```
a    0.0
b    1.0
e    4.0
dtype: float64
```

DataFrame ile, dizin değerleri her iki eksenden de silinebilir.

In [152]:

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

In [153]:

data

Out[153]:

	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 [154]:

data.drop(['Colorado', 'Ohio'])

Out[154]:

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

axis=1 ya da axis='columns' sütunlardan değerleri kaldırabilirsiniz.

In [155]:

```
data.drop('two', axis=1)
```

Out[155]:

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

In [156]:

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

Out[156]:

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

Birçok fonksiyon bir Series veya DataFrame'in boyutunu veya şeklini değiştir en drop gibi , yeni bir nesne döndürmeden bir nesneyi inplace işleyebilir.

In [157]:

```
obj.drop('c', inplace=True)
```

In [158]:

obj

Out[158]:

```
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64
```

İndeksleme, Seçim ve Filtreleme

Seri indeksleme (obj [...]), NumPy dizi indekslemesine benzer şekilde çalışır, tek farkı, sadece tamsayılar yerine Serinin indeks değerlerini kullanabil meniz.

In [3]:

```
import numpy as np
import pandas as pd
obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
```

In [4]:

```
obj
```

Out[4]:

```
a    0.0
b    1.0
c    2.0
d    3.0
dtype: float64
```

In [5]:

```
obj['b']
```

Out[5]:

```
1.0
```

In [6]:

```
obj[1]
```

Out[6]:

```
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[[1, 3]]
```

Out[9]:

```
b    1.0  
d    3.0  
dtype: float64
```

In [10]:

```
obj[obj < 2]
```

Out[10]:

```
a    0.0  
b    1.0  
dtype: float64
```

Etiketlerle dilimleme, normal Python dilimlemesinden farklı davranışır. Nokta kapsayıcıdır.

In [11]:

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

Out[11]:

```
b    1.0  
c    2.0  
dtype: float64
```

Bu yöntemleri kullanarak, Serinin karşılık gelen bölümünü değiştirebilirsiniz.

In [12]:

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

Bir DataFrame'e indeksleme, tek bir değer veya sıra ile bir veya daha fazla sütunu alabilirsiniz.

In [13]:

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

In [14]:

```
data
```

Out[14]:

	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 [15]:

```
data['two']
```

Out[15]:

```
Ohio      1
Colorado  5
Utah      9
New York  13
Name: two, dtype: int64
```

In [16]:

```
data[['three', 'one']]
```

Out[16]:

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

In [17]:

```
data[:2]
```

Out[17]:

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

In [18]:

```
data[data['three'] > 5]
```

Out[18]:

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

[] işaretini tek bir öğe veya bir liste sütunları seçer.

In [19]:

```
data < 5
```

Out[19]:

	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 [20]:

```
data[data < 5] = 0
```

In [21]:

```
data
```

Out[21]:

	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

1- Loc ve iloc ile seçim

Eksen etiketleri (loc) veya tamsayılar (iloc) kullanarak NumPy benzeri gösterime sahip bir DataFrame'den satırların ve sütunların bir alt kümesini seçmenizi sağlar.

In [22]:

```
data.loc['Colorado', ['two', 'three']]
```

Out[22]:

```
two      5
three    6
Name: Colorado, dtype: int64
```

In [23]:

```
data.iloc[2, [3, 0, 1]]
```

Out[23]:

```
four    11
one     8
two     9
Name: Utah, dtype: int64
```

In [24]:

```
data.iloc[2]
```

Out[24]:

```
one     8
two     9
three   10
four    11
Name: Utah, dtype: int64
```

In [26]:

```
data.iloc[[1, 2], [3, 0, 1]]
```

Out[26]:

	four	one	two
Colorado	7	0	5
Utah	11	8	9

Her iki dizinleme işlevi de tekli etiketlere veya etiket listelerine ek olarak dilimlerle çalışır.

In [27]:

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

Out[27]:

```
Ohio      0
Colorado  5
Utah     9
Name: two, dtype: int64
```

In [28]:

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

Out[28]:

	one	two	three
Colorado	0	5	6
Utah	8	9	10
New York	12	13	14

DataFrame ile indeksleme seçenekleri

In [29]:

```
from IPython.display import Image
Image(filename='img/Picture38.png')
```

Out[29]:

Type	Notes
df[val]	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)
df.loc[val]	Selects single row or subset of rows from the DataFrame by label
df.loc[:, val]	Selects single column or subset of columns by label
df.loc[val1, val2]	Select both rows and columns by label
df.iloc[where]	Selects single row or subset of rows from the DataFrame by integer position
df.iloc[:, where]	Selects single column or subset of columns by integer position
df.iloc[where_i, where_j]	Select both rows and columns by integer position
df.at[label_i, label_j]	Select a single scalar value by row and column label
df.iat[i, j]	Select a single scalar value by row and column position (integers)
reindex method	Select either rows or columns by labels
get_value, set_value methods	Select single value by row and column label

Integer İndekleme

Tamsayılar tarafından indekslenmiş pandas nesneleriyle çalışmak, listeler ve tuples gibi yerleşik Python veri yapılarında indeksleme anlambilimindeki bazı farklılıklar nedeniyle genellikle yeni kullanıcıları tedirgin eden bir şeydir. Örneğin, aşağıdaki kodun bir hata oluşturmasını beklemeyebilirsiniz.

In [35]:

```
ser = pd.Series(np.arange(3.))
```

Bu durumda, pandalar tamsayı indekslemede "geri dönebilir", ancak bunu genel

olarak ince hatalar olmadan yapmak zordur. Burada 0, 1, 2 içeren bir dizine sahibiz, ancak kullanıcının ne istediğini anlamak (etiket tabanlı dizin oluşturma veya konum tabanlı) zordur.

In [36]:

```
ser
```

Out[36]:

```
0    0.0
1    1.0
2    2.0
dtype: float64
```

Tamsayı olmayan bir indekste, belirsizlik potansiyeli yoktur.

In [37]:

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

In [38]:

```
ser2[-1]
```

Out[38]:

```
2.0
```

Her şeyi tutarlı tutmak için, tamsayılar içeren bir eksen dizininiz varsa, veri seçimi her zaman etikete yönelik olacaktır. Daha hassas işlem için loc (etiketler için) veya iloc (tamsayılar için) kullanın.

In [39]:

```
ser[:1]
```

Out[39]:

```
0    0.0
dtype: float64
```

In [40]:

```
ser.loc[:1]
```

Out[40]:

```
0    0.0
1    1.0
dtype: float64
```

In [41]:

```
ser.iloc[:1]
```

Out[41]:

```
0    0.0
dtype: float64
```

Aritmetik ve Veri Hizalama

In [42]:

```
s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])
```

In [43]:

```
s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],
                index=['a', 'c', 'e', 'f', 'g'])
```

In [44]:

```
s1
```

Out[44]:

```
a    7.3
c   -2.5
d    3.4
e    1.5
dtype: float64
```

In [45]:

```
s2
```

Out[45]:

```
a   -2.1
c    3.6
e   -1.5
f    4.0
g    3.1
dtype: float64
```

In [46]:

```
s1 + s2
```

Out[46]:

```
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

Veri hizalaması, etiket konumlarında çakışmayan eksik değerleri ortaya çıkarır. Eksik değerler daha sonra başka aritmetik hesaplamalarda yayılır.

DataFrame durumunda, hizalama hem satırlarda hem de sütunlarda gerçekleştirilir.

In [51]:

```
df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)), columns=list('bcd'),
                   index=['Ohio', 'Texas', 'Colorado'])
```

In [52]:

```
df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)), columns=list('bde'),
                   index=['Utah', 'Ohio', 'Texas', 'Oregon'])
```

In [53]:

```
df1
```

Out[53]:

	b	c	d
Ohio	0.0	1.0	2.0
Texas	3.0	4.0	5.0
Colorado	6.0	7.0	8.0

In [54]:

```
df2
```

Out[54]:

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

Bunların birbirine eklenmesi, dizini ve sütunları her bir DataFrame'dekileri n birleşimleri olan bir DataFrame döndürür.

In [55]:

df1 + df2

Out[55]:

	b	c	d	e
Colorado	NaN	NaN	NaN	NaN
Ohio	3.0	NaN	6.0	NaN
Oregon	NaN	NaN	NaN	NaN
Texas	9.0	NaN	12.0	NaN
Utah	NaN	NaN	NaN	NaN

Her iki DataFrame nesnesinde de 'c' ve 'e' sütunları bulunmadığından, sonuçta tümü eksik olarak görünür. Aynı durum, etiketleri her iki nesne için ortak olmayan satırlar için de geçerlidir.

Ortak sütun veya satır etiketi olmayan DataFrame nesneleri eklerseniz, sonuçta tüm boş değerleri içerecektir.

In [56]:

df1 = pd.DataFrame({'A': [1, 2]})

In [57]:

df2 = pd.DataFrame({'B': [3, 4]})

In [58]:

df1

Out[58]:

A
0 1
1 2

In [59]:

df2

Out[59]:

B
0 3
1 4

In [60]:

df1 = df2

Out[60]:

	A	B
0	NaN	NaN
1	NaN	NaN

1- Aritmatik metodlar ile fill values

Farklı dizine alınmış nesneler arasındaki aritmetik işlemlerde, bir nesnede eksen etiketi bulunduğuanda ancak diğerinde bulunmadığında 0 gibi özel bir değerle doldurmak isteyebilirsiniz.

In [62]:

df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),
columns=list('abcd'))

In [63]:

df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),
columns=list('abcde'))

In [64]:

df2.loc[1, 'b'] = np.nan

In [65]:

df1

Out[65]:

	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 [66]:

```
df2
```

Out[66]:

	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

Bunların birbirine eklenmesi, çakışmayan konumlarda NA değerlerine neden olur.

In [67]:

```
df1 + df2
```

Out[67]:

	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 [68]:

```
df1.add(df2, fill_value=0)
```

Out[68]:

	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 [69]:

1 / df1

Out[69]:

	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 [70]:

df1.rdiv(1)

Out[70]:

	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

Benzer şekilde, bir Seriyi veya DataFrame'i yeniden indekslerken, farklı bir fill değeri de belirtebilirsiniz.

In [71]:

df1.reindex(columns=df2.columns, fill_value=0)

Out[71]:

	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

Flexible aritmetik yöntemler

```

add, radd : Ekleme yöntemleri (+)
sub, rsub : Çıkarma yöntemleri (-)
div, rdiv : Bölme yöntemleri (/)
floordiv, : floor bölme yöntemleri (//)
mul, rmul : Çarpma yöntemleri (*)
pow, rpow : Üs alma yöntemleri (**)

```

2- DataFrame ve Seriler arasındaki işlemler

Farklı boyutlardaki NumPy dizilerinde olduğu gibi, DataFrame ve Series a

rasındaki aritmetik de tanımlanmıştır.

In [74]:

```
arr = np.arange(12.).reshape((3, 4))
```

```
arr
```

In [75]:

```
arr[0]
```

Out[75]:

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

In [76]:

```
arr - arr[0]
```

Out[76]:

```
array([[0., 0., 0., 0.],
       [4., 4., 4., 4.],
       [8., 8., 8., 8.]])
```

Arr [0] 'ı arr'den çıkardığımızda, çıkarma her satır için bir kez yapılır.

In [77]:

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

In [78]:

```
series = frame.iloc[0]
```

In [79]:

```
frame
```

Out[79]:

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

In [80]:

series

Out[80]:

```
b    0.0
d    1.0
e    2.0
Name: Utah, dtype: float64
```

Varsayılan olarak, DataFrame ve Series arasındaki aritmetik, DataFrame'in sütunlarındaki Seri diziniyle eşleşerek satırları oluşturur.

In [81]:

frame - series

Out[81]:

	b	d	e
Utah	0.0	0.0	0.0
Ohio	3.0	3.0	3.0
Texas	6.0	6.0	6.0
Oregon	9.0	9.0	9.0

DataFrame'in sütunlarında veya Serinin dizininde bir dizin değeri bulunmazsa, nesneler birleşmeyi oluşturmak için yeniden dizine alınacaktır.

In [82]:

series2 = pd.Series(range(3), index=['b', 'e', 'f'])

In [83]:

frame + series2

Out[83]:

	b	d	e	f
Utah	0.0	NaN	3.0	NaN
Ohio	3.0	NaN	6.0	NaN
Texas	6.0	NaN	9.0	NaN
Oregon	9.0	NaN	12.0	NaN

Bunun yerine sütunlar üzerinden yayın yapmak, satırları eşleştirmek istiyorsanız, aritmetik yöntemlerden birini kullanmanız gereklidir. Örneğin:

In [84]:

```
series3 = frame['d']
```

In [85]:

```
frame
```

Out[85]:

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

In [86]:

```
series3
```

Out[86]:

```
Utah      1.0
Ohio      4.0
Texas     7.0
Oregon    10.0
Name: d, dtype: float64
```

In [87]:

```
frame.sub(series3, axis='index')
```

Out[87]:

	b	d	e
Utah	-1.0	0.0	1.0
Ohio	-1.0	0.0	1.0
Texas	-1.0	0.0	1.0
Oregon	-1.0	0.0	1.0

Geçtiğiniz eksen numarası, eşleşecek eksendir. DataFrame'in satır dizininde
(eksen = 'dizin' veya eksen = 0)

Fonksiyon Uygulaması ve Haritalama

NumPy ufuncs (element-wise dizi yöntemleri) ayrıca pandas nesneleriyle de çalışır.

In [88]:

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

In [89]:

frame

Out[89]:

	b	d	e
Utah	0.686819	0.262094	1.350754
Ohio	0.532064	-0.222477	-1.928672
Texas	-0.085077	0.215716	0.493001
Oregon	0.633262	0.284283	1.750000

In [90]:

np.abs(frame)

Out[90]:

	b	d	e
Utah	0.686819	0.262094	1.350754
Ohio	0.532064	0.222477	1.928672
Texas	0.085077	0.215716	0.493001
Oregon	0.633262	0.284283	1.750000

Diğer bir sık yapılan işlem, tek boyutlu dizilerde her bir sütun veya satırda bir işlem uygulamaktır.

In [91]:

f = lambda x: x.max() - x.min()

In [92]:

frame.apply(f)

Out[92]:

```
b    0.771896
d    0.506760
e    3.678672
dtype: float64
```

Burada bir Serinin maksimum ve minimumları arasındaki farkı hesaplayan f fonksiyonu, çerçevedeki her sütunda bir kez çağrılır. Sonuç, indeksi olarak çerçevedeki sütunlarını içeren bir Seridir.

axis = 'column' değerini yazarsanız, fonksiyon bunun yerine satır başına bir kez çağrılacaktır.

In [93]:

```
frame.apply(f, axis='columns')
```

Out[93]:

```
Utah      1.088660
Ohio      2.460736
Texas     0.578079
Oregon    1.465717
dtype: float64
```

Birden çok değer içeren bir Seri de döndürebilir.

In [94]:

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

In [95]:

```
frame.apply(f)
```

Out[95]:

	b	d	e
min	-0.085077	-0.222477	-1.928672
max	0.686819	0.284283	1.750000

In [96]:

```
format = lambda x: '%.2f' % x
```

In [97]:

```
frame.applymap(format)
```

Out[97]:

	b	d	e
Utah	0.69	0.26	1.35
Ohio	0.53	-0.22	-1.93
Texas	-0.09	0.22	0.49
Oregon	0.63	0.28	1.75

In [98]:

```
frame['e'].map(format)
```

Out[98]:

```
Utah      1.35
Ohio     -1.93
Texas      0.49
Oregon     1.75
Name: e, dtype: object
```

Sorting ve Ranking

Satır veya sütun indeksine göre sıralamak için, yeni, sıralı bir nesne döndüren `sort_index` yöntemini kullanın.

In [99]:

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

In [100]:

```
obj.sort_index()
```

Out[100]:

```
a    1
b    2
c    3
d    0
dtype: int64
```

`DataFrame` ile her iki eksende dizine göre sıralama yapabilirsiniz.

In [101]:

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

In [102]:

```
frame.sort_index()
```

Out[102]:

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

In [103]:

```
frame.sort_index(axis=1)
```

Out[103]:

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

Veriler varsayılan olarak artan düzende sıralanır, ancak azalan düzende de sıralanabilir.

In [104]:

```
frame.sort_index(axis=1, ascending=False)
```

Out[104]:

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

Bir Seriyi değerlerine göre sıralamak için sort_values yöntemini kullanın.

In [105]:

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

In [107]:

```
obj.sort_values()
```

Out[107]:

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

Eksik değerler, varsayılan olarak Serinin sonunda sıralanır.

In [108]:

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

In [110]:

```
obj.sort_values()
```

Out[110]:

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

Bir DataFrame'i sıralarken, bir veya daha fazla sütundaki verileri sort anah tarları olarak kullanabilirsiniz. Bunu yapmak için, sort_values seçeneğiyle bir veya daha fazla sütun adı seçin.

In [111]:

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

In [112]:

```
frame
```

Out[112]:

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

In [113]:

```
frame.sort_values(by='b')
```

Out[113]:

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

Birden çok sütuna göre sıralamak için bir ad listesi yazın.

In [114]:

```
frame.sort_values(by=['a', 'b'])
```

Out[114]:

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

Ranking, bir dizideki geçerli veri noktalarının sayısı kadar bir ile sıralamaya atar. Seriler ve DataFrame için rank methods bakılacak yerdir; varsayılan olarak sıralama, her gruba ortalama sıralamayı atayarak bağları bozar.

In [115]:

```
obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
```

In [116]:

```
obj.rank()
```

Out[116]:

0	6.5
1	1.0
2	6.5
3	4.5
4	3.0
5	2.0
6	4.5

dtype: float64

Sıralar, verilerde gözlemlendikleri sıraya göre de atanabilir.

In [117]:

```
obj.rank(method='first')
```

Out[117]:

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

Burada, 0 ve 2 girişleri için ortalama 6.5 sıralaması kullanmak yerine, bunun yerine 6 ve 7 olarak ayarlanmıştır çünkü etiket 0, verilerde etiket 2'den önce gelir.

In [118]:

```
obj.rank(ascending=False, method='max')
```

Out[118]:

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

DataFrame, satırlar veya sütunlar üzerindeki dereceleri hesaplayabilir.

In [119]:

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

In [120]:

```
frame
```

Out[120]:

	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 [121]:

```
frame.rank(axis='columns')
```

Out[121]:

	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

Rank ile Tie-breaking metod

In [122]:

```
from IPython.display import Image
Image(filename='img/Picture39.png')
```

Out[122]:

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 method='min', but ranks always increase by 1 in between groups rather than the number of equal elements in a group

Duplicate Labels ile Axis İndeksleme

In [123]:

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

In [124]:

```
obj
```

Out[124]:

```
a    0
a    1
b    2
b    3
c    4
dtype: int64
```

Dizinin `is_unique` özelliği, etiketlerinin benzersiz olup olmadığını size söyleyebilir.

```
In [125]:
```

```
obj.index.is_unique
```

```
Out[125]:
```

```
False
```

Veri seçimi, kopyalarda farklı davranışın ana şeylerden biridir. Birden çok girişî olan bir etiketin dizine alınması bir Seri döndürürken tekli girişler bir skaler değer döndürür.

```
In [126]:
```

```
obj['a']
```

```
Out[126]:
```

```
a    0  
a    1  
dtype: int64
```

```
In [127]:
```

```
obj['c']
```

```
Out[127]:
```

```
4
```

Aynı mantık, bir DataFrame'deki satırları indekslemeye de geçerli.

```
In [128]:
```

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

```
In [129]:
```

```
df
```

```
Out[129]:
```

	0	1	2
a	-0.722238	0.669238	-1.023335
a	-0.425140	-1.240469	0.009322
b	0.768060	2.537518	0.743817
b	0.902909	-0.091148	0.907495

5- Tanımlayıcı İstatistikleri Öztleme ve Hesaplama

Pandas nesneleri bir dizi yaygın matematiksel ve istatistiksel yöntemle donatılmıştır. Bunların çoğu, bir DataFrame'in satırlarından veya sütunlarından bir Seriden veya bir Değer Dizisinden tek bir değer (toplam veya ortalama gibi) çıkan yöntemler olan indirimler veya özet istatistikler kategorisine girer. NumPy dizilerinde bulunan benzer yöntemlerle karşılaşıldığında, eksik veriler için yerlesik işleme sahiptirler.

In [130]:

```
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 [131]:

```
df
```

Out[131]:

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

DataFrame'in sum yöntemini çağırırmak, sütun toplamlarını içeren bir Seri döndürür.

In [132]:

```
df.sum()
```

Out[132]:

```
one    9.25
two   -5.80
dtype: float64
```

In [133]:

```
df.sum(axis='columns')
```

Out[133]:

```
a    1.40
b    2.60
c    0.00
d   -0.55
dtype: float64
```

In [134]:

```
df.mean(axis='columns', skipna=False)
```

Out[134]:

```
a      NaN
b     1.300
c      NaN
d    -0.275
dtype: float64
```

Azaltma yöntemleri için seçenekler

axis: Eksenin aşırı azaltmak için; DataFrame'in satırları için 0 ve sütunlar için 1

skipna: Eksik değerleri hariç tut; True by varsayılan

level: Eksen hiyerarşik olarak indekslenmişse seviyeye göre gruplandırılmış olarak azaltın (MultiIndex)

idxmin ve **idxmax** gibi bazı yöntemler, minimum veya maksimum değerlerin elde edildiği dizin değeri gibi dolaylı istatistikleri döndürür.

In [135]:

```
df.idxmax()
```

Out[135]:

```
one    b
two    d
dtype: object
```

In [136]:

```
df.cumsum()
```

Out[136]:

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

In [137]:

```
df.describe()
```

Out[137]:

	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

Sayısal olmayan veriler üzerinde, describe alternatif özet istatistikleri üretir.

In [138]:

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

In [139]:

```
obj.describe()
```

Out[139]:

count	16
unique	3
top	a
freq	8
dtype:	object

Tanımlayıcı ve özet istatistikler

In [141]:

```
Image( filename='img/Picture40.png' )
```

Out[141]:

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

Korelasyon ve kovaryans gibi bazı özet istatistikler, argüman çiftlerinden hesaplanır.

In [143]:

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

In [144]:

```
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 [145]:

```
returns = price.pct_change()
```

In [146]:

```
returns.tail()
```

Out[146]:

	AAPL	IBM	MSFT	GOOG
Date				
2020-11-02	-0.000827	0.011195	-0.000691	0.003097
2020-11-03	0.015354	0.011071	0.020264	0.014871
2020-11-04	0.040837	-0.019797	0.048249	0.059944
2020-11-05	0.035494	0.025648	0.031887	0.008141
2020-11-06	-0.001136	-0.006361	0.001926	-0.000919

Series'in corr yöntemi, iki Serideki örtüsen, NA olmayan, indeksle hizalanmış değerlerin korelasyonunu hesaplar. Benzer şekilde, cov kovaryansı hesaplar.

In [147]:

```
returns['MSFT'].corr(returns['IBM'])
```

Out[147]:

```
0.569485783390246
```

In [148]:

```
returns['MSFT'].cov(returns['IBM'])
```

Out[148]:

```
0.00016119609098997384
```

MSFT geçerli bir Python niteliği olduğundan, bu sütunları daha kısa sözdizimi kullanarak da seçebiliriz.

In [149]:

```
returns.MSFT.corr(returns.IBM)
```

Out[149]:

```
0.569485783390246
```

DataFrame'in corr ve cov yöntemleri, sırasıyla bir DataFrame olarak tam bir korelasyon veya kovaryans matrisi döndürür.

In [150]:

```
returns.corr()
```

Out[150]:

	AAPL	IBM	MSFT	GOOG
AAPL	1.000000	0.483570	0.719072	0.656737
IBM	0.483570	1.000000	0.569486	0.525197
MSFT	0.719072	0.569486	1.000000	0.780992
GOOG	0.656737	0.525197	0.780992	1.000000

In [151]:

```
returns.cov()
```

Out[151]:

	AAPL	IBM	MSFT	GOOG
AAPL	0.000361	0.000149	0.000239	0.000208
IBM	0.000149	0.000262	0.000161	0.000142
MSFT	0.000239	0.000161	0.000306	0.000228
GOOG	0.000208	0.000142	0.000228	0.000278

DataFrame'in corrwith yöntemini kullanarak, bir DataFrame'in sütunları veya satırları ile başka bir Series veya DataFrame arasındaki ikili korelasyonları hesaplayabilirsiniz. Bir Seriyi geçmek, her sütun için hesaplanan korelasyon değerine sahip bir Seri döndürür.

In [152]:

```
returns.corrwith(returns.IBM)
```

Out[152]:

```
AAPL      0.483570
IBM      1.000000
MSFT      0.569486
GOOG      0.525197
dtype: float64
```

In [153]:

```
returns.corrwith(volume)
```

Out[153]:

```
AAPL     -0.096466
IBM     -0.080754
MSFT     -0.095916
GOOG     -0.162652
dtype: float64
```

Unique Values, Value Counts, ve Membership

İlgili yöntemlerin başka bir sınıfı, tek boyutlu bir Seride bulunan değerler hakkındaki bilgileri ayıklar.

In [154]:

```
obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])
```

In [155]:

```
uniques = obj.unique()
```

In [156]:

```
uniques
```

Out[156]:

```
array(['c', 'a', 'd', 'b'], dtype=object)
```

In [157]:

```
obj.value_counts()
```

Out[157]:

```
c    3
a    3
b    2
d    1
dtype: int64
```

Seri, kolaylık sağlamak için azalan düzende değere göre sıralanmıştır. value

`_counts`, herhangi bir dizi veya diziyile kullanılabilen pandas yöntemidir.

In [158]:

```
pd.value_counts(obj.values, sort=False)
```

Out[158]:

```
a    3  
b    2  
d    1  
c    3  
dtype: int64
```

`isin`, vektörleştirilmiş bir membership denetimi gerçekleştirir ve bir veri kümesini DataFrame'deki bir Seri veya sütundaki değerlerin bir alt kümesine farklı trelemede yararlı olabilir.

In [159]:

```
obj
```

Out[159]:

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

In [160]:

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

In [161]:

```
mask
```

Out[161]:

```
0    True  
1   False  
2   False  
3   False  
4   False  
5    True  
6    True  
7    True  
8    True  
dtype: bool
```

In [162]:

obj[mask]

Out[162]:

```
0    c
5    b
6    b
7    c
8    c
dtype: object
```

`Index.get_indexer` yöntemi, farklı olmayan değerler dizisinden başka bir farklı değerler dizisine bir dizin dizisi verir.

In [163]:

to_match = pd.Series(['c', 'a', 'b', 'b', 'c', 'a'])

In [164]:

unique_vals = pd.Series(['c', 'b', 'a'])

In [165]:

pd.Index(unique_vals).get_indexer(to_match)

Out[165]:

array([0, 2, 1, 1, 0, 2])

Unique, value counts, ve set membership metodları

In [167]:

Image(filename='img/Picture41.png')

Out[167]:

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

Bazı durumlarda, bir DataFrame'deki birden çok ilgili sütunda histogram hesaplamak isteyebilirsiniz.

In [168]:

```
data = pd.DataFrame({'Qu1': [1, 3, 4, 3, 4],  
                     'Qu2': [2, 3, 1, 2, 3],  
                     'Qu3': [1, 5, 2, 4, 4]})
```

In [169]:

```
data
```

Out[169]:

	Qu1	Qu2	Qu3
0	1	2	1
1	3	3	5
2	4	1	2
3	3	2	4
4	4	3	4

In [170]:

```
result = data.apply(pd.value_counts).fillna(0)
```

In [171]:

```
result
```

Out[171]:

	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

Burada, sonuçtaki satır etiketleri, tüm sütunlarda ortaya çıkan farklı değerlerdir. Değerler, her bir sütundaki bu değerlerin ilgili sayılarıdır.

In []:

Veri Yükleme, Depolama ve Dosya Biçimleri

6.1 Metin Biçiminde Veri Okuma ve Yazma

pandas, tablo şeklindeki verileri bir DataFrame nesnesi olarak okumak için bir dizi fonksiyon içerir.

In [1]:

```
from IPython.display import Image
Image("img/picture42.png")
```

Out[1]:

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

Metin verilerini bir DataFrame'e dönüştürmeyi amaçlayan bu fonksiyonlar, isteğe bağlı argümanlar birkaç kategoriye ayrılabilir.

1)Endeksleme Bir veya daha fazla sütunu döndürülen DataFrame olarak ve dosyadan, kullanıcının sütun adlarının alınıp alınmayacağıni veya hiç almayacağını kabul edebilir.

2)Tür çıkarımı ve veri dönüştürme Bu, kullanıcı tanımlı değer dönüşümlerini ve eksik değer işaretçilerinin özel listesini içerir.

3)Tarih saat ayırtırma Birden çok sütuna yayılmış tarih ve saat bilgilerinin sonuçtaki tek bir sütunda birleştirilmesi dahil olmak üzere birleştirme özelliğini içerir.

4)Yinelemek Çok büyük dosyaların yiğinlarını yineleme desteği.

5) Temiz olmayan veri sorunları Binlerce virgülle ayrılmış sayısal veriler gibi satırları veya altbilgiyi, yorumları veya diğer küçük şeyleri atlama.

Gerçek dünyadaki verilerin dağınık olabileceğinden dolayı, veri yükleme fonksiyonlarından bazıları (özellikle `read_csv`) zaman içinde çok karmaşık hale geldi.

`Pandas.read_csv` gibi bu işlevlerden bazıları tür çıkarımı gerçekleştirir, çünkü sütun veri türleri veri biçiminin bir parçası değildir. Bu, hangi sütunların sayısal, tam sayı, boolen veya dize olduğunu belirtmeniz gerekmeli anlamına gelir. HDF5, Feather ve msgpack gibi diğer veri formatları, formatta depolanan veri türlerine sahiptir.

Tarihleri ve diğer özel türleri işlemek ekstra çaba gerektirebilir. Virgülle ayrılmış (CSV) metin dosyasını ele alalım.

In [2]:

```
!cat ch06/ex1.csv
```

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

Dosya virgülle ayrılmış olduğundan, bunu bir `DataFrame`'e okumak için `read_csv`'yi kullanabiliriz.

In [3]:

```
import pandas as pd
df = pd.read_csv('ch06/ex1.csv')
```

In [4]:

```
df
```

Out[4]:

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

Ayrıca `read_table` kullanabilir ve sınırlayıcıyı belirleyebilirsiniz.

In [5]:

```
pd.read_table('ch06/ex1.csv', sep=',')
```

Out[5]:

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

Bir dosyanın her zaman bir başlık satırı olmayacağıdır. Bu dosyayı düşünün:

In [6]:

```
!cat ch06/ex2.csv
```

```
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

Bu dosyayı okumak için birkaç seçenek vardır. Pandas varsayılan sütun adlarını atamasına izin verebilir veya adları kendiniz belirtebilirsiniz.

In [7]:

```
pd.read_csv('ch06/ex2.csv', header=None)
```

Out[7]:

	0	1	2	3	4
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

In [8]:

```
pd.read_csv('ch06/ex2.csv', names=['a', 'b', 'c', 'd', 'message'])
```

Out[8]:

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

İleti sütununun döndürülen DataFrame'in dizini olmasını istediğiniz varsayılm. Sütunu dizin 4'te istediğinizizi belirtebilir veya dizin_kol değişkenini kullanarak 'iletı' olarak adlandırabilirsiniz.

In [9]:

```
names = ['a', 'b', 'c', 'd', 'message']
```

In [10]:

```
pd.read_csv('ch06/ex2.csv', names=names, index_col='message')
```

Out[10]:

	a	b	c	d
--	---	---	---	---

message				
---------	--	--	--	--

hello	1	2	3	4
-------	---	---	---	---

world	5	6	7	8
-------	---	---	---	---

foo	9	10	11	12
-----	---	----	----	----

Birden çok sütündan hiyerarşik bir dizin oluşturmak istemeniz durumunda, süt un numaraları veya adların bir listesini iletin.

In [11]:

```
!cat ch06/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
```

In [12]:

```
parsed = pd.read_csv('ch06/csv_mindex.csv',
                      index_col=['key1', 'key2'])
```

In [13]:

parsed

Out[13]:

value1 value2

key1 key2

	a	1	2
one	b	3	4
	c	5	6
	d	7	8
	a	9	10
two	b	11	12
	c	13	14
	d	15	16

Bazı durumlarda, bir tablonun alanları ayırmak için beyaz boşluk veya başka bir kalıp kullanan sabit bir sınırlayıcısı olmayabilir. Şuna benzeyen bir metin dosyası düşünün:

In [14]:

list(open('ch06/ex3.txt'))

Out[14]:

```
[ '           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\n']
```

El ile biraz kesme yapabilirsiniz, ancak buradaki alanlar değişken miktarda boşlukla ayrılır. Bu durumlarda, `read_table` için bir ayırıcı olarak normal bir ifade iletебilirsiniz. Bu, `\s+` düzenli ifadesi ile ifade edilebilir, dolayısıyla bizde:

In [15]:

result = pd.read_table('ch06/ex3.txt', sep='\s+')

In [16]:

result

Out[16]:

	A	B	C
aaa	-0.264438	-1.026059	-0.619500
bbb	0.927272	0.302904	-0.032399
ccc	-0.264273	-0.386314	-0.217601
ddd	-0.871858	-0.348382	1.100491

Veri satırlarının sayısından bir daha az sütun adı olduğundan, `read_table` bu özel durumda ilk sütunun `DataFrame`'in indeksi olması gereği sonucuna varır.

Bir dosyanın birinci, üçüncü ve dördüncü satırlarını `skiprows` ile atlayabilirsiniz.

In [17]:

!cat ch06/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 [18]:

pd.read_csv('ch06/ex4.csv', skiprows=[0, 2, 3])

Out[18]:

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

Eksik değerlerin ele alınması, dosya ayrıştırma işleminin önemli ve sıkılıkla nüanslı bir parçasıdır. Eksik veriler genellikle mevcut değildir (boş dizedir) veya bazı gözlemci değerlerle işaretlenir. Pandas, varsayılan olarak NA ve NULL gibi yaygın olarak ortaya çıkan bir dizi nöbetçi kullanır.

In [19]:

```
!cat ch06/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 [20]:

```
result = pd.read_csv('ch06/ex5.csv')
```

In [21]:

```
result
```

Out[21]:

	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 [22]:

```
pd.isnull(result)
```

Out[22]:

	something	a	b	c	d	message
0	False	False	False	False	False	True
1	False	False	False	True	False	False
2	False	False	False	False	False	False

Na_values seçeneği, eksik değerleri göz önünde bulundurmak için bir liste ve ya dize kümesini alabilir.

In [23]:

```
result = pd.read_csv('ch06/ex5.csv', na_values=['NULL'])
```

In [24]:

```
result
```

Out[24]:

	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

Bir diktodedeki her sütun için farklı NA nöbetçileri belirlenebilir.

In [25]:

```
sentinels = {'message': ['foo', 'NA'], 'something': ['two']}
```

In [26]:

```
pd.read_csv('ch06/ex5.csv', na_values=sentinels)
```

Out[26]:

	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

In [27]:

Image("img/picture43.png")

Out[27]:

Table 6-2. Some `read_csv/read_table` function arguments

Argument	Description
<code>path</code>	String indicating filesystem location, URL, or file-like object
<code>sep or delimiter</code>	Character sequence or regular expression to use to split fields in each row
<code>header</code>	Row number to use as column names; defaults to 0 (first row), but should be <code>None</code> if there is no header row
<code>index_col</code>	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
<code>names</code>	List of column names for result, combine with <code>header=None</code>
<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 'foo' 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> , <code>'</code> or <code>.</code>).

1) Metin Dosyalarını Parçalar Halinde Okuma

Çok büyük dosyaları işlerken veya büyük bir dosyayı doğru bir şekilde işlemek için doğru argümanlar setini bulurken, dosyanın yalnızca küçük bir parçasını okumak veya dosyanın daha küçük parçalarını yinelemek isteyebilirsiniz.

Büyük bir dosyaya bakmadan önce, pandas görüntü ayarlarını daha kompakt hale getiriyoruz.

In [28]:

```
pd.options.display.max_rows = 10
```

In [29]:

```
result = pd.read_csv('ch06/ex6.csv')
```

In [30]:

```
result
```

Out[30]:

	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
...
9995	2.311896	-0.417070	-1.409599	-0.515821	L
9996	-0.479893	-0.650419	0.745152	-0.646038	E
9997	0.523331	0.787112	0.486066	1.093156	K
9998	-0.362559	0.598894	-1.843201	0.887292	G
9999	-0.096376	-1.012999	-0.657431	-0.573315	0

10000 rows × 5 columns

Yalnızca az sayıda satırı okumak istiyorsanız (tüm dosyayı okumaktan kaçınmak), bunu sayılarla belirtin.

In [31]:

```
pd.read_csv('ch06/ex6.csv', nrows=5)
```

Out[31]:

	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

Bir dosyayı parçalar halinde okumak için, bir yığın boyutunu birkaç satır olarak belirtin.

In [32]:

```
chunker = pd.read_csv('ch06/ex6.csv', chunksize=1000)
```

In [33]:

```
chunker
```

Out[33]:

```
<pandas.io.parsers.TextFileReader at 0x7ff379f775b0>
```

Read_csv tarafından döndürülen TextParser nesnesi, yığın boyutuna göre dosyaının bölümleri üzerinde yineleme yapmanıza olanak tanır. Örneğin, ex6.csv üzerinde yineleme yapabiliriz, 'anahtar' sütunundaki değer sayılarını şu şekilde toplayabiliriz:

In [34]:

```
chunker = pd.read_csv('ch06/ex6.csv', chunksize=1000)
```

```
tot = pd.Series([], dtype=pd.StringDtype())
for piece in chunker:
    tot = tot.add(piece['key'].value_counts(), fill_value=0)
tot = tot.sort_values(ascending=False)
```

In [37]:

tot[:10]

Out[37]:

Series([], dtype: string)

TextParser ayrıca rastgele boyuttaki parçaları okumanızı sağlayan bir get_chunk yöntemiyle donatılmıştır.

2) Metin Formatına Veri Yazma

Veriler ayrıca sınırlandırılmış bir biçimde dışa aktarılabilir. Daha önce okunan CSV dosyalarından birini ele alalım.

In [38]:

data = pd.read_csv('ch06/ex5.csv')

In [39]:

data

Out[39]:

	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

DataFrame'in to_csv yöntemini kullanarak verileri virgülle ayrılmış bir dosyaaya yazabilirisiz.

In [40]:

data.to_csv('ch06/out.csv')

In [41]:

!cat ch06/out.csv

```
,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
```

Elbette diğer sınırlayıcılar da kullanılabilir (sys.stdout'a yazarak metin s onucunu konsola yazdırır).

In [42]:

```
import sys
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
```

Eksik değerler, çıktıda boş dizeler olarak görünür. Onları başka bir koruyucu değerle belirtmek isteyebilirsiniz.

In [43]:

```
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
```

Başka seçenek belirtilmeden hem satır hem de sütun etiketleri yazılır. Burada ikisi de devre dışı bırakılabilir.

In [44]:

```
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
```

Ayrıca, sütunların yalnızca bir alt kümesini seçtiğiniz bir sırada yazabilirsiniz.

In [45]:

```
data.to_csv(sys.stdout, index=False, columns=['a', 'b', 'c'])
```

```
a,b,c
1,2,3.0
5,6,
9,10,11.0
```

Serinin ayrıca bir to_csv yöntemi vardır.

In [46]:

```
dates = pd.date_range('1/1/2000', periods=7)
```

In [47]:

```
import numpy as np
ts = pd.Series(np.arange(7), index=dates)
```

In [48]:

```
ts.to_csv('ch06/tseries.csv')
```

In [49]:

```
!cat ch06/tseries.csv
```

```
,0
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
```

3) Sınırlandırılmış Biçimlerle Çalışma

Pandas.read_table gibi fonksiyonları kullanarak çoğu tablolu veri biçimini diskten yüklemek mümkündür. Ancak bazı durumlarda, bazı manuel işlemler gerekli olabilir. Read_table hatası veren bir veya daha fazla hatalı biçimlenmiş satır içeren bir dosya almak alışılmadık bir durum değildir.

In [50]:

```
!cat ch06/ex7.csv
```

```
"a","b","c"
"1","2","3"
"1","2","3","4"
```

Tek karakterlik sınırlayıcıya sahip herhangi bir dosya için Python'un csv modülünü kullanabilirsiniz. Kullanmak için, herhangi bir açık dosyayı veya dosya benzeri nesneyi csv.reader'a iletin.

In [51]:

```
import csv
f = open('ch06/ex7.csv')
reader = csv.reader(f)
```

In [52]:

```
for line in reader:
    print(line)
```

```
['a', 'b', 'c']
['1', '2', '3']
['1', '2', '3', '4']
```

İlk olarak, dosyayı bir satır listesine çevirin.

In [53]:

```
with open('ch06/ex7.csv') as f:
    lines = list(csv.reader(f))
```

Ardından satırları başlık satırına ve veri satırlarına böleriz.

In [54]:

```
header, values = lines[0], lines[1:]
```

Sonra bir sözlük anlayışı kullanarak bir veri sütunları sözlüğü oluşturabiliyoruz ve satırları sütunlara dönüştüren zip (*değerler) ifadesi:

In [55]:

```
data_dict = {h: v for h, v in zip(header, *values)}
```

In [56]:

```
data_dict
```

Out[56]:

```
{'a': ('1', '1'), 'b': ('2', '2'), 'c': ('3', '3')}
```

CSV dosyalarının birçok farklı türü vardır. Farklı bir sınırlayıcı, dize tırnaklama kuralı veya satır sonlandırıcı ile yeni bir biçim tanımlamak için, csv.Dialect'in basit bir alt sınıfını tanımlarız.

```
class my_dialect(csv.Dialect):
    lineterminator = '\n'
    delimiter = ';'
    quotechar = '"'
    quoting = csv.QUOTE_MINIMAL

reader = csv.reader(f, dialect=my_dialect)
reader = csv.reader(f, delimiter='|')
```

In [57]:

Image("img/picture44.png")

Out[57]:

Table 6-3. CSV dialect options

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 csv.QUOTE_ALL (quote all fields), csv.QUOTE_MINIMAL (only fields with special characters like the delimiter), csv.QUOTE_NONNUMERIC, and csv.QUOTE_NONE (no quoting). See Python's documentation for full details. Defaults to QUOTE_MINIMAL.
skipinitialspace	Ignore whitespace after each delimiter; default is False.
doublequote	How to handle quoting character inside a field; if True, it is doubled (see online documentation for full detail and behavior).
escapechar	String to escape the delimiter if quoting is set to csv.QUOTE_NONE; disabled by default.

Sınırlandırılmış dosyaları manuel olarak yazmak için csv.writer'ı kullanabilirsiniz. Açık, yazılabilir bir dosya nesnesini ve csv.reader ile aynı lehçe ve biçim seçeneklerini kabul eder.

```
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))
```

3) JSON Data

JSON (JavaScript Object Notation'ın kısaltması), web tarayıcıları ve diğer uygulamalar arasında HTTP isteği ile veri göndermek için standart formatlardan biri haline geldi. CSV gibi tablo şeklinde bir metin biçiminden çok daha serbest biçimli bir veri biçimidir. İşte bir örnek:

In [59]:

```
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"]}]
} """
```

JSON, boş değeri null ve diğer bazı nüanslar (listelerin sonundaki sondaki virgülere izin vermemek gibi) dışında neredeyse geçerli bir Python kodudur.

Temel türler nesneler (dicts), diziler (listeler), dizeler, sayılar, boolea n'lar ve boş değerlerdir. Bir nesnedeki tüm anahtarlar dizge olmalıdır. JSON verilerini okumak ve yazmak için birkaç Python kitaplığı vardır. Python standard kitaplığında yerlesik olduğu için burada json kullanacağız. JSON dizesini Python biçimine dönüştürmek için json.loads kullanın.

In [60]:

```
import json
```

In [61]:

```
result = json.loads(obj)
```

In [62]:

```
result
```

Out[62]:

```
{'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']}]}
```

Diğer yandan json.dumps, bir Python nesnesini tekrar JSON'a dönüştürür.

In [63]:

```
asjson = json.dumps(result)
```

Analiz için bir JSON nesnesini veya nesne listesini DataFrame'e veya başka bir veri yapısına nasıl dönüştürebileceğiniz size bağlıdır. Elverişli olarak, DataFrame yapıcısına bir dikt listesi (önceden JSON nesneleri idi) geçirilebilir ve veri alanlarının bir alt kümesini seçebilirsiniz.

In [64]:

```
siblings = pd.DataFrame(result['siblings'], columns=['name', 'age'])
```

In [65]:

```
siblings
```

Out[65]:

	name	age
0	Scott	30
1	Katie	38

Pandas.read_json, belirli düzenlemelerdeki JSON veri kümelerini otomatik olarak Seri veya Veri Çerçeve sine dönüştürebilir. Örneğin:

```
!cat ch06/example.json
[{"a": 1, "b": 2, "c": 3},
 {"a": 4, "b": 5, "c": 6},
 {"a": 7, "b": 8, "c": 9}]
```

Pandas.read_json için varsayılan seçenekler, JSON dizisindeki her nesnenin tablodaki bir satır olduğunu varsayar.

```
data = pd.read_json('ch06/example.json')
data
   a b c
0 1 2 3
1 4 5 6
2 7 8 9
```

Pandas'tan JSON'a veri aktarmanız gerekiyorsa, bunun bir yolu Series ve DataFrame'de to_json yöntemlerini kullanmaktır.

In [66]:

```
print(data.to_json())
```

```
{"something":{"0":"one","1":"two","2":"three"}, "a":{"0":1,"1":5,"2":9}, "b":{"0":2,"1":6,"2":10}, "c":{"0":3.0,"1":null,"2":11.0}, "d":{"0":4,"1":8,"2":12}, "message":{"0":null,"1":"world","2":"foo"}}
```

In [67]:

```
print(data.to_json(orient='records'))
```

```
[{"something": "one", "a": 1, "b": 2, "c": 3.0, "d": 4, "message": null}, {"something": "two", "a": 5, "b": 6, "c": null, "d": 8, "message": "world"}, {"something": "three", "a": 9, "b": 10, "c": 11.0, "d": 12, "message": "foo"}]
```

4) XML and HTML: Web Scraping

Python, her yerde bulunan HTML ve XML formatlarında veri okumak ve yazmak için birçok kitaplığa sahiptir. Örnekler arasında lxml, BeautifulSoup ve html5lib bulunur. Genel olarak lxml nispeten daha hızlı olsa da, diğer kitaplıklar hatalı biçimlendirilmiş HTML veya XML dosyalarını daha iyi işleyebilir.

Pandas, HTML dosyalarındaki tabloları otomatik olarak DataFrame nesneleri olarak ayırtırmak için lxml ve BeautifulSoup gibi kitaplıklarını kullanan bir `read_html` fonksiyona sahiptir.

Pandas.`read_html` işlevinin bir dizi seçeneği vardır, ancak varsayılan olarak `<table>` etiketlerinde bulunan tüm tablo verilerini arar ve ayırtırmaya çalışır. Sonuç, DataFrame nesnelerinin bir listesidir.

In []:

```
!conda install lxml
```

Collecting package metadata (current_repodata.json): done
 Solving environment: done

Package Plan

environment location: /Users/veyseldogan/opt/anaconda3

added / updated specs:
 - lxml

The following packages will be downloaded:

package	build			
ca-certificates-2020.12.5	h033912b_0	138 KB	cond	a-forge
certifi-2020.12.5	py38h50d1736_0	143 KB	cond	a-forge
lxml-4.6.2	py38h249bda7_0	1.2 MB	cond	a-forge
openssl-1.1.1h	haf1e3a3_0	1.9 MB	cond	a-forge
<hr/>				
	Total:	3.4 MB		

The following packages will be UPDATED:

ca-certificates anaconda::ca-certificates-2020.10.14-0 --> conda-forge::ca-certificates-2020.12.5-h033912b_0
 certifi anaconda::certifi-2020.6.20-py38_0 --> conda-forge::certifi-2020.12.5-py38h50d1736_0

The following packages will be SUPERSEDED by a higher-priority channel:

lxml pkgs/main::lxml-4.6.2-py38h26b266a_0 --> conda-forge::lxml-4.6.2-py38h249bda7_0
 openssl anaconda --> conda-forge

Proceed ([y]/n)?

In []:

```
!pip install beautifulsoup4 html5lib
```

In []:

```
!pip install lxml
```

In []:

```
import pandas as pd
tables = pd.read_html('examples/fdic_failed_bank_list.html')
```

In []:

```
len(tables)
```

In []:

```
failures = tables[0]
```

In []:

```
failures.head()
```

Hataların birçok sütunda olduğundan, pandas bir satır sonu karakteri ekler \.

In []:

```
close_timestamps = pd.to_datetime(failures['Closing Date'])
```

In []:

```
close_timestamps.dt.year.value_counts()
```

1) XML'i lxml.objectify ile ayrıştırma

XML (eXtensible Markup Language) hiyerarşik, iç içe geçmiş verileri meta verilerle destekleyen başka bir yaygın yapılandırılmış veri formatıdır. Şu anda okuduğunuz kitap aslında bir dizi büyük XML belgeden oluşturulmuştur.

```
<INDICATOR> <INDICATOR_SEQ>373889</INDICATOR_SEQ> <PARENT_SEQ></PARENT_SEQ>
MemphisTN 9956 Milwaukee WI 20364
Acquiring Institution
    Today's Bank September 23, 2016 November 17, 2016
    Closing Date Updated Date
    <AGENCY_NAME>Metro-North Railroad</AGENCY_NAME>
    <INDICATOR_NAME>Escalator Availability</INDICATOR_NAME>
    <DESCRIPTION>Percent of the time that escalators are operational
    systemwide. The availability rate is based on physical observations performed
    the morning of regular business days only. This is a new indicator the agency
    began reporting in 2009.</DESCRIPTION>
    <PERIOD_YEAR>2011</PERIOD_YEAR> <PERIOD_MONTH>12</PERIOD_MONTH> <CATEGORY>Service Indicators</CATEGORY> <FREQUENCY>M</FREQUENCY> <DESIRED_CHANGE>U</DESIRED_CHANGE> <INDICATOR_UNIT>%</INDICATOR_UNIT> <DECIMAL_PLACES>1</DECIMAL_PLACES> <YTD_TARGET>97.00</YTD_TARGET> <YTD_ACTUAL></YTD_ACTUAL> <MONTHLY_TARGET>97.00</MONTHLY_TARGET> <MONTHLY_ACTUAL></MONTHLY_ACTUAL>
</INDICATOR>
```

New York Metropolitan Transportation Authority (MTA) otobüs ve tren hizmetleri hakkında bir dizi veri dizisi yayınıyor. Burada, bir dizi XML dosyasında bulunan performans verilerine bakacağımız. Her tren veya otobüs hizmetinin, aşağıdaki gibi görünen bir dizi XML kaydı olarak aylık verileri içeren farklı bir dosyası (Metro-Kuzey Demiryolu için Performance_MNR.xml gibi) vardır:

Lxml.objectify kullanarak, dosyayı ayırtırır ve getroot ile XML dosyasının kök düğümüne bir referans alırız.

In []:

```
from lxml import objectify
```

In []:

```
path = 'datasets/mta_perf/Performance_MNR.xml'
parsed = objectify.parse(open(path))
root = parsed.getroot()
```

root.INDICATOR, her <INDICATOR> XML öğesini veren bir oluşturucu döndürür. Her kayıt için, veri değerlerine (birkaç etiket hariç) bir etiket adı diktesi (YTD_ACTUAL gibi) yerlestirebiliriz.

In []:

```
data = []

skip_fields = ['PARENT_SEQ', 'INDICATOR_SEQ',
               'DESIRED_CHANGE', 'DECIMAL_PLACES']

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)
```

Son olarak, bu dicts listesini bir DataFrame'e dönüştürün.

In []:

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

In []:

```
perf.head()
```

XML verileri bu örnektenden çok daha karmaşık hale gelebilir. Her etiketin meta verileri de olabilir. Aynı zamanda geçerli XML olan bir HTML bağlantı etiket i düşünün.

In []:

```
from io import StringIO
tag = '<a href="http://www.google.com">Google</a>'
root = objectify.parse(StringIO(tag)).getroot()
```

Artık etiket veya bağlantı metnindeki herhangi bir alana (href gibi) erişebilirsiniz.

In []:

```
root
```

In []:

```
root.get('href')
```

In []:

```
root.text
```

6.2 İkili Veri Formatları

Verileri (serileştirme olarak da bilinir) verimli bir şekilde ikili biçimde depolamanın en kolay yollarından biri Python'un built-in pickle serialization'u kullanmaktadır. Pandas nesnelerinin tümü, verileri diske pickle biçiminde yazan bir to_pickle yöntemine sahiptir.

In []:

```
frame = pd.read_csv('examples/ex1.csv')
```

In []:

```
frame
```

In []:

```
frame.to_pickle('examples/frame_pickle')
```

In []:

```
pd.read_pickle('examples/frame_pickle')
```

Pandas, iki tane daha ikili veri formatı için yerleşik desteği sahiptir: HDF 5 ve Mesaj Paketi. Pandas veya NumPy verileri için diğer bazı depolama biçimleri şunları içerir:

Bcolz:

Blosc sıkıştırma kitaplığına dayalı, sıkıştırılabilir sütun yönelimli ikili biçim.

Feather:

R program topluluğunun Hadley Wickham'ıyla tasarladığım diller arası sütun o daklı bir dosya formatı. Tüy, Apache Arrow sütun bellek formatını kullanır.

2) HDF5 Formatı Kullanmak

HDF5, büyük miktarlarda bilimsel dizi verisini depolamaya yönelik, saygınlık bir dosya formatıdır. C kütüphanesi olarak mevcuttur ve Java, Julia, MATLAB ve Python dahil olmak üzere diğer birçok dilde kullanılabilen arayzlere sahiptir. HDF5'teki "HDF", hiyerarşik veri formatı anlamına gelir. Her bir HDF5 dosyası birden fazla veri kümesini depolayabilir ve meta verileri destekleyebilir. Daha basit formatlarla karşılaştırıldığında HDF5, çeşitli sıkıştırma modlarıyla anında sıkıştırmayı destekler ve tekrarlanan desenlere sahip verilerin daha verimli bir şekilde depolanmasını sağlar. Çok daha büyük dizilerin küçük bölümlerini verimli bir şekilde okuyup yazabildiğiniz için, HDF5 belleğe sıçmayan çok büyük veri kümeleriyle çalışmak için iyi bir seçim olabilir.

PyTables veya h5py kitaplıklarını kullanarak HDF5 dosyalarına doğrudan erişmek mümkün olsa da pandas, Series ve DataFrame nesnelerini depolamayı basitleştiren yüksek seviyeli bir arayüz sağlar. HDFStore sınıfı bir kural gibi çalısır ve düşük düzeyli ayrıntıları işler.

In []:

```
!conda install pytables
```

In []:

```
!pip install --user tables
```

In []:

```
import numpy as np
import pandas as pd
frame = pd.DataFrame({'a': np.random.randn(100)})
```

In []:

```
store = pd.HDFStore('mydata.h5')
```

In []:

```
store['obj1'] = frame
```

In []:

```
store['obj1_col'] = frame['a']
```

In []:

```
store
```

HDF5 dosyasında bulunan nesneler daha sonra aynı dikte benzeri API ile alınabilir.

In []:

```
store['obj1']
```

HDFStore, "sabit" ve "tablo" olmak üzere iki depolama şemasını destekler. İkinci incisi genellikle daha yavaştır, ancak özel bir sözdizimi kullanan sorgu işlemelerini destekler.

In []:

```
store.put('obj2', frame, format='table')
```

In []:

```
store.select('obj2', where=['index >= 10 and index <= 15'])
```

In []:

```
store.close()
```

Pandas.read_hdf işlevi, bu araçlara kısayol sağlar:

In []:

```
frame.to_hdf('mydata.h5', 'obj3', format='table')
```

In []:

```
pd.read_hdf('mydata.h5', 'obj3', where=['index < 5'])
```

3) Microsoft Excel Dosyalarını Okuma

Pandas, ExcelFile sınıfını veya pandas.read_excel işlevini kullanarak Excel 2003 (ve üstü) dosyalarında depolanan tablo verilerinin okunmasını da destekler. Dahili olarak bu araçlar, sırasıyla XLS ve XLSX dosyalarını okumak için xlrd ve openpyxl eklenti paketlerini kullanır.

```
In [ ]:
```

```
xlsx = pd.ExcelFile('examples/ex1.xlsx')
```

Bir sayfada depolanan veriler daha sonra ayrıştırılarak DataFrame'e okunabilir.

```
In [ ]:
```

```
pd.read_excel(xlsx, 'Sheet1')
```

```
In [ ]:
```

```
frame = pd.read_excel('examples/ex1.xlsx', 'Sheet1')
```

```
In [ ]:
```

```
frame
```

Pandas Göndermek Excel yazmak için önce bir ExcelWriter, ardından pandas nesnelerinin to_excel yöntemini kullanarak ona veri yazmanız gereklidir.

```
In [ ]:
```

```
writer = pd.ExcelWriter('examples/ex2.xlsx')
```

```
In [ ]:
```

```
frame.to_excel(writer, 'Sheet1')
```

```
In [ ]:
```

```
writer.save()
```

Ayrıca, to_excel'e bir dosya yolu iletebilir ve ExcelWriter'dan kaçınabilirsiniz.

```
In [ ]:
```

```
frame.to_excel('examples/ex2.xlsx')
```

6.3 Web API'leri ile Etkileşim

Birçok web sitesinde JSON veya başka bir format aracılığıyla veri beslemeleri sağlayan genel API'ler bulunur. Bu API'lere Python'dan erişmenin birkaç yoludur.

```
In [ ]:
```

```
import requests
```

```
In [ ]:
```

```
url = 'https://api.github.com/repos/pandas-dev/pandas/issues'
```

```
In [ ]:
```

```
resp = requests.get(url)
```

```
resp
```

Response nesnesinin json yöntemi, yerel Python nesnelerine ayrıstırılmış JSON içeren bir sözlük döndürür.

```
In [ ]:
```

```
data = resp.json()
```

```
In [ ]:
```

```
data[0]['title']
```

Verilerdeki her öğe, GitHub sorun sayfasında bulunan tüm verileri içeren bir sözlüktür (yorumlar hariç). Verileri doğrudan DataFrame'e aktarabilir ve ilgili alanlarını çıkarabiliriz.

```
In [ ]:
```

```
issues = pd.DataFrame(data, columns=['number', 'title',  
                                      'labels', 'state'])
```

```
In [ ]:
```

```
issues
```

Kolay analiz için DataFrame nesnelerini döndüren yaygın web API'lerine bazı üst düzey arayüzler oluşturabilirsiniz.

6.4 Veritabanları ile Etkileşim

Bir iş ortamında, çoğu veri metin veya Excel dosyalarında depolanmayabilir.

SQL tabanlı ilişkisel veritabanları (SQL Server, PostgreSQL ve MySQL gibi) yaygın olarak kullanılmaktadır ve birçok alternatif veritabanı oldukça popüler hale gelmiştir. Veritabanı seçimi genellikle bir uygulamanın performansına, veri bütünlüğüne ve ölçeklenebilirlik ihtiyaçlarına bağlıdır.

SQL'den DataFrame'e veri yüklemek oldukça basittir ve pandas işlemi basitleştirmek için bazı fonksiyonları vardır. Örnek olarak, Python'un yerleşiksqlite3 sürücüsünü kullanarak bir SQLite veritabanı oluşturalım.

In []:

```
import sqlite3
```

In []:

```
query = """
.....: CREATE TABLE test
.....: (a VARCHAR(20), b VARCHAR(20),
.....:      c REAL,           d INTEGER
.....: );"""
```

In []:

```
con = sqlite3.connect('mydata.sqlite')
```

In []:

```
con.execute(query)
```

In []:

```
con.commit()
```

Ardından, birkaç veri satırı ekleyin.

In []:

```
data = [('Atlanta', 'Georgia', 1.25, 6),
        ('Tallahassee', 'Florida', 2.6, 3),
        ('Sacramento', 'California', 1.7, 5)]
```

In []:

```
stmt = "INSERT INTO test VALUES(?, ?, ?, ?, ?)"
```

In []:

```
con.executemany(stmt, data)
```

In []:

```
con.commit()
```

Çoğu Python SQL sürücüsü (PyODBC, psycopg2, MySQLdb, pymysql, vb.) Bir tablo dan veri seğerken bir tuple listesi döndürür.

In []:

```
cursor = con.execute('select * from test')
```

In []:

```
rows = cursor.fetchall()
```

In []:

```
rows
```

Tuple listesini DataFrame yapıcısına iletebilirsiniz, ancak imlecin açıklama özniteliğinde bulunan sütun adlarına da ihtiyacınız vardır.

In []:

```
cursor.description
```

In []:

```
pd.DataFrame(rows, columns=[x[0] for x in cursor.description])
```

Bu, veritabanını her sorguladığınızda tekrar etmemeyi tercih ettiğiniz için epeyce bir parça. SQLAlchemy projesi, SQL veritabanları arasındaki yaygın farkların çoğunu ortadan kaldırın popüler bir Python SQL araç setidir. Pandas, verileri genel bir SQLAlchemy bağlantısından kolayca okumanızı sağlayan bir `read_sql` işlevine sahiptir.

```
In [135]: import sqlalchemy as sqla
In [136]: db = sqla.create_engine('sqlite:///mydata.sqlite')
In [137]: pd.read_sql('select * from test', db)
Out[137]:
   a   b       c   d
0  Atlanta  Georgia  1.25  6
1  Tallahassee  Florida  2.60  3
2  Sacramento  California  1.70  5
```

Veri Temizleme ve Hazırlama

Veri analizi ve modelleme yapılırken, veri hazırlamaya önemli miktarda zaman harcanır: yükleme, temizleme, dönüştürme ve yeniden düzenleme. Bu tür görevlerin genellikle analistin zamanının % 80'ini veya daha fazlasını aldığı bildirilir. Bazen verilerin dosyalarda veya veritabanlarında depolanma şekli, belirli bir görev için doğru formatta olmayabilir. Pek çok araştırmacı, Python, Perl, R veya Java gibi genel amaçlı bir programlama dili veya sed veya awk gibi Unix metin işleme araçları kullanarak bir formdan diğerine geçici veri işlemeyi seçer. Neyse ki pandas, yerleşik Python dil özelliklerinin yanı sıra, verileri doğru biçimde dönüştürmenize olanak tanıyan yüksek düzeyde, esnek ve hızlı bir araç seti sağlar.

7.1 Eksik Verilerin İşlenmesi

Eksik veriler, birçok veri analizi uygulamasında yaygın olarak ortaya çıkar. Pandaların amaçlarından biri, eksik verilerle çalışmayı mümkün olduğunda zahmetsiz hale getirmektir. Örneğin, pandalar nesneleriyle ilgili tüm açıklayıcı istatistikler eksik verileri varsayılan olarak hariç tutar.

Pandas nesnelerinde eksik verilerin temsil edilme biçimini bir şekilde kusurludur, ancak birçok kullanıcı için işlevseldir. Pandas, sayısal veriler için eksik verileri temsil etmek için kayan nokta değeri NaN (Sayı Değil) kullanır. Buna, kolayca tespit edilebilen bir sentinel değer diyoruz.

In [1]:

```
import pandas as pd
import numpy as np
string_data = pd.Series(['aardvark', 'artichoke', np.nan, 'avocado'])
```

In [2]:

```
string_data
```

Out[2]:

```
0    aardvark
1    artichoke
2      NaN
3    avocado
dtype: object
```

In [3]:

```
string_data.isnull()
```

Out[3]:

```
0    False
1    False
2     True
3    False
dtype: bool
```

İstatistik uygulamalarında NA verileri, var olmayan veya var olan ancak gözlemlenmeyen veriler olabilir (örneğin, veri toplama ile ilgili sorunlar yoluya). Analiz için verileri temizlerken, veri toplama sorunlarını veya eksik verilerin neden olduğu verilerdeki olası önyargıları belirlemek için eksik ve rilerin kendisi üzerinde analiz yapmak genellikle önemlidir.

In [4]:

```
string_data[0] = None
```

In [5]:

```
string_data.isnull()
```

Out[5]:

```
0     True
1    False
2     True
3    False
dtype: bool
```

In [6]:

```
from IPython.display import Image
Image("img/picture45.png")
```

Out[6]:

Table 7-1. NA handling methods

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 her zaman pandas.isnull ve boolean indekslemeyi kullanarak elle yapma seçeneğiniz olsa da, dro

pna yardımcı olabilir. Bir Seride, Serileri yalnızca boş olmayan veriler ve dizin değerleri ile döndürür.

In [7]:

```
from numpy import nan as NA
```

In [8]:

```
data = pd.Series([1, NA, 3.5, NA, 7])
```

In [9]:

```
data.dropna()
```

Out[9]:

```
0    1.0
2    3.5
4    7.0
dtype: float64
```

Bu şuna eşdeğерdir:

In [10]:

```
data[data.notnull()]
```

Out[10]:

```
0    1.0
2    3.5
4    7.0
dtype: float64
```

DataFrame nesnelerinde işler biraz daha karmaşıktır. Tamamı NA olan veya yalnızca herhangi bir NA içeren satırları veya sütunları kaldırmak isteyebilirsiniz. dropna varsayılan olarak eksik bir değer içeren herhangi bir satırı bırakır.

In [11]:

```
data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],
                     [NA, NA, NA], [NA, 6.5, 3.]])
```

In [12]:

```
cleaned = data.dropna()
```

In [13]:

```
data
```

Out[13]:

	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 [14]:

```
cleaned
```

Out[14]:

	0	1	2
0	1.0	6.5	3.0

Passing how='all' yalnızca tümü NA olan satırları kaldıracaktır.

In [15]:

```
data.dropna(how='all')
```

Out[15]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
3	NaN	6.5	3.0

Sütunları aynı şekilde bırakmak için, axis = 1 değerini iletin.

In [16]:

```
data[4] = NA
```

In [17]:

```
data
```

Out[17]:

	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 [18]:

```
data.dropna(axis=1, how='all')
```

Out[18]:

	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 ilgili bir yolu, zaman serisi verileriyile ilgili olma eğilimindedir. Yalnızca belirli sayıda gözlem içeren satırları tutmak istediğinizizi varsayıyalım. Bunu eşik argümanıyla belirtebilirsiniz.

In [19]:

```
df = pd.DataFrame(np.random.randn(7, 3))
```

In [20]:

```
df.iloc[:4, 1] = NA
```

In [21]:

```
df.iloc[:2, 2] = NA
```

In [22]:

df

Out[22]:

	0	1	2
0	-0.458417	NaN	NaN
1	1.283162	NaN	NaN
2	-0.966634	NaN	0.862148
3	-0.479179	NaN	0.839415
4	-0.968487	0.254166	-0.030159
5	-0.933947	-1.393678	-0.493367
6	-2.281998	-0.017278	1.543737

In [23]:

df.dropna()

Out[23]:

	0	1	2
4	-0.968487	0.254166	-0.030159
5	-0.933947	-1.393678	-0.493367
6	-2.281998	-0.017278	1.543737

In [24]:

df.dropna(thresh=2)

Out[24]:

	0	1	2
2	-0.966634	NaN	0.862148
3	-0.479179	NaN	0.839415
4	-0.968487	0.254166	-0.030159
5	-0.933947	-1.393678	-0.493367
6	-2.281998	-0.017278	1.543737

1) Eksik Verilerin Doldurulması

Eksik verileri filtrelemek (ve potansiyel olarak onunla birlikte diğer verileri atmak) yerine, "boşlukları" herhangi bir şekilde doldurmak isteyebilirsiniz. Çoğu amaç için, fillna yöntemi kullanılacak en güçlü işlevdir. Fillna'yı bir sabitle çağırırmak, eksik değerleri bu değerle değiştirir.

In [25]:

df.fillna(0)

Out[25]:

	0	1	2
0	-0.458417	0.000000	0.000000
1	1.283162	0.000000	0.000000
2	-0.966634	0.000000	0.862148
3	-0.479179	0.000000	0.839415
4	-0.968487	0.254166	-0.030159
5	-0.933947	-1.393678	-0.493367
6	-2.281998	-0.017278	1.543737

Fillna'yi bir dikteyle çağırarak, her sütun için farklı bir dolgu değeri kullanabilirsiniz.

In [26]:

df.fillna({1: 0.5, 2: 0})

Out[26]:

	0	1	2
0	-0.458417	0.500000	0.000000
1	1.283162	0.500000	0.000000
2	-0.966634	0.500000	0.862148
3	-0.479179	0.500000	0.839415
4	-0.968487	0.254166	-0.030159
5	-0.933947	-1.393678	-0.493367
6	-2.281998	-0.017278	1.543737

fillna yeni bir nesne döndürür, ancak mevcut nesneyi yerinde değiştirebilirsiniz.

In [27]:

_= df.fillna(0, inplace=True)

In [28]:

df

Out[28]:

	0	1	2
0	-0.458417	0.000000	0.000000
1	1.283162	0.000000	0.000000
2	-0.966634	0.000000	0.862148
3	-0.479179	0.000000	0.839415
4	-0.968487	0.254166	-0.030159
5	-0.933947	-1.393678	-0.493367
6	-2.281998	-0.017278	1.543737

Yeniden indeksleme için mevcut olan aynı interpolasyon yöntemleri fillna ile kullanılabilir.

In [29]:

df = pd.DataFrame(np.random.randn(6, 3))

In [30]:

df.iloc[2:, 1] = NA

In [31]:

df.iloc[4:, 2] = NA

In [32]:

df

Out[32]:

	0	1	2
0	-0.852809	-0.469623	-1.176329
1	-1.278962	-0.754296	-1.172694
2	0.539798	NaN	1.784983
3	-2.392411	NaN	2.536561
4	1.671046	NaN	NaN
5	-0.020841	NaN	NaN

In [33]:

```
df.fillna(method='ffill')
```

Out[33]:

	0	1	2
0	-0.852809	-0.469623	-1.176329
1	-1.278962	-0.754296	-1.172694
2	0.539798	-0.754296	1.784983
3	-2.392411	-0.754296	2.536561
4	1.671046	-0.754296	2.536561
5	-0.020841	-0.754296	2.536561

In [34]:

```
df.fillna(method='ffill', limit=2)
```

Out[34]:

	0	1	2
0	-0.852809	-0.469623	-1.176329
1	-1.278962	-0.754296	-1.172694
2	0.539798	-0.754296	1.784983
3	-2.392411	-0.754296	2.536561
4	1.671046	NaN	2.536561
5	-0.020841	NaN	2.536561

Fillna ile biraz yaratıcılıkla başka birçok şey yapabilirsiniz. Örneğin, bir Serinin ortalama veya medyan değerini iletebilirsiniz.

In [35]:

```
data = pd.Series([1., NA, 3.5, NA, 7])
```

In [36]:

```
data.fillna(data.mean())
```

Out[36]:

```
0    1.000000
1    3.833333
2    3.500000
3    3.833333
4    7.000000
dtype: float64
```

In [37]:

Image("img/picture46.png")

Out[37]:

Table 7-2. *fillna* function arguments

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

7.2 Veri Dönüşümü

Filtreleme, temizleme ve diğer dönüşümler bir başka önemli işlem sınıfıdır.

Yinelenenleri Kaldırma

DataFrame'de birçok nedenden dolayı yinelenen satırlar bulunabilir. İşte bir örnek:

In [38]:

```
data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],
                     'k2': [1, 1, 2, 3, 3, 4, 4]})
```

In [39]:

data

Out[39]:

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4

Yinelenen DataFrame yöntemi, her satırın yinelenip yinelenmediğini (önceki bir satırda gözlemlenmiştir) gösteren bir boolean Serisi döndürür.

In [40]:

```
data.duplicated()
```

Out[40]:

```
0    False
1    False
2    False
3    False
4    False
5    False
6    True
dtype: bool
```

Benzer şekilde, drop_duplicates, yinelenen dizinin False olduğu bir DataFrame döndürür.

In [41]:

```
data.drop_duplicates()
```

Out[41]:

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4

Bu yöntemlerin her ikisi de varsayılan olarak tüm sütunları dikkate alır; alternatif olarak, kopyaları algılamak için bunların herhangi bir alt kümesini belirtebilirsiniz. Ek bir değer sütunumuz olduğunu ve kopyaları yalnızca "k" sütununa göre filtrelemek istediğimizi varsayıyalım:

In [42]:

```
data['v1'] = range(7)
```

In [43]:

```
data.drop_duplicates(['k1'])
```

Out[43]:

	k1	k2	v1
0	one	1	0
1	two	1	1

duplicated ve drop_duplicates varsayılan olarak ilk gözlenen değer kombinasyonunu korur. Keep = 'last' geçmek sonuncuyu döndürür.

In [44]:

```
data.drop_duplicates(['k1', 'k2'], keep='last')
```

Out[44]:

	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

Bir Fonksiyon veya Eşleme Kullanarak Verileri Dönüşürme

Birçok veri kümesi için, bir DataFrame'deki bir dizi, Seri veya sütundaki değerlere dayalı olarak bazı dönüşümler gerçekleştirmek isteyebilirsiniz. Çeşitli et türleri hakkında toplanan aşağıdaki varsayımsal verileri göz önünde bulundurun.

In [45]:

```
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 [46]:

data

Out[46]:

	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

Her gıdanın geldiği hayvanın türünü gösteren bir sütun eklemek istediğinizi varsayıyalım. Her bir et türünün hayvan türüne göre bir haritasını yazalımlım.

In [47]:

```
meat_to_animal = {
    'bacon': 'pig',
    'pulled pork': 'pig',
    'pastrami': 'cow',
    'corned beef': 'cow',
    'honey ham': 'pig',
    'nova lox': 'salmon'
}
```

Bir Serideki map yöntemi, map içeren bir işlevi veya dikte benzeri nesneyi kabul eder, ancak burada bazı etlerin büyük harfle yazılması ve diğerlerinin olmaması nedeniyle küçük bir sorunumuz var. Bu nedenle, str.lower Series yöntemini kullanarak her bir değeri küçük harfe dönüştürmemiz gereklidir.

In [48]:

```
lowercased = data['food'].str.lower()
```

In [49]:

```
lowercased
```

Out[49]:

```
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 [50]:

```
data['animal'] = lowercased.map(meat_to_animal)
```

In [51]:

```
data
```

Out[51]:

	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

In [52]:

```
data['food'].map(lambda x: meat_to_animal[x.lower()])
```

Out[52]:

```
0      pig
1      pig
2      pig
3      cow
4      cow
5      pig
6      cow
7      pig
8  salmon
Name: food, dtype: object
```

map kullanmak, öğe bazında dönüşümler ve diğer veri temizleme ile ilgili iş emleri gerçekleştirmek için uygun bir yoldur.

Değerleri Değiştirme

Eksik verileri fillna yöntemiyle doldurmak, daha genel bir değer değişiminin özel bir durumudur. Daha önce gördüğünüz gibi, harita bir nesnedeki değerlerin bir alt kümesini değiştirmek için kullanılabilir ancak değiştir, bunu yapmak için daha basit ve daha esnek bir yol sağlar. Bu Seriyi ele alalım:

In [53]:

```
import pandas as pd
import numpy as np
data = pd.Series([1., -999., 2., -999., -1000., 3.])
```

In [54]:

```
data
```

Out[54]:

```
0      1.0
1    -999.0
2      2.0
3    -999.0
4   -1000.0
5      3.0
dtype: float64
```

-999 değerleri, eksik veriler için temel değerler olabilir. Bunları, pandas'ın anladığı NA değerleriyle değiştirmek için, yeni bir Seri üreterek replace'i kullanabiliriz (inplace = True geçmedikçe).

In [55]:

```
data.replace(-999, np.nan)
```

Out[55]:

```
0      1.0
1      NaN
2      2.0
3      NaN
4   -1000.0
5      3.0
dtype: float64
```

Aynı anda birden fazla değeri değiştirmek istiyorsanız,

In [56]:

```
data.replace([-999, -1000], np.nan)
```

Out[56]:

```
0    1.0
1    NaN
2    2.0
3    NaN
4    NaN
5    3.0
dtype: float64
```

In [57]:

```
data.replace([-999, -1000], [np.nan, 0])
```

Out[57]:

```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

Aktarılan argüman ayrıca bir dikte de olabilir.

In [58]:

```
data.replace({-999: np.nan, -1000: 0})
```

Out[58]:

```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

Eksen Dizinlerini Yeniden Adlandırma

Bir Serideki değerler gibi, eksen etiketleri de yeni, farklı şekilde etiketlenmiş nesneler üretmek için bir fonksiyon veya bazı formların eşlemesi ile benzer şekilde dönüştürülebilir. Ayrıca, yeni bir veri yapısı oluşturmadan eksenleri yerinde değiştirebilirsiniz.

In [59]:

```
data = pd.DataFrame(np.arange(12).reshape((3, 4)),  
                    index=['Ohio', 'Colorado', 'New York'],  
                    columns=['one', 'two', 'three', 'four'])
```

Seri gibi, eksen dizinlerinin de bir map yöntemi vardır.

In [60]:

```
transform = lambda x: x[:4].upper()
```

In [61]:

```
data.index.map(transform)
```

Out[61]:

```
Index(['OHIO', 'COLO', 'NEW'], dtype='object')
```

DataFrame'i yerinde değiştirerek dizine atayabilirsiniz.

In [62]:

```
data.index = data.index.map(transform)
```

In [63]:

```
data
```

Out[63]:

	one	two	three	four
OHIO	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

Orjinali değiştirmeden bir veri kümесinin dönüştürülmüş bir sürümünü oluşturma istiyorsanız, kullanışlı bir yöntem yeniden adlandırmaktır.

In [64]:

```
data.rename(index=str.title, columns=str.upper)
```

Out[64]:

	ONE	TWO	THREE	FOUR
Ohio	0	1	2	3
Colo	4	5	6	7
New	8	9	10	11

Özellikle, yeniden adlandırma, eksen etiketlerinin bir alt kümesi için yeni değerler sağlayan dikteye benzer bir nesneye birlikte kullanılabilir.

In [65]:

```
data.rename(index={'OHIO': 'INDIANA'},
           columns={'three': 'peekaboo'})
```

Out[65]:

	one	two	peekaboo	four
INDIANA	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

Yeniden adlandırma, sizi DataFrame'i manuel olarak kopyalama ve dizin ve süt un özniteliklerine atama işinden kurtarır. Bir veri setini yerinde değiştirmek isterseniz, inplace = True yapın.

In [66]:

```
data.rename(index={'OHIO': 'INDIANA'}, inplace=True)
```

In [67]:

```
data
```

Out[67]:

	one	two	three	four
INDIANA	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

Discretization ve Binning

Sürekli veriler genellikle ayrılaştırılır veya başka şekilde analiz için "kutulara" ayrılır. Bir araştırmadaki bir grup insan hakkında veriye sahip olduğunuzu ve onları ayrı yaş grupları halinde grüplamak istediğiniz varsayılmıyor:

In [68]:

```
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

Bunları 18 ile 25, 26 ile 35, 36 ile 60 ve son olarak 61 ve daha büyük kutulara ayıralım. İçin pandas'da bir fonksiyon olan cut kullanmanız gereklidir.

In [69]:

```
bins = [18, 25, 35, 60, 100]
```

In [70]:

```
cats = pd.cut(ages, bins)
```

In [71]:

```
cats
```

Out[71]:

```
[(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]): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]
```

Pandas döndürdüğü nesne özel bir Kategorik nesnedir. Gördüğünüz çıktı, pandas.cut tarafından hesaplanan kutuları açıklar. Bin adını belirten dizisi gibi davranışabilirsiniz; dahili olarak, farklı kategori adlarını belirten bir kategori dizisi ve kodlar özelliğindeki yaş verileri için bir etiketleme içerir.

In [72]:

```
cats.codes
```

Out[72]:

```
array([0, 0, 0, 1, 0, 0, 2, 1, 3, 2, 2, 1], dtype=int8)
```

In [73]:

cats.categories

Out[73]:

```
IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]),
              closed='right',
              dtype='interval[int64]')
```

In [74]:

pd.value_counts(cats)

Out[74]:

(18, 25]	5
(35, 60]	3
(25, 35]	3
(60, 100]	1
dtype:	int64

Pd.value_counts (kediler), pandas.cut sonucu için bin sayıları olduğunu unutmayın.

Aralıklar için matematiksel gösterimle tutarlı olarak, bir parantez kenarının açık olduğu, köşeli parantez ise kapalı (dahil) olduğu anlamına gelir. right = False geçerek hangi tarafın kapalı olduğunu değiştirebilirsiniz.

In [75]:

pd.cut(ages, [18, 26, 36, 61, 100], right=False)

Out[75]:

```
[[18, 26), [18, 26), [18, 26), [26, 36), [18, 26), ..., [26, 36), [61,
100), [36, 61), [36, 61), [26, 36)]
Length: 12
Categories (4, interval[int64]): [[18, 26) < [26, 36) < [36, 61) < [6
1, 100)]
```

Ayrıca, etiketler seçeneğine bir liste veya dizi ileterek kendi bölme adları nızı da iletебilirsiniz.

In [76]:

group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']

In [77]:

```
pd.cut(ages, bins, labels=group_names)
```

Out[77]:

```
[Youth, Youth, Youth, YoungAdult, Youth, ..., YoungAdult, Senior, MiddleAged, MiddleAged, YoungAdult]
Length: 12
Categories (4, object): [Youth < YoungAdult < MiddleAged < Senior]
```

Açık bölme kenarları yerine kesmek için tam sayı bölge sayısını iletirseniz, verilerdeki minimum ve maksimum değerleri temel alarak eşit uzunlukta bölmeli hesaplar. Dörde bölünmüş bazı eşit dağıtılmış verilerin durumunu düşünün.

In [78]:

```
data = np.random.rand(20)
```

In [79]:

```
pd.cut(data, 4, precision=2)
```

Out[79]:

```
[(0.014, 0.25], (0.73, 0.97], (0.25, 0.49], (0.25, 0.49], (0.73, 0.97],
 ..., (0.49, 0.73], (0.73, 0.97], (0.73, 0.97], (0.014, 0.25], (0.25, 0.49])
Length: 20
Categories (4, interval[float64]): [(0.014, 0.25] < (0.25, 0.49] < (0.49, 0.73] < (0.73, 0.97)]
```

Precision = 2 seçeneği, ondalık duyarlığını iki basamakla sınırlar.

Yakından ilişkili bir fonksiyon olan qcut, verileri örnek niceliklerine göre depolar. Verilerin dağılımına bağlı olarak, kesimi kullanmak genellikle her bölmenin aynı sayıda veri noktasına sahip olmasına neden olmaz.

In [80]:

```
data = np.random.randn(1000) # Normal dağılım
```

In [81]:

```
cats = pd.qcut(data, 4) # Çeyreğe böl
```

In [82]:

cats

Out[82]:

```
[(-0.69, 0.0796], (0.708, 2.891], (0.0796, 0.708], (0.0796, 0.708], (-0.69, 0.0796], ..., (0.0796, 0.708], (-2.872999999999998, -0.69], (0.708, 2.891], (-0.69, 0.0796], (0.0796, 0.708])
```

Length: 1000

```
Categories (4, interval[float64]): [(-2.872999999999998, -0.69] < (-0.69, 0.0796] < (0.0796, 0.708] < (0.708, 2.891])
```

In [83]:

pd.value_counts(cats)

Out[83]:

(0.708, 2.891]	250
(0.0796, 0.708]	250
(-0.69, 0.0796]	250
(-2.872999999999998, -0.69]	250

dtype: int64

Cut'a benzer şekilde kendi niceliklerinizi aktarabilirsiniz (0 ile 1 arasında sayılar dahil).

In [84]:

pd.qcut(data, [0, 0.1, 0.5, 0.9, 1.])

Out[84]:

```
[(-1.302, 0.0796], (1.323, 2.891], (0.0796, 1.323], (0.0796, 1.323], (-1.302, 0.0796], ..., (0.0796, 1.323], (-2.872999999999998, -1.302], (1.323, 2.891], (-1.302, 0.0796], (0.0796, 1.323])
```

Length: 1000

```
Categories (4, interval[float64]): [(-2.872999999999998, -1.302] < (-1.302, 0.0796] < (0.0796, 1.323] < (1.323, 2.891])
```

Detecting and Filtering Outliers

Aykırı değerlerin filtrelenmesi veya dönüştürülmesi, büyük ölçüde dizi işlemlerini uygulamakla ilgiliidir. Normal olarak dağıtılmış bazı veriler içeren bir DataFrame düşünün.

In [85]:

data = pd.DataFrame(np.random.randn(1000, 4))

In [86]:

```
data.describe()
```

Out[86]:

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.014331	-0.023405	-0.016937	0.030316
std	0.989140	0.989658	0.996610	0.975124
min	-3.024661	-3.465696	-3.530589	-2.747337
25%	-0.660784	-0.619299	-0.633871	-0.620241
50%	-0.001481	0.017185	-0.025651	0.023571
75%	0.663013	0.632913	0.596469	0.677556
max	2.947850	3.034787	3.356700	2.884999

Mutlak değer olarak 3'ü aşan sütunlardan birinde değerler bulmak istediğiniz i varsayıyalım.

In [87]:

```
col = data[2]
```

In [88]:

```
col[np.abs(col) > 3]
```

Out[88]:

```
2      -3.004751
14      3.356700
302     -3.530589
350      3.173067
766      3.237000
Name: 2, dtype: float64
```

3 veya -3'ü aşan bir değere sahip tüm satırları seçmek için, Boolean Dataframe'deki herhangi bir yöntemi kullanabilirsiniz.

In [89]:

```
data[ (np.abs(data) > 3).any(1) ]
```

Out[89]:

	0	1	2	3
2	-2.543617	0.385843	-3.004751	0.473188
14	-0.054385	-1.493799	3.356700	-1.402880
69	1.503818	3.034787	0.391441	-2.420452
212	-1.157102	-3.465696	1.622510	-1.227350
302	-0.386176	-1.019412	-3.530589	-1.316185
350	-1.364555	0.286165	3.173067	-0.909115
468	-0.157741	-3.305889	0.883252	-0.359958
618	-3.024661	0.451220	-0.413561	0.972491
633	-0.270959	-3.418704	-1.248968	0.058171
766	-0.948437	-0.604777	3.237000	0.531985
898	0.069916	-3.014163	-0.265570	1.310407

Değerler bu kriterlere göre belirlenebilir. Aşağıda -3 ile 3 arasındaki değerleri sınırlamak için kod:

In [90]:

```
data[np.abs(data) > 3] = np.sign(data) * 3
```

In [91]:

```
data.describe()
```

Out[91]:

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.014355	-0.022236	-0.017168	0.030316
std	0.989064	0.985671	0.992425	0.975124
min	-3.000000	-3.000000	-3.000000	-2.747337
25%	-0.660784	-0.619299	-0.633871	-0.620241
50%	-0.001481	0.017185	-0.025651	0.023571
75%	0.663013	0.632913	0.596469	0.677556
max	2.947850	3.000000	3.000000	2.884999

Np.sign (data) ifadesi, verilerdeki değerlerin pozitif veya negatif olmasına

bağlı olarak 1 ve -1 değerleri üretir.

In [92]:

```
np.sign(data).head()
```

Out[92]:

	0	1	2	3
0	1.0	-1.0	-1.0	1.0
1	-1.0	-1.0	-1.0	-1.0
2	-1.0	1.0	-1.0	1.0
3	-1.0	1.0	-1.0	1.0
4	1.0	-1.0	-1.0	1.0

Permütasyon ve Rastgele Örnekleme

DataFrame'deki bir Seriyi veya satırları permütasyon (rastgele yeniden sıralama), numpy.random.permutation fonksiyonunu kullanarak yapmak kolaydır. İzin vermek istediğiniz eksenin uzunluğu ile permütasyon çağırırmak, yeni sıralamayı gösteren bir tamsayı dizisi üretir.

In [93]:

```
df = pd.DataFrame(np.arange(5 * 4).reshape((5, 4)))
```

In [94]:

```
sampler = np.random.permutation(5)
```

In [95]:

```
sampler
```

Out[95]:

```
array([2, 0, 3, 4, 1])
```

Bu dizi daha sonra iloc tabanlı indekslemede veya eşdeğeri take fonksiyonunda kullanılabilir.

In [96]:

df

Out[96]:

	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 [97]:

df.take(sampler)

Out[97]:

	0	1	2	3
2	8	9	10	11
0	0	1	2	3
3	12	13	14	15
4	16	17	18	19
1	4	5	6	7

Değiştirmeden rastgele bir alt küme seçmek için, Series ve DataFrame'de örnek yöntemi kullanabilirsiniz.

In [98]:

df.sample(n=3)

Out[98]:

	0	1	2	3
3	12	13	14	15
4	16	17	18	19
2	8	9	10	11

Değiştirme ile bir numune oluşturmak için (tekrar seçimlerine izin vermek için), replace = True to sample

In [99]:

```
choices = pd.Series([5, 7, -1, 6, 4])
```

In [100]:

```
draws = choices.sample(n=10, replace=True)
```

In [101]:

```
draws
```

Out[101]:

```
3      6
1      7
0      5
3      6
0      5
4      4
2     -1
0      5
0      5
1      7
dtype: int64
```

Hesaplama Göstergesi / Kukla Değişkenler

İstatistiksel modelleme veya makine öğrenimi uygulamaları için başka bir dönüşüm türü, kategorik bir değişkeni "kukla" veya "gösterge" matrisine dönüştürmektedir. Bir DataFrame'deki bir sütunun k farklı değeri varsa, tüm 1'leri ve 0'ları içeren k sütunlu bir matris veya Veri Çerçevesi türetmeliisiniz. Pandas'ın bunu yapmak için bir get_dummies işlevi vardır.

In [102]:

```
df = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                   'data1': range(6)})
```

In [103]:

```
pd.get_dummies(df['key'])
```

Out[103]:

	a	b	c
0	0	1	0
1	0	1	0
2	1	0	0
3	0	0	1
4	1	0	0
5	0	1	0

Bazı durumlarda, gösterge Veri Çerçevesindeki sütunlara, daha sonra diğer ve rilerle birleştirilebilen bir önek eklemek isteyebilirsiniz. `get_dummies`'in bunu yapmak için bir önek argümanı vardır.

In [104]:

```
dummies = pd.get_dummies(df['key'], prefix='key')
```

In [105]:

```
df_with_dummy = df[['data1']].join(dummies)
```

In [106]:

```
df_with_dummy
```

Out[106]:

	data1	key_a	key_b	key_c
0	0	0	1	0
1	1	0	1	0
2	2	1	0	0
3	3	0	0	1
4	4	1	0	0
5	5	0	1	0

Bir DataFrame'deki bir satır birden fazla kategoriye aitse, işler biraz daha karmaşıktır. MovieLens 1M veri kümesine bakalım.

In [107]:

```
mnames = ['movie_id', 'title', 'genres']
import io
```

In [108]:

```
movies = pd.read_table('datasets/movielens/movies.dat', sep='::',
header=None, names=mnames)
```

```
<ipython-input-108-20ac69f290b4>:1: ParserWarning: Falling back to the
'python' engine because the 'c' engine does not support regex separato
rs (separators > 1 char and different from '\s+' are interpreted as re
gex); you can avoid this warning by specifying engine='python'.
  movies = pd.read_table('datasets/movielens/movies.dat', sep='::',
```

In [109]:

movies[:10]

Out[109]:

movie_id		title	genres
0	1	Toy Story (1995)	Animation Children's Comedy
1	2	Jumanji (1995)	Adventure Children's Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama
4	5	Father of the Bride Part II (1995)	Comedy
5	6	Heat (1995)	Action Crime Thriller
6	7	Sabrina (1995)	Comedy Romance
7	8	Tom and Huck (1995)	Adventure Children's
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action Adventure Thriller

Her tür için gösterge değişkenleri eklemek biraz çekişmeyi gerektirir. İlk olarak, veri kümelerindeki benzersiz türlerin listesini çıkarıyoruz.

In [110]:

all_genres = []

In [111]:

```
for x in movies.genres:
    all_genres.extend(x.split(' | '))
```

In [112]:

genres = pd.unique(all_genres)

In [113]:

genres

Out[113]:

```
array(['Animation', "Children's", 'Comedy', 'Adventure', 'Fantasy',
       'Romance', 'Drama', 'Action', 'Crime', 'Thriller', 'Horror',
       'Sci-Fi', 'Documentary', 'War', 'Musical', 'Mystery', 'Film-Noi
r',
       'Western'], dtype=object)
```

DataFrame göstergesini oluşturmanın bir yolu, tüm sıfırlardan oluşan bir DataFrame ile başlamaktır.

In [114]:

```
zero_matrix = np.zeros((len(movies), len(genres)))
```

In [115]:

```
dummies = pd.DataFrame(zero_matrix, columns=genres)
```

Şimdi, her filmi yineleyin ve her bir kukla satırındaki girişleri 1'e ayarla yın. Bunu yapmak için, her tür için sütun indekslerini hesaplamak üzere dummies.columns'u kullanırız.

In [116]:

```
gen = movies.genres[0]
```

In [117]:

```
gen.split('|')
```

Out[117]:

```
['Animation', 'Children\'s', 'Comedy']
```

In [118]:

```
dummies.columns.get_indexer(gen.split('|'))
```

Out[118]:

```
array([0, 1, 2])
```

Daha sonra, bu indekslere dayalı değerleri ayarlamak için .iloc kullanabiliriz.

In [119]:

```
for i, gen in enumerate(movies.genres):
    indices = dummies.columns.get_indexer(gen.split('|'))
    dummies.iloc[i, indices] = 1
```

Ardından, daha önce olduğu gibi, bunu filmlerle birleştirilirsiniz.

In [120]:

```
movies_windic = movies.join(dummies.add_prefix('Genre_'))
```

In [121]:

```
movies_windic.iloc[0]
```

Out[121]:

movie_id	1
title	Toy Story (1995)
genres	Animation Children's Comedy
Genre_Animation	1
Genre_Children's	1
Genre_Comedy	1
Genre_Adventure	0
Genre_Fantasy	0
Genre_Romance	0
Genre_Drama	0
Genre_Action	0
Genre_Crime	0
Genre_Thriller	0
Genre_Horror	0
Genre_Sci-Fi	0
Genre_Documentary	0
Genre_War	0
Genre_Musical	0
Genre_Mystery	0
Genre_Film-Noir	0
Genre_Western	0
Name:	0 , dtype: object

İstatistiksel uygulamalar için kullanımı bir yol, `get_dummies`'i `cut` gibi bir ayrıklaştırma fonksiyonu ile birleştirmektir.

In [122]:

```
np.random.seed(12345)
```

In [123]:

```
values = np.random.rand(10)
```

In [124]:

```
values
```

Out[124]:

```
array([0.92961609, 0.31637555, 0.18391881, 0.20456028, 0.56772503,
       0.5955447 , 0.96451452, 0.6531771 , 0.74890664, 0.65356987])
```

In [125]:

```
bins = [0, 0.2, 0.4, 0.6, 0.8, 1]
```

In [126]:

```
pd.get_dummies(pd.cut(values, bins))
```

Out[126]:

	(0.0, 0.2]	(0.2, 0.4]	(0.4, 0.6]	(0.6, 0.8]	(0.8, 1.0]
0	0	0	0	0	1
1	0	1	0	0	0
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	0
5	0	0	1	0	0
6	0	0	0	0	1
7	0	0	0	1	0
8	0	0	0	1	0
9	0	0	0	1	0

7.3 String Manipülasyonu

Python, dizgi ve metin işleme için kullanım kolaylığı nedeniyle uzun süredir popüler bir ham veri işleme dili olmuştur. Çoğu metin işlemi, dize nesnesini n yerleşik yöntemleriyle basitleştirilir. Daha karmaşık desen eşleştirme ve metin manipülasyonları için düzenli ifadeler gerekebilir. Pandas, dizeyi ve düzenli ifadeleri kısaca tüm veri dizilerine uygulamanızı sağlayarak karışım a ekler ve ayrıca eksik verilerin sıkıntısını giderir.

Dize Nesne Yöntemleri

Pek çok dizi birleştirme ve komut dosyası oluşturma uygulamasında, yerleşik dizi yöntemleri yeterlidir. Örnek olarak, virgülle ayrılmış bir dize bölme ile parçalara ayrılabilir.

In [127]:

```
val = 'a,b, guido'
```

In [128]:

```
val.split(',')
```

Out[128]:

```
['a', 'b', ' guido']
```

split genellikle boşlukları kırpmak için şeritle birleştirilir (satır sonları da dahil).

In [129]:

```
pieces = [x.strip() for x in val.split(',')]
```

In [130]:

```
pieces
```

Out[130]:

```
['a', 'b', 'guido']
```

In [131]:

```
first, second, third = pieces
```

In [132]:

```
first + '::' + second + '::' + third
```

Out[132]:

```
'a::b::guido'
```

Daha iyi bir yol:

In [133]:

```
:::.join(pieces)
```

Out[133]:

```
'a::b::guido'
```

In [134]:

```
'guido' in val
```

Out[134]:

```
True
```

In [135]:

```
val.index(',', 1)
```

Out[135]:

```
1
```

In [136]:

```
val.find(':')
```

Out[136]:

```
-1
```

Bul ve dizin arasındaki farka dikkat edin: dize bulunamadı (-1 döndürür).

```
val.index(':')
```

Buna bağlı olarak, count, belirli bir alt dizenin oluşum sayısını döndürür.

In [138]:

```
val.count(' ', ' )
```

Out[138]:

```
2
```

replace, bir modelin oluşumlarını bir başkasıyla değiştirir. Genellikle boş bir dize geçirerek kalıpları silmek için de kullanılır.

In [139]:

```
val.replace(' ', ' :: ')
```

Out[139]:

```
'a::b:: guido'
```

In [140]:

```
val.replace(' ', ' ')
```

Out[140]:

```
'ab guido'
```

In [141]:

```
from IPython.display import Image
Image("img/picture47.png")
```

Out[141]:

Table 7-3. Python built-in string methods

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.

Regular Expressions

Metindeki dize kalıplarını (genellikle daha karmaşık) aramak veya eşleştirme için esnek bir yol sağlar. Genellikle Regular Expression olarak adlandırılan tek bir ifade, Regular Expression diline göre oluşturulmuş bir dizedir. Python'un yerleşik yeniden modülü, dizelere düzenli ifadeler uygulamaktan sorumludur.

Re modülü işlevleri üç kategoriye ayrılır: örüntü eşleştirme, ikame ve bölüm. Doğal olarak bunların hepsi birbiriyile ilişkilidir; bir normal ifade, dah

a sonra birçok amaç için kullanılabilen, metinde bulunacak bir kalıbı tanımlar. Basit bir örneğe bakalım:
bir dizeyi değişken sayıda boşluk karakteriyle (sekmeler, boşluklar ve satır sonları) bölmek istediğimizi varsayalım. Bir veya daha fazla boşluk karakterini tanımlayan normal ifade \ s + şeklindedir.

In [142]:

`import re`

In [143]:

`text = "foo bar\t baz \tqux"`

In [144]:

`re.split('\s+', text)`

Out[144]:

`['foo', 'bar', 'baz', 'qux']`

`re.split ('\s+', metin)` 'i çağırıldığınızda, ilk olarak normal ifade derlenir ve ardından iletilen metinde bölünme yöntemi çağrılır. Yeniden kullanılabilecek bir regex nesnesi oluşturarak `re.compile` ile regex'i kendiniz derleyebilirsiniz.

In [145]:

`regex = re.compile('\s+')`

In [146]:

`regex.split(text)`

Out[146]:

`['foo', 'bar', 'baz', 'qux']`

Bunun yerine, normal ifadeyle eşleşen tüm kalıpların bir listesini almak istediyiseniz, `findall` kullanırsınız.

In [147]:

`regex.findall(text)`

Out[147]:

`[' ', '\t ', '\t']`

Aynı ifadeyi birçok dizeye uygulamayı düşünüyorsanız, `re.compile` ile bir regex nesnesi oluşturmanız şiddetle önerilir; bunu yapmak CPU döngülerinden kurtaracaktır.

eşleştirme ve arama, findall ile yakından ilgilidir. Findall bir dizedeki tüm eşleşmeleri döndürürken, arama yalnızca ilk eşleşmeyi döndürür. Daha kesin bir ifadeyle, yalnızca dizenin başındaki eşleşmeleri eşleştirin. Daha az önesiz bir örnek olarak, çoğu e-posta adresini tanımlayabilen bir metin bloğunu ve normal bir ifadeyi ele alalım.

In [148]:

```
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}'
# re.IGNORECASE makes the regex case-insensitive
regex = re.compile(pattern, flags=re.IGNORECASE)
```

Metinde findall kullanmak, e-posta adreslerinin bir listesini oluşturur.

In [149]:

```
regex.findall(text)
```

Out[149]:

```
['dave@google.com', 'steve@gmail.com', 'rob@gmail.com', 'ryan@yahoo.co
m']
```

search, metindeki ilk e-posta adresi için özel bir eşleşme nesnesi döndürür. Önceki normal ifade için, eşleştirme nesnesi bize dizedeki kalıbin yalnızca başlangıç ve bitiş konumunu söyleyebilir.

In [150]:

```
m = regex.search(text)
```

In [151]:

```
m
```

Out[151]:

```
<re.Match object; span=(5, 20), match='dave@google.com'>
```

In [152]:

```
text[m.start():m.end()]
```

Out[152]:

```
'dave@google.com'
```

`regex.match`, yalnızca şablonun başlangıcında meydana gelirse eşleşeceği için Yok değerini döndürür.

In [153]:

```
print(regex.match(text))
```

None

Buna bağlı olarak, `sub`, örüntü oluşumlarının yeni bir dizeye değiştirildiği yeni bir dize döndürecektir.

In [154]:

```
print(regex.sub('REDACTED', text))
```

```
Dave REDACTED  
Steve REDACTED  
Rob REDACTED  
Ryan REDACTED
```

E-posta adreslerini bulmak ve her adresi aynı anda üç bileşenine ayırmak istediginizi varsayıyalım: kullanıcı adı, alan adı ve alan soneki. Bunu yapmak için, modelin parçalanacak kısımlarının etrafına parantez koyn.

In [155]:

```
pattern = r'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})'
```

In [156]:

```
regex = re.compile(pattern, flags=re.IGNORECASE)
```

Bu değiştirilmiş normal ifade tarafından üretilen bir eşleştirme nesnesi, gruplar yöntemiyle desen bileşenlerinin bir demetini döndürür.

In [157]:

```
m = regex.match('wesm@bright.net')
```

In [158]:

```
m.groups()
```

Out[158]:

```
('wesm', 'bright', 'net')
```

`findall`, kalıpta gruplar olduğunda tuples döndürür.

In [159]:

```
regex.findall(text)
```

Out[159]:

```
[('dave', 'google', 'com'),
 ('steve', 'gmail', 'com'),
 ('rob', 'gmail', 'com'),
 ('ryan', 'yahoo', 'com')]
```

sub ayrıca \ 1 ve \ 2 gibi özel semboller kullanarak her maçtaki gruptara er işebilir. \ 1 sembolü ilk eşleşen grubu karşılık gelir, \ 2 ikinciye karşılık gelir ve bu böyle devam eder.

In [160]:

```
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
```

Python'da regular expressions'dan çok daha fazlası vardır.

In [161]:

```
Image("img/picture48.png")
```

Out[161]:

Table 7-4. Regular expression methods

Argument	Description
findall	Return all non-overlapping matching patterns in a string as a list
finditer	Like findall, but returns an iterator
match	Match pattern at start of string and optionally segment pattern components into groups; if the pattern matches, returns a match object, and otherwise None
search	Scan string for match to pattern; returning a match object if so; unlike match, the match can be anywhere in the string as opposed to only at the beginning
split	Break string into pieces at each occurrence of pattern
sub, subn	Replace all (sub) or first n occurrences (subn) of pattern in string with replacement expression; use symbols \1, \2, ... to refer to match group elements in the replacement string

Pandas'ta Vektörize String İşlevleri

Analiz için dağınık bir veri kümesini temizlemek, genellikle çok sayıda dizge işleme ve düzenleme gerektirir. Konuları karmaşıklaştırmak için, dizeler içeren bir sütunda bazen eksik veriler olabilir.

In [162]:

```
data = { 'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com',
         'Rob': 'rob@gmail.com', 'Wes': np.nan}
```

In [163]:

```
data = pd.Series(data)
```

In [164]:

```
data
```

Out[164]:

```
Dave      dave@google.com
Steve     steve@gmail.com
Rob       rob@gmail.com
Wes          None
dtype: object
```

In [165]:

```
data.isnull()
```

Out[165]:

```
Dave      False
Steve     False
Rob       False
Wes       True
dtype: bool
```

Her e-posta adresinin içinde str.contains ile 'gmail' olup olmadığını kontrol edebiliriz.

In [166]:

```
data.str.contains('gmail')
```

Out[166]:

```
Dave      False
Steve     True
Rob       True
Wes       None
dtype: object
```

IGNORECASE gibi herhangi bir re seçeneğiyle birlikte Regular expressions da kullanılabilir.

In [167]:

pattern

Out[167]:

'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})'

In [168]:

data.str.findall(pattern, flags=re.IGNORECASE)

Out[168]:

```
Dave      [(dave, google, com)]
Steve    [(steve, gmail, com)]
Rob      [(rob, gmail, com)]
Wes       NaN
dtype: object
```

Vektörize edilmiş eleman alımını yapmanın birkaç yolu vardır. Str niteliği için str.get veya index kullanılır:

In [169]:

matches = data.str.match(pattern, flags=re.IGNORECASE)

In [170]:

matches

Out[170]:

```
Dave     True
Steve   True
Rob     True
Wes      NaN
dtype: object
```

Gömülü listelerdeki ögelere erişmek için, bu işlevlerden herhangi birine bir dizin aktarabiliriz.

In [171]:

data.str[:5]

Out[171]:

```
Dave     dave@
Steve   steve
Rob     rob@g
Wes      NaN
dtype: object
```

Daha fazla pandas String metodları.

In [172]:

Image("img/picture49.png")

Out[172]:

Table 7-5. Partial listing of vectorized string methods

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 x.endswith(pattern) for each element
startswith	Equivalent to x.startswith(pattern) 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 str.isalnum
isalpha	Equivalent to built-in str.isalpha
isdecimal	Equivalent to built-in str.isdecimal
isdigit	Equivalent to built-in str.isdigit
islower	Equivalent to built-in str.islower
isnumeric	Equivalent to built-in str.isnumeric
isupper	Equivalent to built-in str.isupper
join	Join strings in each element of the Series with passed separator
len	Compute length of each string
lower, upper	Convert cases; equivalent to x.lower() or x.upper() for each element

In [173]:

Image("img/picture50.png")

Out[173]:

Method	Description
match	Use re.match 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 pad(side='both')
repeat	Duplicate values (e.g., s.str.repeat(3) is equivalent to x * 3 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

Veri Düzenleme: Ekleme, Birleştirme ve Yeniden Şekillendirme

Birçok uygulamada, veriler bir dizi dosya veya veri tabanına yayılabilir veya analiz edilmesi kolay olmayan bir biçimde düzenlenebilir.

8.1 Hiyerarşik İndeksleme

Hiyerarşik indeksleme, bir eksende birden çok (iki veya daha fazla) indeks seviyesine sahip olmanızı sağlayan önemli bir pandas özelliğidir. Biraz soyut olarak, daha düşük boyutlu bir formda daha yüksek boyutlu verilerle çalışmak için bir yol sağlar.

In [1]:

```
import pandas as pd
import numpy as np
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 [2]:

```
data
```

Out[2]:

```
a    1    -0.279672
      2    -0.376812
      3     0.966134
b    1    -1.046658
      3     1.004824
c    1    -0.637444
      2     0.356900
d    2     0.575774
      3    -0.882728
dtype: float64
```

Gördüğünüz şey, dizini MultiIndex olan bir Serinin güzel bir görünümüdür. Dizin ekranındaki "boşluklar", "doğrudan yukarıdaki etiketi kullanın" anlamına gelir.

In [3]:

```
data.index
```

Out[3]:

```
MultiIndex([( 'a', 1),
            ('a', 2),
            ('a', 3),
            ('b', 1),
            ('b', 3),
            ('c', 1),
            ('c', 2),
            ('d', 2),
            ('d', 3)],
           )
```

Hiyerarşik olarak indekslenmiş bir nesneyle, verilerin alt kümelerini kısaca seçmenize olanak tanıyan kısmi indeksleme mümkündür.

In [4]:

```
data['b']
```

Out[4]:

```
1    -1.046658
3     1.004824
dtype: float64
```

In [5]:

```
data['b':'c']
```

Out[5]:

```
b    1    -1.046658
      3     1.004824
c    1    -0.637444
      2     0.356900
dtype: float64
```

In [6]:

```
data.loc[['b', 'd']]
```

Out[6]:

```
b    1    -1.046658
      3     1.004824
d    2     0.575774
      3    -0.882728
dtype: float64
```

"İç" seviyeden bile seçim mümkündür.

In [7]:

```
data.loc[:, 2]
```

Out[7]:

```
a    -0.376812
c    0.356900
d    0.575774
dtype: float64
```

Hiyerarşik indeksleme, verileri yeniden şekillendirmede ve bir pivot tablo oluşturmak gibi grup tabanlı işlemlerde önemli bir rol oynar. Örneğin, unstack yönteminin kullanarak verileri DataFrame'e yeniden düzenleyebilirsiniz.

In [8]:

```
data.unstack()
```

Out[8]:

	1	2	3
a	-0.279672	-0.376812	0.966134
b	-1.046658	NaN	1.004824
c	-0.637444	0.356900	NaN
d	NaN	0.575774	-0.882728

Unstack işlevinin tersi stack.

In [9]:

```
data.unstack().stack()
```

Out[9]:

```
a  1    -0.279672
   2    -0.376812
   3    0.966134
b  1    -1.046658
   3    1.004824
c  1    -0.637444
   2    0.356900
d  2    0.575774
   3   -0.882728
dtype: float64
```

Bir DataFrame ile, her iki eksen hiyerarşik bir dizine sahip olabilir.

In [10]:

```
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 [11]:

frame

Out[11]:

	Ohio		Colorado
	Green	Red	Green
a	1	0	1
	2	3	4
b	1	6	7
	2	9	10
			11

Hiyerarşik seviyelerin isimleri olabilir (dizeler veya herhangi bir Python nesnesi olarak), bunlar konsol çıktısında görünecektir.

In [12]:

frame.index.names = ['key1', 'key2']

In [13]:

frame.columns.names = ['state', 'color']

In [14]:

frame

Out[14]:

	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

Kısmi sütun indeksleme ile benzer şekilde sütun gruplarını seçebilirsiniz.

In [15]:

```
frame['Ohio']
```

Out[15]:

	color	Green	Red
key1	key2		
a	1	0	1
	2	3	4
b	1	6	7
	2	9	10

Bir MultiIndex kendi başına oluşturulabilir ve daha sonra yeniden kullanılabilir; Önceki DataFrame'deki düzey adlarıyla sütunlar şu şekilde oluşturulabilir:

```
MultiIndex.from_arrays([['Ohio', 'Ohio', 'Colorado'], ['Green', 'Red', 'Green']],
                       names=['state', 'color'])
```

Reordering ve Seviyeleri Sıralama

Bazen, bir eksendeki seviyelerin sırasını yeniden düzenlemeniz veya verileri belirli bir seviyedeki değerlere göre sıralamanız gerekecektir. Takas seviyesi, iki seviye numarası veya adı alır ve seviyeleri değiştirilmiş yeni bir nesne döndürür (ancak veriler başka türlü değiştirilmez).

In [16]:

```
frame.swaplevel('key1', 'key2')
```

Out[16]:

state	Ohio	Colorado	
color	Green	Red	Green
key2	key1		
1	a	0	1
2	a	3	4
1	b	6	7
2	b	9	10
			11

sort_index ise verileri yalnızca tek bir düzeydeki değerleri kullanarak sıralar.

lar. Düzeyleri değiştirirken, sonucun sözlübilimsel olarak belirtilen düzeye göre sıralanması için `sort_index`'in kullanılması da nadir değildir.

In [17]:

```
frame.sort_index(level=1)
```

Out[17]:

	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 [18]:

```
frame.swaplevel(0, 1).sort_index(level=0)
```

Out[18]:

	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 birçok tanımlayıcı ve özet istatistik, belirli bir eksende toplamak istediğiniz seviyeyi belirtebileceğiniz bir seviye seçeneğine sahiptir. Yukarıdaki DataFrame'i düşünün; aşağıdaki gibi satırlarda veya sütunlarda seviyeye göre toplayabiliriz:

In [19]:

```
frame.sum(level='key2')
```

Out[19]:

state	Ohio	Colorado	
color	Green	Red	Green
key2			
1	6	8	10
2	12	14	16

In [20]:

```
frame.sum(level='color', axis=1)
```

Out[20]:

	color	Green	Red
key1	key2		
a	1	2	1
	2	8	4
b	1	14	7
	2	20	10

DataFrame'in sütunlarıyla dizine ekleme

Satır dizini olarak bir DataFrame'den bir veya daha fazla sütun kullanmak istemeniz alışılmadık bir durum değildir; alternatif olarak, satır dizinini DataFrame'in sütunlarına taşımak isteyebilirsiniz. Aşağıda bir DataFrame örneği verilmiştir.

In [21]:

```
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 [22]:

frame

Out[22]:

	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 fonksiyonu, dizin olarak bir veya daha fazla sütununu kullanarak yeni bir DataFrame oluşturur.

In [23]:

frame2 = frame.set_index(['c', 'd'])

In [24]:

frame2

Out[24]:

	a	b
0	0	7
one	1	6
2	2	5
0	3	4
1	4	3
two	5	2
3	6	1

Varsayılan olarak sütunlar DataFrame'den kaldırılır, ancak bunları içinde bırakabilirsiniz.

In [25]:

```
frame.set_index(['c', 'd'], drop=False)
```

Out[25]:

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

`reset_index` ise `set_index`'in tersini yapar; hiyerarşik dizin seviyeleri sütunlara taşınır.

In [26]:

```
frame2.reset_index()
```

Out[26]:

	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	

8.2 Combining and Merging Datasets

Pandas nesnelerinde bulunan veriler, birkaç yolla bir araya getirilebilir:

- `pandas.merge`, DataFrame'lerdeki satırları bir veya daha fazla anahtara göre bağlar. Bu, veritabanı birleştirme işlemlerini gerçekleştirir
- `pandas.concat`, bir eksen boyunca nesneleri birleştirir veya "yığınlar".
- `comb_first` örnek yöntemi, bir nesnedeki eksik değerleri diğerinden gelen değerlerle doldurmak için çakışan verileri birbirine eklemeyi sağlar.

Veritabanı Tarzı DataFrame Birleştirme

Merge veya join işlemleri, bir veya daha fazla anahtar kullanarak satırları birbirine bağlayarak veri kümelerini birleştirir. Bu işlemler ilişkisel veritabanları için merkezidir.

In [27]:

```
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})
```

In [28]:

```
df2 = pd.DataFrame({'key': ['a', 'b', 'd'],
                    'data2': range(3)})
```

In [29]:

```
df1
```

Out[29]:

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	a	5
6	b	6

In [30]:

```
df2
```

Out[30]:

	key	data2
0	a	0
1	b	1
2	d	2

Bu, bire çok birleşim örneğidir; df1'deki verilerin a ve b olarak etiketlenmiş birden çok satırı varken, df2'nin anahtar sütunundaki her değer için yalnızca bir satırı vardır.

In [31]:

```
pd.merge(df1, df2)
```

Out[31]:

	key	data1	data2
0	b	0	1
1	b	1	1
2	b	6	1
3	a	2	0
4	a	4	0
5	a	5	0

Hangi sütuna join yapılacağını belirtmeyi unutmayın. Bu bilgi belirtilmemiş e, join, anahtar olarak üst üste binen sütun adlarını kullanır.

In [32]:

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

Out[32]:

	key	data1	data2
0	b	0	1
1	b	1	1
2	b	6	1
3	a	2	0
4	a	4	0
5	a	5	0

Her nesnede sütun adları farklıysa, bunları ayrı ayrı belirtebilirsiniz.

In [33]:

```
df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})
```

In [34]:

```
df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'],
                    'data2': range(3)})
```

In [35]:

```
pd.merge(df3, df4, left_on='lkey', right_on='rkey')
```

Out[35]:

	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

Sonuçta 'c' ve 'd' değerlerinin ve ilişkili verilerin eksik olduğunu fark edebilirsiniz. Varsayılan olarak join, bir 'iç' birleştirme yapar; sonuktaki anahatlar kesişim veya her iki tabloda da bulunan ortak kümedir. Diğer olası seçenekler "sol", "sağ" ve "diş" tır. Dış join, hem sol hem de sağ birleştirmelerin uygulanmasının etkisini birleştirmek tuşların birleşimini alır.

In [36]:

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

Out[36]:

	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

In [37]:

```
from IPython.display import Image
Image("img/picture51.png")
```

Out[37]:

Table 8-1. Different join types with how argument

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

Çoktan çoğa birleştirme, sezgisel olmasa da iyi tanımlanmış bir davranışa sahiptir. İşte bir örnek:

In [38]:

```
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                    'data1': range(6)})
```

In [39]:

```
df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],
                    'data2': range(5)})
```

In [40]:

```
df1
```

Out[40]:

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	b	5

In [41]:

```
df2
```

Out[41]:

	key	data2
0	a	0
1	b	1
2	a	2
3	b	3
4	d	4

In [42]:

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

Out[42]:

	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

Çoktan çoğa birleşimler, satırların Kartezyen çarpımını oluşturur. Soldaki DataFrame'de üç 'b' satırı ve sağdaki iki 'b' satırı olduğu için, sonuçta altı 'b' satırı vardır. Birleştirme yöntemi yalnızca sonuçta görünen farklı anahtar değerlerini etkiler.

In [43]:

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

Out[43]:

	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

Birden çok anahtarla birleştirmek için sütun adları listesini iletin.

In [44]:

```
left = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
                     'key2': ['one', 'two', 'one'],
                     'lval': [1, 2, 3]})
```

In [45]:

```
right = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
                      'key2': ['one', 'one', 'one', 'two'],
                      'rval': [4, 5, 6, 7]})
```

In [46]:

```
pd.merge(left, right, on=['key1', 'key2'], how='outer')
```

Out[46]:

	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 yönteminin seçimine bağlı olarak sonuçta hangi tuş kombinasyonları görüneneğini belirlemek için, çoklu anahtarları tek bir birleştirme anahtarı olarak kullanılabilecek bir tuple dizisi oluşturuyor olarak düşünün (aslında bu şekilde uygulanmamış olsa bile).

Birleştirme işlemlerinde dikkate alınması gereken son bir konu, çakışan sütun adlarının işlenmesidir. Çakışmayı manuel olarak ele alabilirseniz de (ekseni etiketlerini yeniden adlandırmaya ilgili önceki bölüme bakın), birleştirime, sol ve sağ DataFrame nesnelerindeki çakışan adlara eklenecek dizeleri belirtmek için bir sonek seçeneğine sahiptir.

In [47]:

```
pd.merge(left, right, on='key1')
```

Out[47]:

	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 [48]:

```
pd.merge(left, right, on='key1', suffixes=('_left', '_right'))
```

Out[48]:

	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

In [49]:

```
Image("img/picture52.png")
```

Out[49]:

Table 8-2. *merge* function arguments

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; True 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 False, 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.

Endekste Birleştirme

Bazı durumlarda, DataFrame'deki birleştirme anahtar (lar) ı, dizininde bulunur. Bu durumda, dizinin birleştirme anahtarı olarak kullanılması gerektiğini belirtmek için `left_index = True` veya `right_index = True` (veya her ikisini de) iletебилиrsiniz.

In [50]:

```
import pandas as pd
import numpy as np
left1 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'],
                      'value': range(6)})
```

In [51]:

```
right1 = pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
```

In [52]:

```
left1
```

Out[52]:

	key	value
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

In [53]:

```
right1
```

Out[53]:

	group_val
a	3.5
b	7.0

In [54]:

```
pd.merge(left1, right1, left_on='key', right_index=True)
```

Out[54]:

	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

Varsayılan birleştirme yöntemi birleştirme anahtarlarını kesiştirmek olduğundan, bunun yerine bunların birleşimini bir dış birleştirmeyle oluşturabilirsiniz.

In [55]:

```
pd.merge(left1, right1, left_on='key', right_index=True, how='outer')
```

Out[55]:

	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

Hiyerarşik olarak dizine alınmış verilerde, dizine birleştirme örtük olarak çok anahtarlı bir birleştirme olduğu için işler daha karmaşıktır.

In [56]:

```
lefth = pd.DataFrame({'key1': ['Ohio', 'Ohio', 'Ohio',
                               'Nevada', 'Nevada'],
                      'key2': [2000, 2001, 2002, 2001, 2002],
                      'data': np.arange(5.)})
```

In [57]:

```
righth = 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 [58]:

lefth

Out[58]:

	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 [59]:

righth

Out[59]:

		event1	event2
Nevada	2001	0	1
	2000	2	3
	2000	4	5
Ohio	2000	6	7
	2001	8	9
	2002	10	11

Bu durumda, bir liste olarak birleştirilecek birden çok sütunu belirtmeniz gereklidir (how='outer' ile yinelenen dizin değerlerinin işlenmesine dikkat edin).

In [60]:

pd.merge(left, righth, left_on=['key1', 'key2'], right_index=True)

Out[60]:

	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 [61]:

```
pd.merge(lefth, righth, left_on=['key1', 'key2'],
         right_index=True, how='outer')
```

Out[61]:

	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 dizinlerini kullanmak da mümkündür.

In [62]:

```
left2 = pd.DataFrame([[1., 2.], [3., 4.], [5., 6.]],
                     index=['a', 'c', 'e'],
                     columns=['Ohio', 'Nevada'])
```

In [63]:

```
right2 = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [13., 14.]],
                      index=['b', 'c', 'd', 'e'],
                      columns=['Missouri', 'Alabama'])
```

In [64]:

```
left2
```

Out[64]:

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

In [65]:

right2

Out[65]:

	Missouri	Alabama
b	7.0	8.0
c	9.0	10.0
d	11.0	12.0
e	13.0	14.0

In [66]:

pd.merge(left2, right2, how='outer', left_index=True, right_index=True)

Out[66]:

	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

DataFrame, dizine göre birleştirme için uygun bir birleştirme örneğine sahip tir. Aynı veya benzer dizinlere sahip ancak üst üste binmeyen sütunlara sahi p birçok DataFrame nesnesini birleştirmek için de kullanılabilir. Önceki örn ekte şunları yazabilirdik:

In [67]:

left2.join(right2, how='outer')

Out[67]:

	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

Kısmen eski nedenlerden dolayı, DataFrame'in birleştirme yöntemi, sol çerçevelenin satır dizinini tam olarak koruyarak birleştirme anahtarlarında bir sol birleştirme gerçekleştirir. Ayrıca, çağrılan DataFrame'in sütunlarından birinde geçirilen DataFrame'in dizinine katılmayı da destekler.

In [68]:

```
left1.join(right1, on='key')
```

Out[68]:

	key	value	group_val
0	a	0	3.5
1	b	1	7.0
2	a	2	3.5
3	a	3	3.5
4	b	4	7.0
5	c	5	NaN

Son olarak, basit dizinde dizin birleştirmeleri için, bir sonraki bölümde açılan daha genel concat işlevini kullanmaya alternatif olarak birleştirilecek DataFrame'lerin bir listesini iletebilirsiniz.

In [69]:

```
another = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [16., 17.]],
                       index=['a', 'c', 'e', 'f'],
                       columns=['New York', 'Oregon'])
```

In [70]:

another

Out[70]:

	New York	Oregon
a	7.0	8.0
c	9.0	10.0
e	11.0	12.0
f	16.0	17.0

In [71]:

```
left2.join([right2, another])
```

Out[71]:

	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 [72]:

```
left2.join([right2, another], how='outer')
```

Out[72]:

	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

Başka bir tür veri birleştirme işlemi, birbirinin yerine birleştirme, bağlam veya yığınlama olarak adlandırılır. NumPy'nin birleştirme işlevi bunu NumPy dizileriyle yapabilir.

In [73]:

```
arr = np.arange(12).reshape((3, 4))
```

In [74]:

```
arr
```

Out[74]:

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

In [75]:

```
np.concatenate([arr, arr], axis=1)
```

Out[75]:

```
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]])
```

Series ve DataFrame gibi panda nesneleri bağlamında, etiketlenmiş eksenlere sahip olmak, dizi birleştirmeyi daha da genelleştirmenize olanak tanır. Özellikle, düşünmeniz gereken birçok ek şey var:

- Nesneler diğer eksenlerde farklı şekilde indeksleniyorsa, bu eksenlerdeki farklı öğeleri birleştirmeli miyiz yoksa sadece paylaşılan değerleri mi kullanmalıyız (kesisim)?

- Birleştirilmiş veri parçalarının ortaya çıkan nesnede tanımlanabilir olması gerekiyor mu?
- "Birleştirme ekseni" korunması gereken verileri içeriyor mu? Çoğu durumda, bir DataFrame'deki varsayılan tamsayı etiketleri en iyi şekilde birleştirme sırasında atılır.

Pandas'da concat işlevi, bu endişelerin her birini ele almak için tutarlı bir yol sağlar. Nasıl çalıştığını göstermek için birkaç örnek aşağıdadır. indeks çakışması olmayan üç Serimiz olduğunu varsayalım:

In [76]:

```
import pandas as pd
import numpy as np
s1 = pd.Series([0, 1], index=['a', 'b'])
```

In [77]:

```
s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])
```

In [78]:

```
s3 = pd.Series([5, 6], index=['f', 'g'])
```

Bir listedeki bu nesnelerle concat çağrılması, değerleri ve dizinleri birbirine yapıştırır.

In [79]:

```
pd.concat([s1, s2, s3])
```

Out[79]:

```
a    0
b    1
c    2
d    3
e    4
f    5
g    6
dtype: int64
```

Varsayılan olarak concat, eksen = 0 boyunca çalışır ve başka bir Seri oluşturur. Eksen = 1'i geçirirseniz, sonuç bunun yerine bir DataFrame olacaktır (eksen = 1 sütunlardır).

In [80]:

```
pd.concat([s1, s2, s3], axis=1)
```

Out[80]:

	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

Bu durumda, diğer eksende hiçbir örtüşme olmaz, gördüğünüz gibi dizinlerin sıralı birleşimi ('dış' birleşim). Bunun yerine join = 'interior' öğesini geçerek onlarla kesişebilirsiniz.

In [81]:

```
s4 = pd.concat([s1, s3])
```

In [82]:

```
s4
```

Out[82]:

```
a      0
b      1
f      5
g      6
dtype: int64
```

In [83]:

```
pd.concat([s1, s4], axis=1)
```

Out[83]:

	0	1
a	0.0	0
b	1.0	1
f	NaN	5
g	NaN	6

In [84]:

```
pd.concat([s1, s4], axis=1, join='inner')
```

Out[84]:

	0	1
a	0	0
b	1	1

Bu son örnekte, `join='inner'` seçeneği nedeniyle 'f' ve 'g' etiketleri kayboldu.

`Join_axes` ile diğer eksenlerde kullanılacak eksenleri bile belirtebilirsiniz.

```
pd.concat([s1, s4], axis=1, join_axes=[['a', 'c', 'b', 'e']])
```

In [86]:

```
result = pd.concat([s1, s1, s3], keys=['one', 'two', 'three'])
```

In [87]:

```
result
```

Out[87]:

```
one    a    0
      b    1
two    a    0
      b    1
three   f    5
        g    6
dtype: int64
```

In [88]:

```
result.unstack()
```

Out[88]:

	a	b	f	g
one	0.0	1.0	NaN	NaN
two	0.0	1.0	NaN	NaN
three	NaN	NaN	5.0	6.0

`Series'in eksen = 1 boyunca birleştirilmesi durumunda, anahtarlar DataFrame sütun başlıklarını olur.`

In [89]:

```
pd.concat([s1, s2, s3], axis=1, keys=['one', 'two', 'three'])
```

Out[89]:

	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 uzanır.

In [90]:

```
df1 = pd.DataFrame(np.arange(6).reshape(3, 2), index=['a', 'b', 'c'],
                   columns=['one', 'two'])
```

In [91]:

```
df2 = pd.DataFrame(5 + np.arange(4).reshape(2, 2), index=['a', 'c'],
                   columns=['three', 'four'])
```

In [92]:

```
df1
```

Out[92]:

	one	two
a	0	1
b	2	3
c	4	5

In [93]:

df2

Out[93]:

	three	four
a	5	6
c	7	8

In [94]:

pd.concat([df1, df2], axis=1, keys=['level1', 'level2'])

Out[94]:

	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

Liste yerine bir nesne diktesini iletirseniz, diktenin anahtarları dict's seçeneği için kullanılır.

In [95]:

pd.concat({'level1': df1, 'level2': df2}, axis=1)

Out[95]:

	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 [96]:

pd.concat([df1, df2], axis=1, keys=['level1', 'level2'], names=['upper', 'lower'])

Out[96]:

	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		

Son bir husus, satır dizininin herhangi bir ilgili veri içermediği DataFrame s ile ilgilidir.

In [97]:

```
df1 = pd.DataFrame(np.random.randn(3, 4), columns=['a', 'b', 'c', 'd'])
```

In [98]:

```
df2 = pd.DataFrame(np.random.randn(2, 3), columns=['b', 'd', 'a'])
```

In [99]:

```
df1
```

Out[99]:

	a	b	c	d
0	-0.934785	1.006070	-0.200388	-0.901305
1	0.369793	0.717820	-0.559990	-1.231826
2	-1.291572	-2.359808	0.632185	-0.453443

In [100]:

```
df2
```

Out[100]:

	b	d	a
0	0.295543	3.139428	-0.433797
1	-0.216733	0.845583	1.223680

Bu durumda, ignore_index = True yazabilirsiniz.

In [101]:

```
pd.concat([df1, df2], ignore_index=True)
```

Out[101]:

	a	b	c	d
0	-0.934785	1.006070	-0.200388	-0.901305
1	0.369793	0.717820	-0.559990	-1.231826
2	-1.291572	-2.359808	0.632185	-0.453443
3	-0.433797	0.295543	NaN	3.139428
4	1.223680	-0.216733	NaN	0.845583

In [102]:

```
from IPython.display import Image
Image("img/picture53.png")
```

Out[102]:

Table 8-3. concat function arguments

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 Örtüşme ile Birleştirme

Birleştirme veya birleştirme işlemi olarak ifade edilemeyen başka bir veri birleştirme durumu vardır. Dizinleri tamamen veya kısmen çakışan iki veri küməniz olabilir. Motive edici bir örnek olarak, if-else ifadesinin dizi odaklı eşdeğerini gerçekleştiren NumPy'nin where işlevini düşünün.

In [103]:

```
a = pd.Series([np.nan, 2.5, np.nan, 3.5, 4.5, np.nan],
              index=['f', 'e', 'd', 'c', 'b', 'a'])
```

In [104]:

```
b = pd.Series(np.arange(len(a), dtype=np.float64),
              index=['f', 'e', 'd', 'c', 'b', 'a'])
```

In [105]:

```
b[-1] = np.nan
```

In [106]:

```
a
```

Out[106]:

```
f      NaN  
e      2.5  
d      NaN  
c      3.5  
b      4.5  
a      NaN  
dtype: float64
```

In [107]:

```
b
```

Out[107]:

```
f      0.0  
e      1.0  
d      2.0  
c      3.0  
b      4.0  
a      NaN  
dtype: float64
```

In [108]:

```
np.where(pd.isnull(a), b, a)
```

Out[108]:

```
array([0., 2.5, 2., 3.5, 4.5, nan])
```

Seri, pandaların olağan veri hizalama mantığıyla birlikte bu işlemin eşdeğerini gerçekleştiren bir `birleştirme_first` yöntemine sahiptir.

In [109]:

```
b[:-2].combine_first(a[2:])
```

Out[109]:

```
a      NaN  
b      4.5  
c      3.0  
d      2.0  
e      1.0  
f      0.0  
dtype: float64
```

DataFrames ile, `comb_first` aynı şeyi sütun yapar, böylece bunu, ilettiğiniz nesneden gelen verilerle çağırılan nesnedeki eksik verilere "yama uygula mak" olarak düşünebilirsiniz.

In [110]:

```
df1 = pd.DataFrame({'a': [1., np.nan, 5., np.nan],
                    'b': [np.nan, 2., np.nan, 6.],
                    'c': range(2, 18, 4)})
```

In [111]:

```
df2 = pd.DataFrame({'a': [5., 4., np.nan, 3., 7.],
                    'b': [np.nan, 3., 4., 6., 8.]})
```

In [112]:

df1

Out[112]:

	a	b	c
0	1.0	NaN	2
1	NaN	2.0	6
2	5.0	NaN	10
3	NaN	6.0	14

In [113]:

df2

Out[113]:

	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 [114]:

df1.combine_first(df2)

Out[114]:

	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

8.3 Yeniden Şekillendirme ve Döndürme

Tablo şeklindeki verileri yeniden düzenlemek için bir dizi temel işlem vardır. Bunlara alternatif olarak yeniden şekillendirme veya döndürme işlemleri adı verilir.

Reshaping with Hierarchical Indexing

Hiyerarşik indeksleme, bir DataFrame'deki verileri yeniden düzenlemek için tutarlı bir yol sağlar. İki temel eylem vardır:

`stack`:

Bu, verilerdeki sütunlardan satırlara "döner" veya döner

`unstack`:

Bu, satırlardan sütunlara döner

Bu işlemleri bir dizi örnekle açıklayalım. Satır ve sütun indeksleri olarak dizi dizileri içeren küçük bir Veri Çerçevesi düşünün:

In [115]:

```
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 [116]:

```
data
```

Out[116]:

	number	one	two	three
state				
Ohio	0	1	2	
Colorado	3	4	5	

Bu verilerde yoğun yöntemini kullanmak, sütunları satırlara döndürerek bir Seri oluşturur.

In [117]:

```
result = data.stack()
```

In [118]:

result

Out[118]:

state	number
Ohio	one 0
	two 1
	three 2
Colorado	one 3
	two 4
	three 5

dtype: int64

Hiyerarşik olarak indekslenmiş bir Seriden, verileri tekrar Veri olarak yeniden düzenleyebilirsiniz. Frame with unstack:

In [119]:

result.unstack()

Out[119]:

state	number	one	two	three
Ohio	0	1	2	
Colorado	3	4	5	

Varsayılan olarak en içteki seviye istiflenmemiş durumdadır (yığınla aynıdır). Bir seviye numarası veya adı geçerek farklı bir seviyeyi kaldırabilirsiniz.

In [120]:

result.unstack(0)

Out[120]:

state	Ohio	Colorado
number		
one	0	3
two	1	4
three	2	5

In [121]:

```
result.unstack('state')
```

Out[121]:

```
state  Ohio  Colorado
```

```
number
```

one	0	3
two	1	4
three	2	5

Seviyedeki tüm değerler alt grupların her birinde bulunmazsa, yiğinin kaldırılması eksik verileri ortaya çıkarabilir.

In [122]:

```
s1 = pd.Series([0, 1, 2, 3], index=['a', 'b', 'c', 'd'])
```

In [123]:

```
s2 = pd.Series([4, 5, 6], index=['c', 'd', 'e'])
```

In [124]:

```
data2 = pd.concat([s1, s2], keys=['one', 'two'])
```

In [125]:

```
data2
```

Out[125]:

```
one   a    0
      b    1
      c    2
      d    3
two   c    4
      d    5
      e    6
dtype: int64
```

In [126]:

data2.unstack()

Out[126]:

	a	b	c	d	e
one	0.0	1.0	2.0	3.0	NaN
two	NaN	NaN	4.0	5.0	6.0

Yığınlama, eksik verileri varsayılan olarak filtreler, böylece işlem daha kolay tersine çevrilebilir.

In [127]:

data2.unstack()

Out[127]:

	a	b	c	d	e
one	0.0	1.0	2.0	3.0	NaN
two	NaN	NaN	4.0	5.0	6.0

In [128]:

data2.unstack().stack()

Out[128]:

```
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 [129]:

data2.unstack().stack(dropna=False)

Out[129]:

```
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 yığınlamayı kaldırığınızda, yığınlanmamış düzey en düşük düzey olur

sonuç:

In [130]:

```
df = pd.DataFrame({'left': result, 'right': result + 5},
                  columns=pd.Index(['left', 'right'], name='side'))
```

In [131]:

```
df
```

Out[131]:

	side	left	right
state	number		
	one	0	5
Ohio	two	1	6
	three	2	7
	one	3	8
Colorado	two	4	9
	three	5	10

In [132]:

```
df.unstack('state')
```

Out[132]:

	side	left		right	
state	Ohio	Colorado	Ohio	Colorado	
number					
one	0	3	5	8	
two	1	4	6	9	
three	2	5	7	10	

Yığını çağırırken, yığılacak eksenin adını belirtebiliriz.

In [133]:

```
df.unstack('state').stack('side')
```

Out[133]:

	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

Pivoting “Long” to “Wide” Format

Veritabanlarında ve CSV'de birden çok zaman serisini depolamanın yaygın bir yolu, uzun veya yığılmış biçimdedir.

In [134]:

```
data = pd.read_csv('examples/macrodata.csv')
```

In [135]:

```
data.head()
```

Out[135]:

	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbillrate	unemp
0	1959.0	1.0	2710.349	1707.4	286.898	470.045	1886.9	28.98	139.7	2.82	5.8
1	1959.0	2.0	2778.801	1733.7	310.859	481.301	1919.7	29.15	141.7	3.08	5.1
2	1959.0	3.0	2775.488	1751.8	289.226	491.260	1916.4	29.35	140.5	3.82	5.3
3	1959.0	4.0	2785.204	1753.7	299.356	484.052	1931.3	29.37	140.0	4.33	5.6
4	1960.0	1.0	2847.699	1770.5	331.722	462.199	1955.5	29.54	139.6	3.50	5.2

In [136]:

```
periods = pd.PeriodIndex(year=data.year, quarter=data.quarter,
                         name='date')
```

In [137]:

```
columns = pd.Index(['realgdp', 'infl', 'unemp'], name='item')
```

In [138]:

```
data = data.reindex(columns=columns)
```

In [139]:

```
data.index = periods.to_timestamp('D', 'end')
```

In [140]:

```
ldata = data.stack().reset_index().rename(columns={0: 'value'})
```

Şimdi, ldata şöyle görünür:

In [141]:

```
ldata[:10]
```

Out[141]:

		date	item	value
0	1959-03-31 23:59:59.999999999	realgdp		2710.349
1	1959-03-31 23:59:59.999999999	infl		0.000
2	1959-03-31 23:59:59.999999999	unemp		5.800
3	1959-06-30 23:59:59.999999999	realgdp		2778.801
4	1959-06-30 23:59:59.999999999	infl		2.340
5	1959-06-30 23:59:59.999999999	unemp		5.100
6	1959-09-30 23:59:59.999999999	realgdp		2775.488
7	1959-09-30 23:59:59.999999999	infl		2.740
8	1959-09-30 23:59:59.999999999	unemp		5.300
9	1959-12-31 23:59:59.999999999	realgdp		2785.204

Bu, çoklu zaman serileri veya iki veya daha fazla tuşa sahip diğer gözlemlsel veriler için sözde uzun formattır (burada, anahtarlarımız tarih ve maddedir). Tablodaki her satır tek bir gözlemi temsil eder.

Sabit bir şema (sütun adları ve veri türleri), tabloya veri eklendikçe öğe sütunundaki farklı değerlerin sayısının değişmesine izin verdiğinde, veriler MySQL gibi ilişkisel veritabanlarında sıkılıkla bu şekilde depolanır. Önceki örnekte, tarih ve öğe genellikle hem ilişkisel bütünlük hem de daha kolay birleştirilmeler sunan birincil anahtarlar olacaktır (ilişkisel veritabanı söyle). Bazı durumlarda, verilerle bu formatta çalışmak daha zor olabilir; Tarih sütunundaki zaman damgalarına göre dizine alınan her farklı öğe değer i için bir sütun içeren bir DataFrame'e sahip olmayı tercih edebilirsiniz. DataFrame'in pivot yöntemi tam olarak şu dönüşümü gerçekleştirir:

In [142]:

```
pivoted = ldata.pivot('date', 'item', 'value')
```

In [143]:

```
pivoted
```

Out[143]:

	item	infl	realgdp	unemp
	date			
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8	
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1	
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3	
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6	
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2	
...	
2008-09-30 23:59:59.999999999	-3.16	13324.600	6.0	
2008-12-31 23:59:59.999999999	-8.79	13141.920	6.9	
2009-03-31 23:59:59.999999999	0.94	12925.410	8.1	
2009-06-30 23:59:59.999999999	3.37	12901.504	9.2	
2009-09-30 23:59:59.999999999	3.56	12990.341	9.6	

203 rows × 3 columns

Aktarılan ilk iki değer sırasıyla satır ve sütun dizini olarak kullanılacak sütunlar, ardından son olarak DataFrame'i doldurmak için istege bağlı bir değer sütunudur. Aynı anda yeniden şekillendirmek istediğiniz iki değer sütununuzu olduğunu varsayıyalım

In [144]:

```
ldata['value2'] = np.random.randn(len(ldata))
```

In [145]:

ladata[:10]

Out[145]:

	date	item	value	value2
0	1959-03-31 23:59:59.999999999	realgdp	2710.349	1.194086
1	1959-03-31 23:59:59.999999999	infl	0.000	-0.197643
2	1959-03-31 23:59:59.999999999	unemp	5.800	-0.906974
3	1959-06-30 23:59:59.999999999	realgdp	2778.801	-0.207346
4	1959-06-30 23:59:59.999999999	infl	2.340	0.016003
5	1959-06-30 23:59:59.999999999	unemp	5.100	1.218104
6	1959-09-30 23:59:59.999999999	realgdp	2775.488	-0.212278
7	1959-09-30 23:59:59.999999999	infl	2.740	-1.218607
8	1959-09-30 23:59:59.999999999	unemp	5.300	-0.540738
9	1959-12-31 23:59:59.999999999	realgdp	2785.204	1.306979

Son bağımsız değişkeni atlayarak, hiyerarşik sütunlara sahip bir DataFrame elde edersiniz.

In [146]:

pivoted = ladata.pivot('date', 'item')

In [147]:

pivoted[:5]

Out[147]:

date	item	value			value2		
		infl	realgdp	unemp	infl	realgdp	unemp
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8	-0.197643	1.194086	-0.906974	
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1	0.016003	-0.207346	1.218104	
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3	-1.218607	-0.212278	-0.540738	
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6	-1.012934	1.306979	0.614581	
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2	0.437570	0.092416	1.930479	

In [148]:

```
pivoted['value'][:5]
```

Out[148]:

	item	infl	realgdp	unemp
	date			
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8	
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1	
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3	
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6	
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2	

Pivotun, `set_index` kullanarak hiyerarşik bir dizin oluşturmaya eşdeğer olduğunu ve ardından bir yığın yığını kaldırma çağrıları ile eşdeğer olduğunu unutmayın.

In [149]:

```
unstacked = ldata.set_index(['date', 'item']).unstack('item')
```

In [150]:

```
unstacked[:7]
```

Out[150]:

	value				value2		
	item	infl	realgdp	unemp	infl	realgdp	unemp
	date						
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8	-0.197643	1.194086	-0.906974	
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1	0.016003	-0.207346	1.218104	
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3	-1.218607	-0.212278	-0.540738	
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6	-1.012934	1.306979	0.614581	
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2	0.437570	0.092416	1.930479	
1960-06-30 23:59:59.999999999	0.14	2834.390	5.2	-1.879514	-1.241695	-0.271091	
1960-09-30 23:59:59.999999999	2.70	2839.022	5.6	-0.246986	-0.023288	-0.253302	

Pivoting “Wide” to “Long” Format

DataFrames için ters dönme işlemi pandas.melt'dir. Yeni bir DataFrame'de bir sütunu çok sayıda sütuna dönüştürmek yerine, birden çok sütunu tek bir sütun'a birleştirerek girdiden daha uzun bir DataFrame oluşturur. Bir örneğe bakalım:

In [151]:

```
df = pd.DataFrame({'key': ['foo', 'bar', 'baz'],
                   'A': [1, 2, 3],
                   'B': [4, 5, 6],
                   'C': [7, 8, 9]})
```

In [152]:

```
df
```

Out[152]:

	key	A	B	C
0	foo	1	4	7
1	bar	2	5	8
2	baz	3	6	9

'Anahtar' sütunu bir grup göstergesi olabilir ve diğer sütunlar veri değerleridir. Pandas.melt kullanırken, hangi sütunların (varsı) grup göstergesi olduğunu belirtmeliyiz. Buradaki tek grup göstergesi olarak 'key'i kullanalım:

In [153]:

```
melted = pd.melt(df, ['key'])
```

In [154]:

```
melted
```

Out[154]:

	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

Pivot kullanarak, orijinal düzene geri dönebiliriz.

In [155]:

```
reshaped = melted.pivot('key', 'variable', 'value')
```

In [156]:

```
reshaped
```

Out[156]:

variable	A	B	C
key			
bar	2	5	8
baz	3	6	9
foo	1	4	7

Pivotun sonucu, satır etiketleri olarak kullanılan sütundan bir dizin oluşturduğundan, verileri bir sütuna geri taşımak için reset_index'i kullanmak isteyebiliriz.

In [157]:

```
reshaped.reset_index()
```

Out[157]:

variable	key	A	B	C
0	bar	2	5	8
1	baz	3	6	9
2	foo	1	4	7

Değer sütunları olarak kullanmak için bir sütun alt kümesi de belirtebilirsiniz.

In [158]:

```
pd.melt(df, id_vars=['key'], value_vars=['A', 'B'])
```

Out[158]:

	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 [159]:

```
pd.melt(df, value_vars=['A', 'B', 'C'])
```

Out[159]:

	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 [160]:

```
pd.melt(df, value_vars=['key', 'A', 'B'])
```

Out[160]:

	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 (bazen grafikler de denir) yapmak, veri analizindeki en önemli görevlerden biridir. Keşif sürecinin bir parçası olabilir – örneğin, aykırı değerleri veya ihtiyaç duyulan veri dönüşümlerini belirlemeye yardımcı olmak veya modeller için fikir üretmenin bir yolu olarak. Diğerleri için, web için etkileşimli bir görselleştirme oluşturmak nihai hedef olabilir.

In [1]:

```
%matplotlib notebook
```

9.1 A Brief matplotlib API Primer

Matplotlib ile aşağıdaki içe aktarma kuralını kullanıyoruz:

In [2]:

```
import matplotlib.pyplot as plt
```

In [3]:

```
import numpy as np
```

In [4]:

```
data = np.arange(10)
```

In [5]:

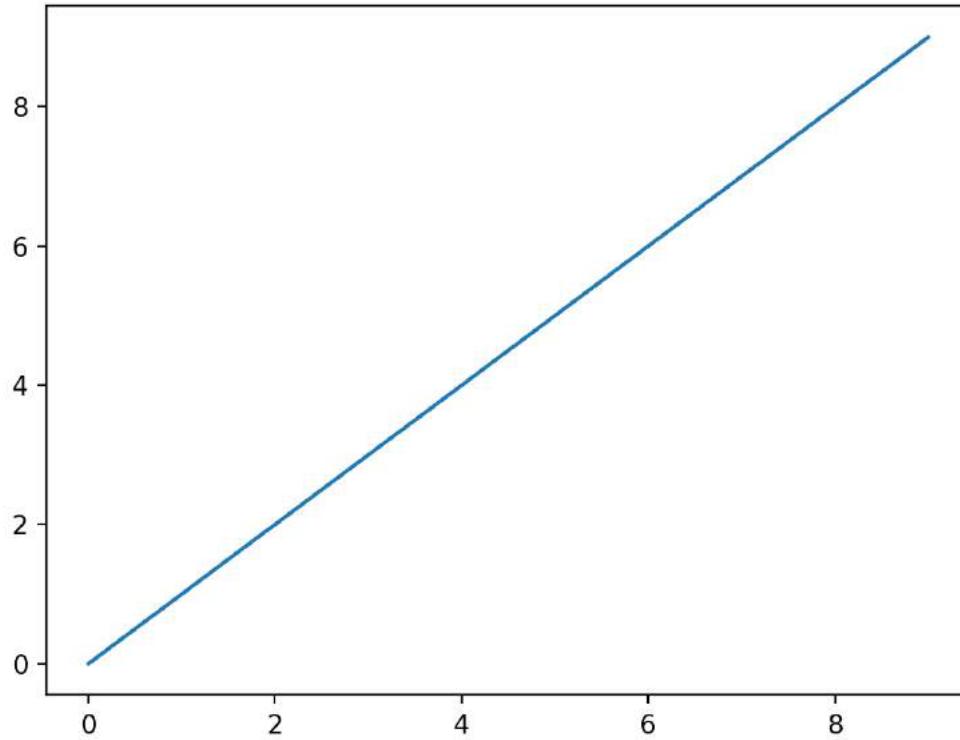
```
data
```

Out[5]:

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

In [6]:

```
plt.plot(data)
```



Out[6]:

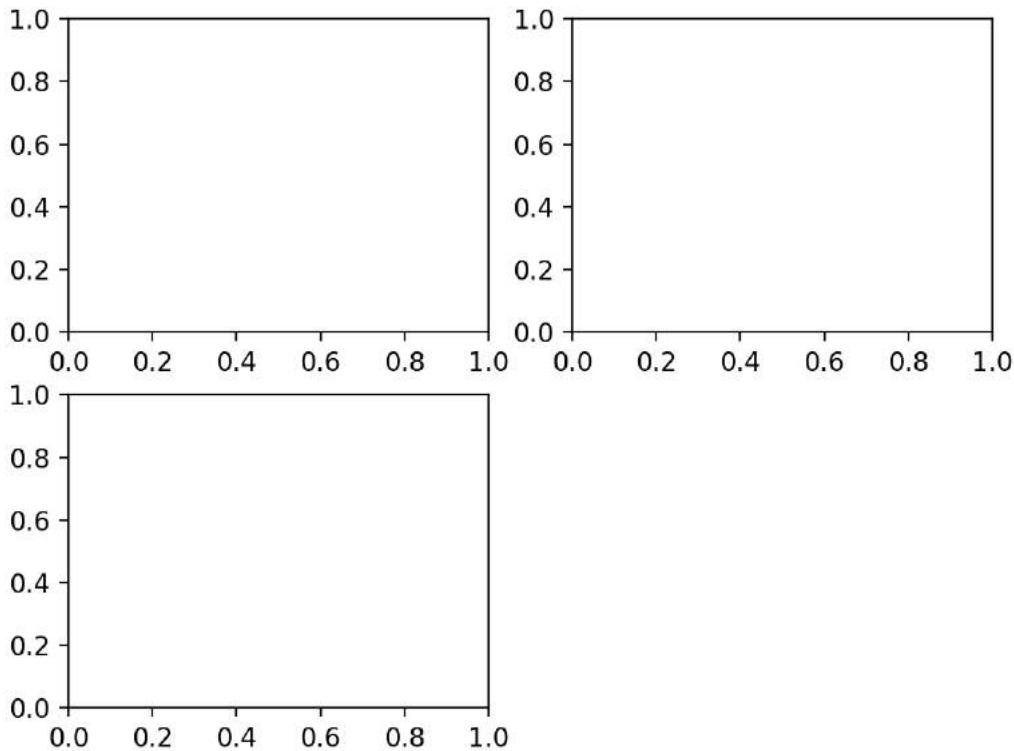
```
[<matplotlib.lines.Line2D at 0x7ff863364610>]
```

Figürler ve Alt Grafikler

Matplotlib'deki grafikler bir Figure nesnesi içinde bulunur. plt.figure ile yeni bir figür oluşturabilirsiniz:

In [7]:

```
fig = plt.figure()
```



`plt.figure` bir dizi seçeneğe sahiptir; özellikle `figsize`, diske kaydedildiği nde figürün belirli bir boyuta ve en boy oranına sahip olmasını garanti ede r.

Boş bir figürle olay örgüsü yapamazsınız. `Add_subplot` kullanarak bir veya da ha fazla alt plan oluşturmanız gereklidir.

In [8]:

```
ax1 = fig.add_subplot(2, 2, 1)
```

Bu, şeklin 2×2 olması gerekīği anlamına gelir (yani toplamda en fazla dör t grafik) ve dört alt noktadan ilkini (1'den numaralandırılmış) seçiyoruz. S onraki iki alt grafīgi oluşturursanız, Aşağıdaki görsele benzeyen bir görsel leştirme elde edersiniz.

In [9]:

```
ax2 = fig.add_subplot(2, 2, 2)
```

In [10]:

```
ax3 = fig.add_subplot(2, 2, 3)
```

In [11]:

```
from IPython.display import Image
Image("img/picture54.png")
```

Out[11]:

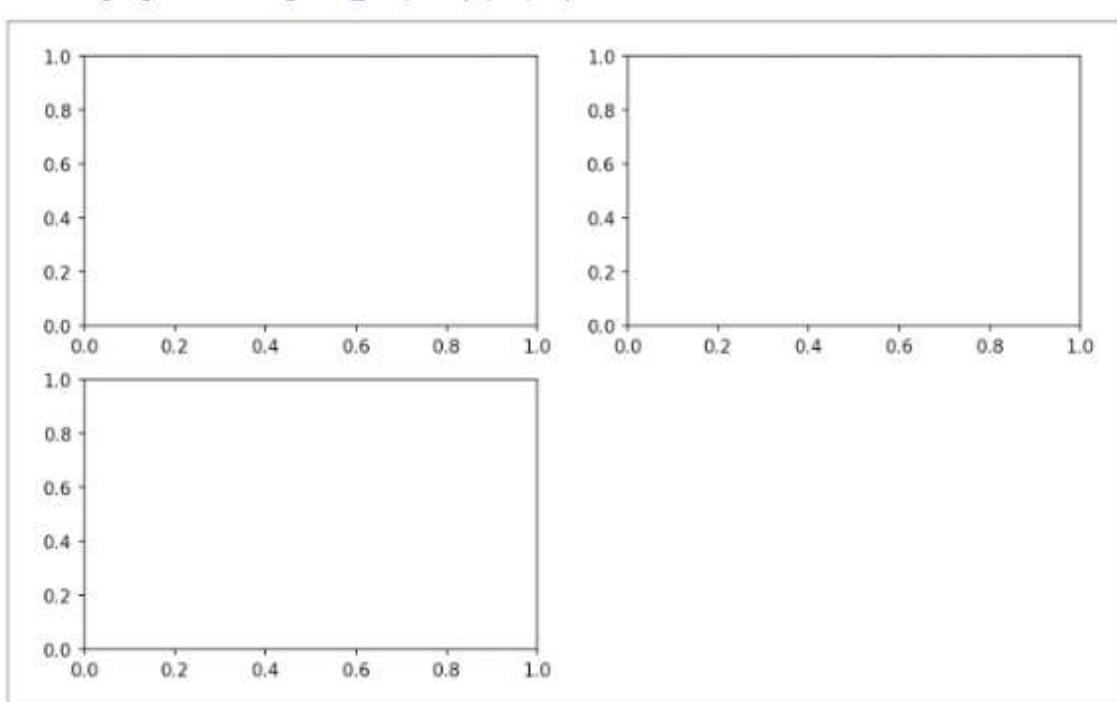
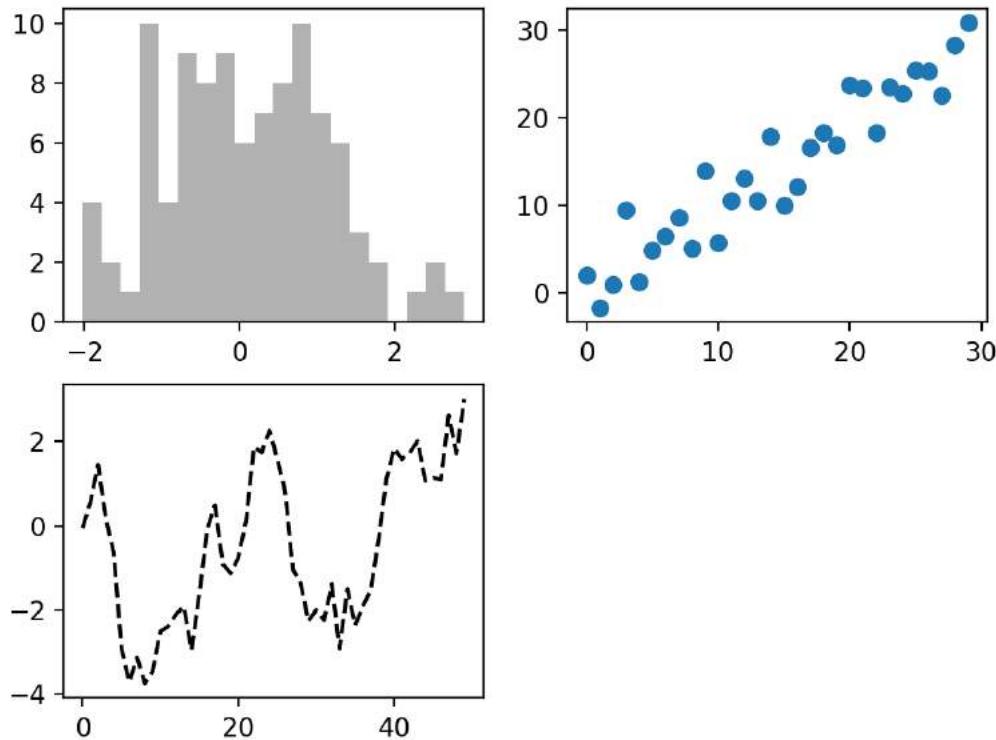


Figure 9-2. An empty matplotlib figure with three subplots

In [12]:

```
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
```



`plt.plot ([1.5, 3.5, -2, 1.6])` gibi bir çizim komutu verdığınızda, matplotlib kullanılan son şekli ve alt grafiği çizer (gerekirse bir tane oluşturur)

r), böylece şekli ve alt grafik oluşturmayı gizler . Yani aşağıdaki komutu e klersek, Aşağıdaki şekil gibi bir şey elde edersiniz.

In [13]:

```
plt.plot(np.random.randn(50).cumsum(), 'k--')
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x7ff863d73c10>]
```

'K--' matplotlib'e siyah kesikli bir çizgi çizmesini söyleyen bir stil seçenektedir. Burada fig.add_subplot tarafından döndürülen nesneler AxesSubplot nesneleridir ve bunlardan her birinin örnek yöntemini çağırarak diğer boş alt çizgilere doğrudan çizim yapabilirsiniz.

In [14]:

```
_ = ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)
```

In [15]:

```
ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```

Out[15]:

```
<matplotlib.collections.PathCollection at 0x7ff865801f70>
```

In [16]:

```
Image("img/picture55.png")
```

Out[16]:

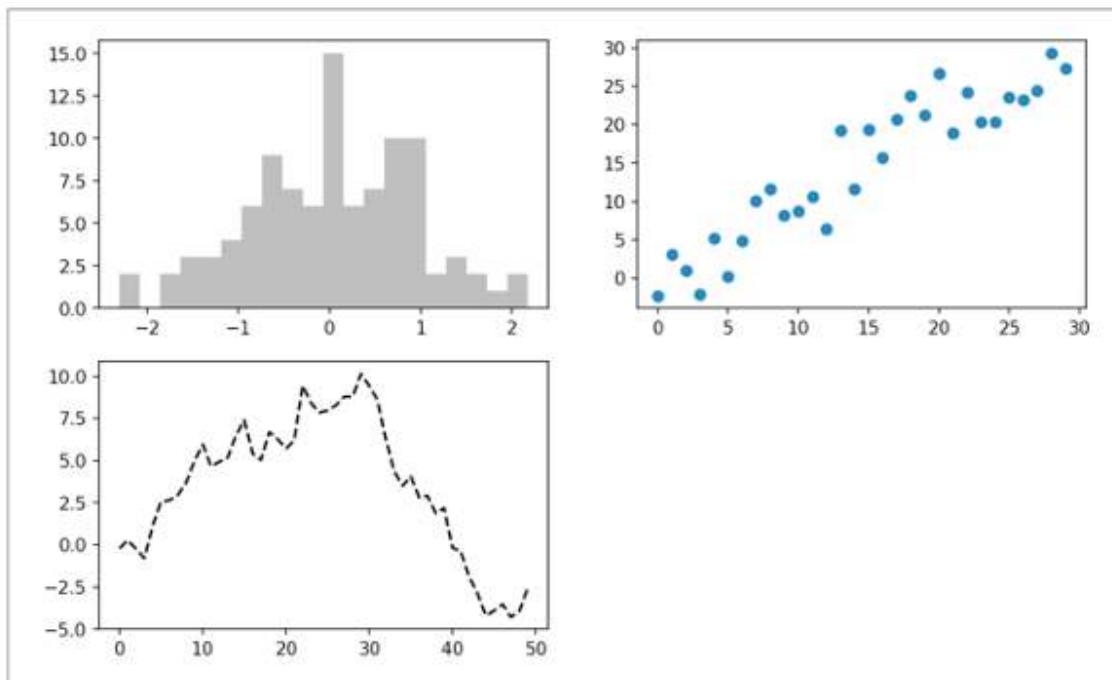
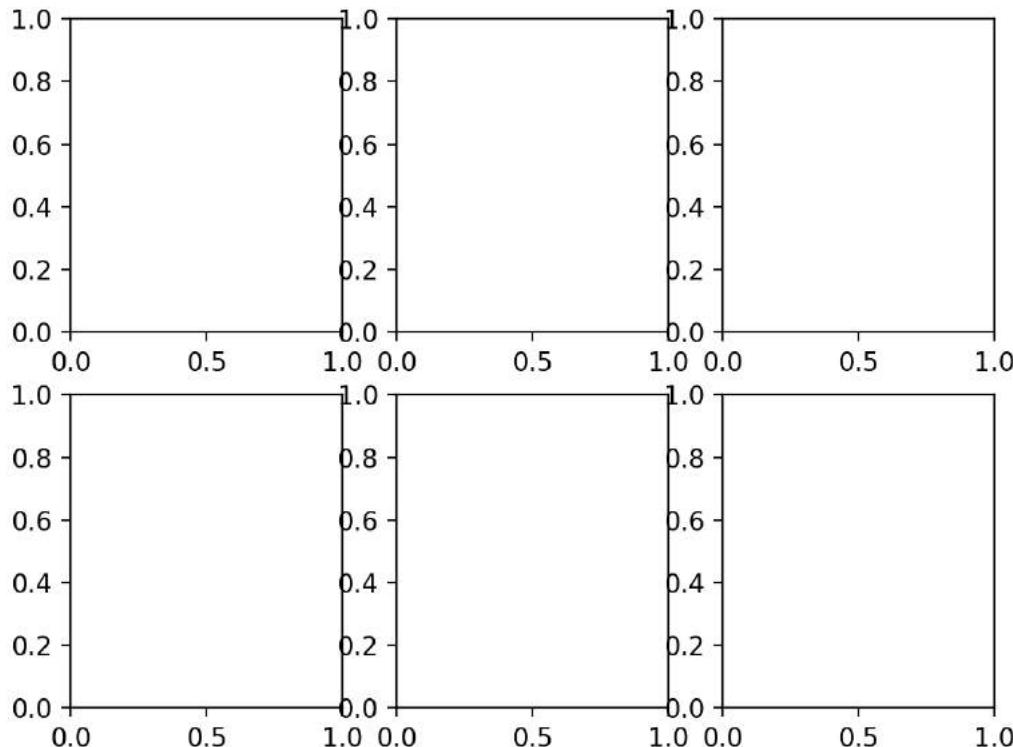


Figure 9-4. Data visualization after additional plots

Alt grafiklerden oluşan bir ızgaraya sahip bir şekil oluşturmak çok yaygın bir görevdir, bu nedenle matplotlib, yeni bir şekil oluştururan ve oluşturulan alt grafik nesnelerini içeren bir NumPy dizisi döndüren plt.subplots adlı bir uygunluk yöntemini içerir.

In [17]:

```
fig, axes = plt.subplots(2, 3)
```



In [19]:

```
axes
```

Out[19]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7ff865ce5b5>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7ff865d0c0d>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7ff865d391f>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7ff865d645e>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7ff865d8fa0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7ff865dbfee>]],
     dtype=object)
```

Axes dizisi iki boyutlu bir dizi gibi kolayca indekslenebildiği için bu çok kullanışlıdır; örneğin, eksenler [0, 1]. Ayrıca, sırasıyla sharex ve sharey

kullanarak alt grafiklerin aynı x veya y eksenine sahip olması gerektiğini d e belirtebilirsiniz. Bu, özellikle verileri aynı ölçekte karşılaştırırken ku llanışlıdır; aksi takdirde, matplotlib otomatik ölçekler sınırları bağımsız olarak çizer.

In [20]:

```
Image("img/picture56.png")
```

Out[20]:

Table 9-1. pyplot.subplots options

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 subplots are used when creating the figure, such as <code>plt.subplots(2, 2, figsize=(8, 6))</code>

Alt parseller etrafındaki boşluğu ayarlama

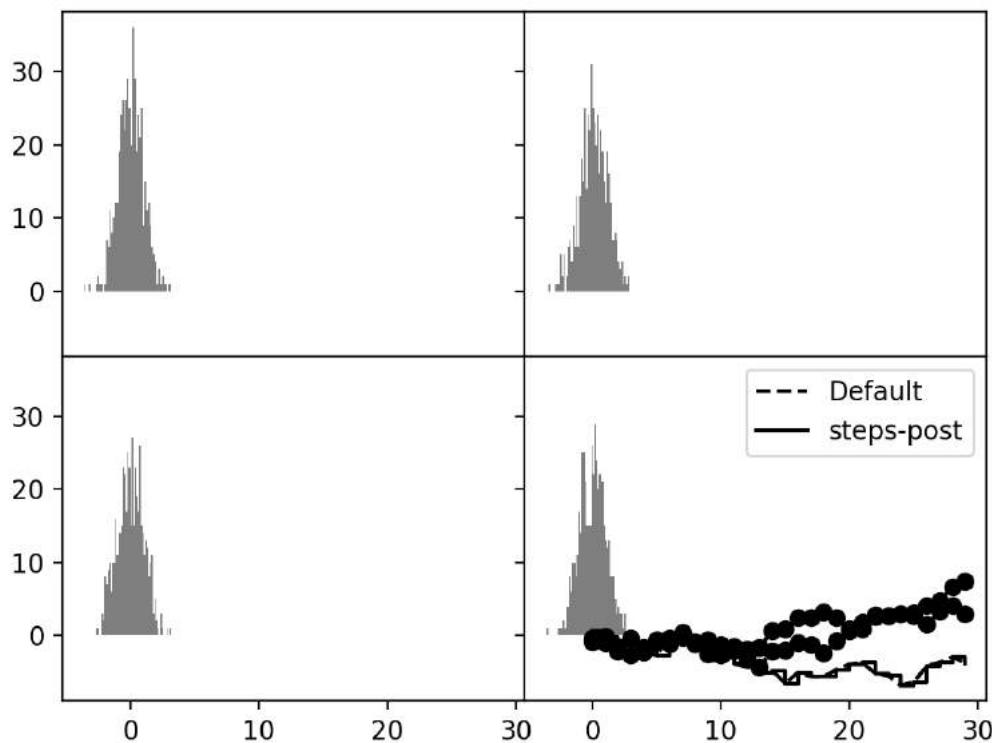
Varsayılan olarak matplotlib, alt grafiklerin dışında belirli miktarda dolgu ve alt noktalar arasında boşluk bırakır. Bu aralığın tümü, grafiğin yüksekli ğine ve genişliğine göre belirlenir, böylece grafiği grafiksel olarak veya G UI penceresini kullanarak manuel olarak yeniden boyutlandırırsanız, grafik k endini dinamik olarak ayarlayacaktır. Şekil nesnelerinde `subplots_adjust` yön temini kullanarak aralığı değiştirebilirsiniz.

```
subplots_adjust(left=None, bottom=None, right=None, top=None,
                 wspace=None, hspace=None)
```

`wspace` ve `hspace`, alt grafikler arasında boşluk olarak kullanmak için sırası yla şekil genişliği ve şekil yüksekliğinin yüzdesini kontrol eder. Burada, a ralığı sıfıra indirilen bir örnek var (bkz.Şekil 9-5).

In [21]:

```
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)
```



Eksen etiketlerinin örtüştüğünü fark edebilirsiniz. matplotlib etiketlerin örtüşüp örtüşmediğini kontrol etmez, bu nedenle böyle bir durumda, açık kene konumları ve tik etiketleri belirleyerek etiketleri kendiniz düzeltmeniz gereklidir.

Renkler, İşaretleyiciler ve Çizgi Stilleri

Matplotlib'in ana çizim işlevi, x ve y koordinatlarının dizilerini ve istege bağlı olarak renk ve çizgi stilini belirten bir dizgi kısaltmasını kabul eder. Örneğin, x'e karşı y'yi yeşil çizgilerle çizmek için şunları yaparsınız:

```
ax.plot(x, y, 'g--')
```

Bir dizede hem rengi hem de çizgi stilini belirtmenin bu yolu kolaylık sağlar; pratikte programlı olarak grafikler oluşturuyorsanız, istenen stile sahip grafikler oluşturmak için dizeleri bir araya getirmemeyi tercih edebilirsiniz. Aynı olay örgüsü şu şekilde de daha açık bir şekilde ifade edilebilirdi:

```
ax.plot(x, y, linestyle='--', color='g')
```

Çizgi grafikleri ek olarak gerçek veri noktalarını vurgulamak için işaretler e sahip olabilir. Matplotlib, noktalar arasında interpolasyon yapan sürekli bir çizgi grafiği oluşturduğundan, noktaların nerede olduğu zaman zaman belirsiz olabilir. İşaretçi, renk ve ardından işaretleyici türü ve çizgi stilin e sahip olması gereken stil dizesinin bir parçası olabilir.

In [22]:

```
from numpy.random import randn
```

In [23]:

```
plt.plot(randn(30).cumsum(), 'ko--')
```

Out[23]:

```
[<matplotlib.lines.Line2D at 0x7ff86622e580>]
```

Bu, daha açık bir şekilde şu şekilde de yazılabildi:

In [24]:

```
plt.plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
```

Out[24]:

```
[<matplotlib.lines.Line2D at 0x7ff865325e80>]
```

Çizgi çizimleri için, sonraki noktaların doğrusal olarak interpolasyonunu fark edeceksiniz.

In [25]:

```
data = np.random.randn(30).cumsum()
```

In [26]:

```
plt.plot(data, 'k--', label='Default')
```

Out[26]:

```
[<matplotlib.lines.Line2D at 0x7ff8661925b0>]
```

In [27]:

```
plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')
```

Out[27]:

```
[<matplotlib.lines.Line2D at 0x7ff8661af760>]
```

In [28]:

```
plt.legend(loc='best')
```

Out[28]:

```
<matplotlib.legend.Legend at 0x7ff866240bb0>
```

Bunu çalıştırıldığınızda <matplotlib.lines.Line2D at ...> gibi çıktıları fark edebilirsiniz. matplotlib, yeni eklenen çizim alt bileşenine başvuran nesne leri döndürür. Çoğu zaman bu çıktıyı güvenle göz ardı edebilirsiniz. Burada, plot için etiket argümanlarını aktardığımızdan, plt.legend kullanarak her satırı tanımlamak için bir çizim göstergesi oluşturabiliriz.

Ticks, Labels, ve Legends

yordamsal pyplot arayüzü (yani, matplotlib.pyplot) ve daha nesne yönelimli doğal matplotlib API'sini kullanmak.

Etkileşimli kullanım için tasarlanan pyplot arayüzü, xlim, xticks ve xticklabels gibi yöntemlerden oluşur. Bunlar sırasıyla çizim aralığını, işaret konumlarını ve işaret etiketlerini kontrol eder. İki şekilde kullanılabilirler:

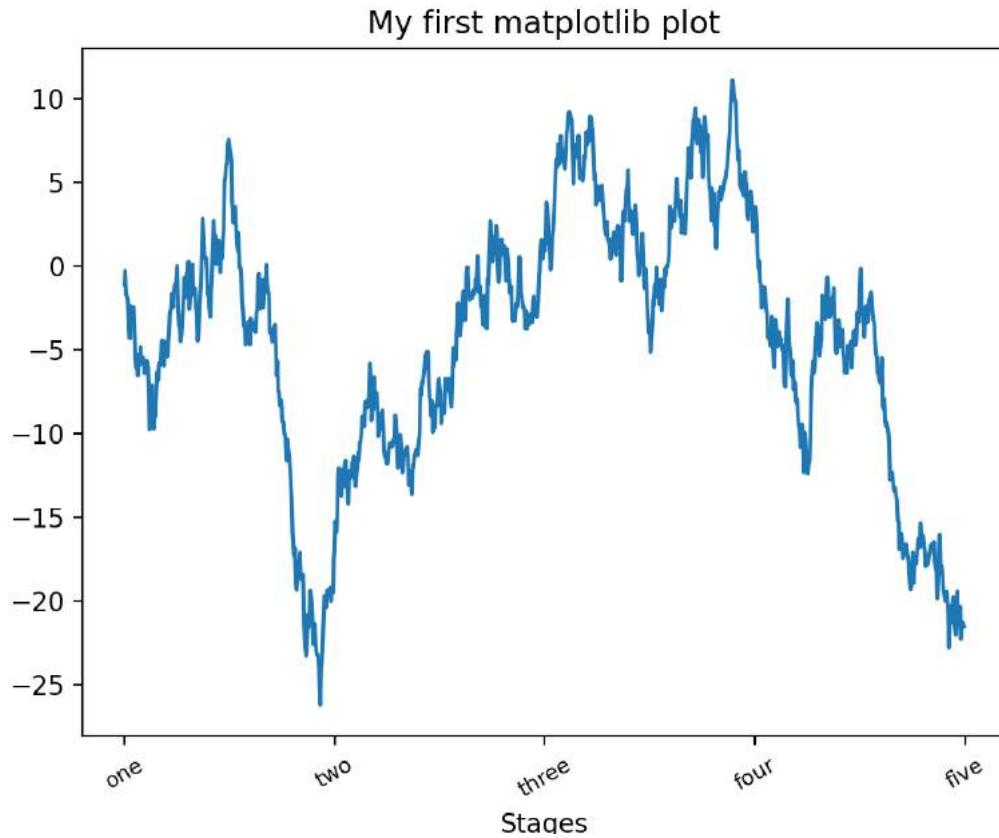
- Bağımsız değişken olmadan çağrıldığında, geçerli parametre değerini döndürür (ör. Plt.xlim (), geçerli x ekseni çizim aralığını döndürür)
- Parametrelerle çağrıldığında parametre değerini ayarlar (ör. Plt.xlim ([0, 10]), x ekseni aralığını 0 ila 10'a ayarlar)

Bu tür tüm yöntemler, aktif veya en son oluşturulmuş AxesSubplot üzerinde haraket eder. Her biri, alt grafik nesnesinin kendisinde iki yöntemle karşılık gelir; xlim durumunda bunlar ax.get_xlim ve ax.set_xlim'dir.

Başlığı, eksen etiketlerini, onay işaretlerini ve onay etiketlerini ayarlama

In [29]:

```
fig = plt.figure()
```



In [31]:

```
ax = fig.add_subplot(1, 1, 1)
```

```
<ipython-input-31-9bd4eae76b03>:1: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.
```

```
ax = fig.add_subplot(1, 1, 1)
```

In [32]:

```
ax.plot(np.random.randn(1000).cumsum())
```

Out[32]:

```
[<matplotlib.lines.Line2D at 0x7ff8663373d0>]
```

In [33]:

```
from IPython.display import Image
Image("img/picture57.png")
```

Out[33]:

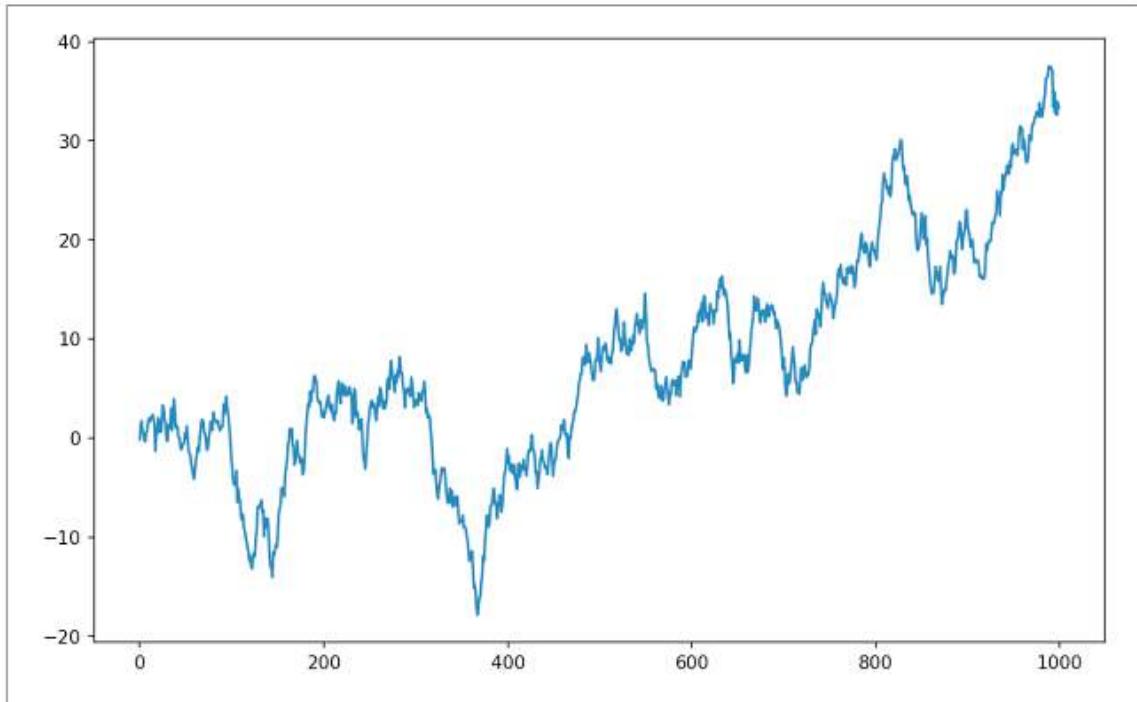


Figure 9-8. Simple plot for illustrating xticks (with label)

X ekseni işaretlerini değiştirmek için en kolayı `set_xticks` ve `set_xticklabels` kullanmaktır. İlk, `matplotlib`'e kenelerin veri aralığı boyunca nereye yerleştirileceğini bildirir; varsayılan olarak bu konumlar aynı zamanda etiketler olacaktır. Ancak `set_xticklabels` kullanarak diğer değerleri etiket olarak ayarlayabiliriz.

In [34]:

```
ticks = ax.set_xticks([0, 250, 500, 750, 1000])
```

In [35]:

```
labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'],
                           rotation=30, fontsize='small')
```

Döndürme seçeneği, x onay etiketlerini 30 derecelik bir dönüşe ayarlar.

In [36]:

```
ax.set_title('My first matplotlib plot')
```

Out[36]:

```
Text(0.5, 1.0, 'My first matplotlib plot')
```

In [37]:

```
ax.set_xlabel('Stages')
```

Out[37]:

```
Text(0.5, 24.438360157402354, 'Stages')
```

In [38]:

```
Image("img/picture58.png")
```

Out[38]:

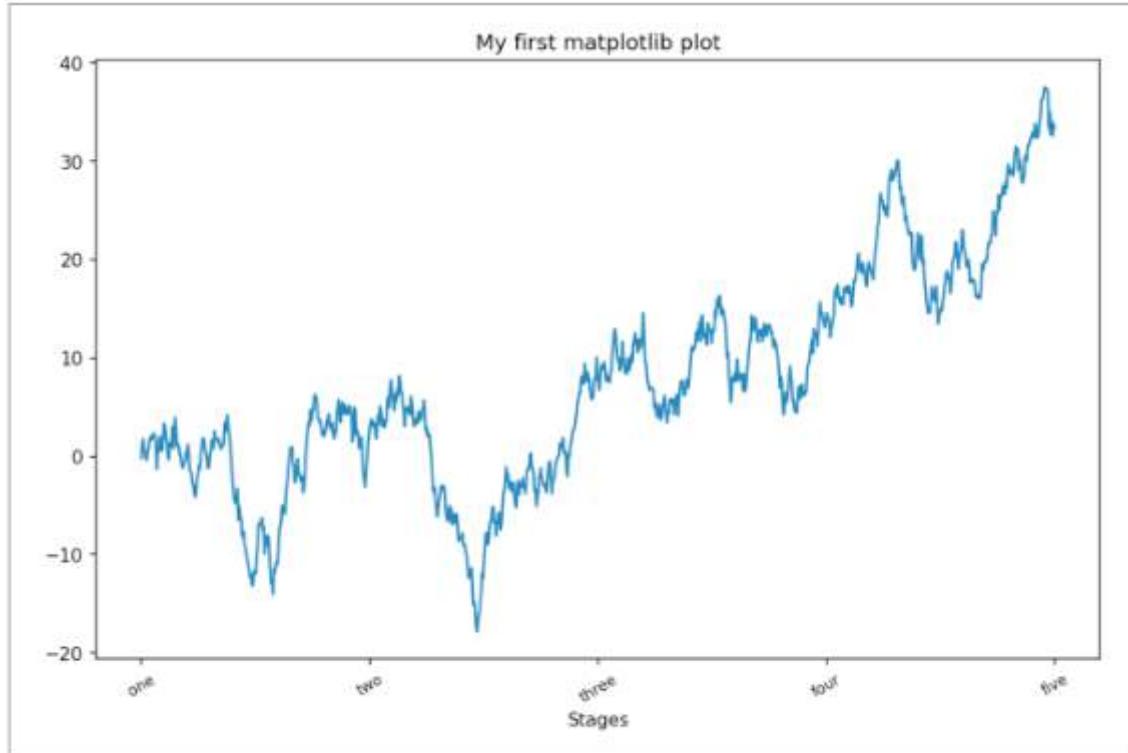


Figure 9-9. Simple plot for illustrating xticks

Y eksenini değiştirmek, yukarıda x yerine y yerine geçen aynı süreci içerir. Axes sınıfı, çizim özelliklerinin toplu olarak ayarlanması izin veren bir set yöntemine sahiptir. Önceki örnekten şunu da yazabilirdik:

In [39]:

```
props = {
    'title': 'My first matplotlib plot',
    'xlabel': 'Stages'
}
ax.set(**props)
```

Out[39]:

```
[Text(0.5, 24.438360157402354, 'Stages'),
 Text(0.5, 1.0, 'My first matplotlib plot')]
```

legends ekleme

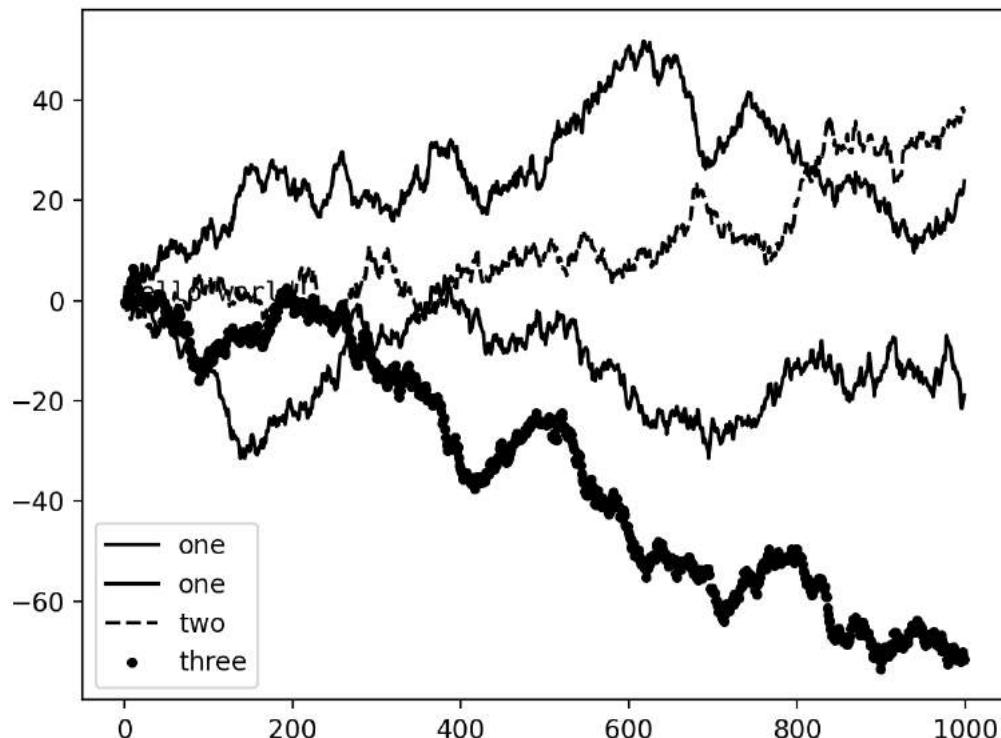
legends, plot öğelerini tanımlamak için başka bir kritik unsurdur. Bir tane eklemenin birkaç yolu vardır. En kolayı, arsanın her bir parçasını eklerken etiket argümanını iletmemektir.

In [40]:

```
from numpy.random import randn
```

In [41]:

```
fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
```



In [43]:

```
ax.plot(randn(1000).cumsum(), 'k', label='one')
```

Out[43]:

```
[<matplotlib.lines.Line2D at 0x7ff867cb7eb0>]
```

In [44]:

```
ax.plot(randn(1000).cumsum(), 'k--', label='two')
```

Out[44]:

```
[<matplotlib.lines.Line2D at 0x7ff867d151f0>]
```

In [45]:

```
ax.plot(randn(1000).cumsum(), 'k.', label='three')
```

Out[45]:

```
[<matplotlib.lines.Line2D at 0x7ff866341250>]
```

Bunu yaptıktan sonra, otomatik olarak bir lejand oluşturmak için ax.legend() veya plt.legend() çağırabilirsiniz. Ortaya çıkan grafik Şekil 9-10'da:

In [46]:

```
ax.legend(loc='best')
```

Out[46]:

```
<matplotlib.legend.Legend at 0x7ff867d15a00>
```

In [47]:

Image("img/picture59.png")

Out[47]:

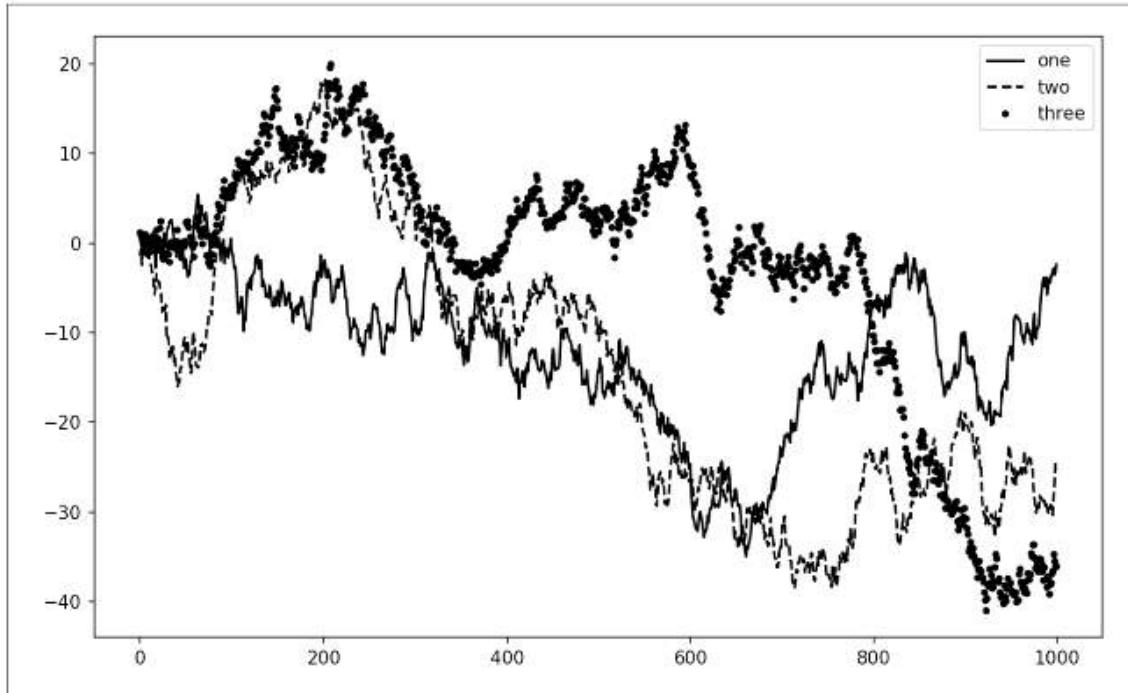


Figure 9-10. Simple plot with three lines and legend

Açıklama yöntemi, konum loc argümanı için birkaç başka seçenek sahiptir. Lo c, matplotlib'e arsayı nereye yerleştireceğini söyler. Seçici değilseniz, 'e n iyi' iyi bir seçenektedir, çünkü en çok yoldan çıkan yeri seçecektir. Göster geden bir veya daha fazla ögeyi hariç tutmak için, etiket veya etiket = '_ n olegend_' iletmeyin.

Bir Alt Grafikte Ek Açıklamalar ve Çizim

Standart çizim türlerine ek olarak, metin, oklar veya diğer şekillerden oluş abilecek kendi grafik notlarınızı çizmek isteyebilirsiniz. Metin, ok ve açık lama işlevlerini kullanarak ek açıklamalar ve metin ekleyebilirsiniz. metin, isteğe bağlı özel stil ile çizim üzerinde belirli koordinatlarda (x, y) meti n çizer.

In [48]:

```
x=0
y=0
ax.text(x, y, 'Hello world!',
        family='monospace', fontsize=10)
```

Out[48]:

Text(0, 0, 'Hello world!')

Ek açıklamalar, uygun şekilde düzenlenmiş hem metin hem de oklar çizebilir.

Örnek olarak, 2007'den beri kapanış S&P 500 endeks fiyatını (Yahoo! Finance'den alınmıştır) çizelim ve 2008–2009 mali krizinin bazı önemli tarihleriyle birlikte not alalım. Bu kod örneğini bir Jupyter not defterinde tek bir hücrede en kolay şekilde yeniden üretebilirsiniz.

In [49]:

```
from datetime import datetime
import pandas as pd
import numpy as np
```

In [50]:

```

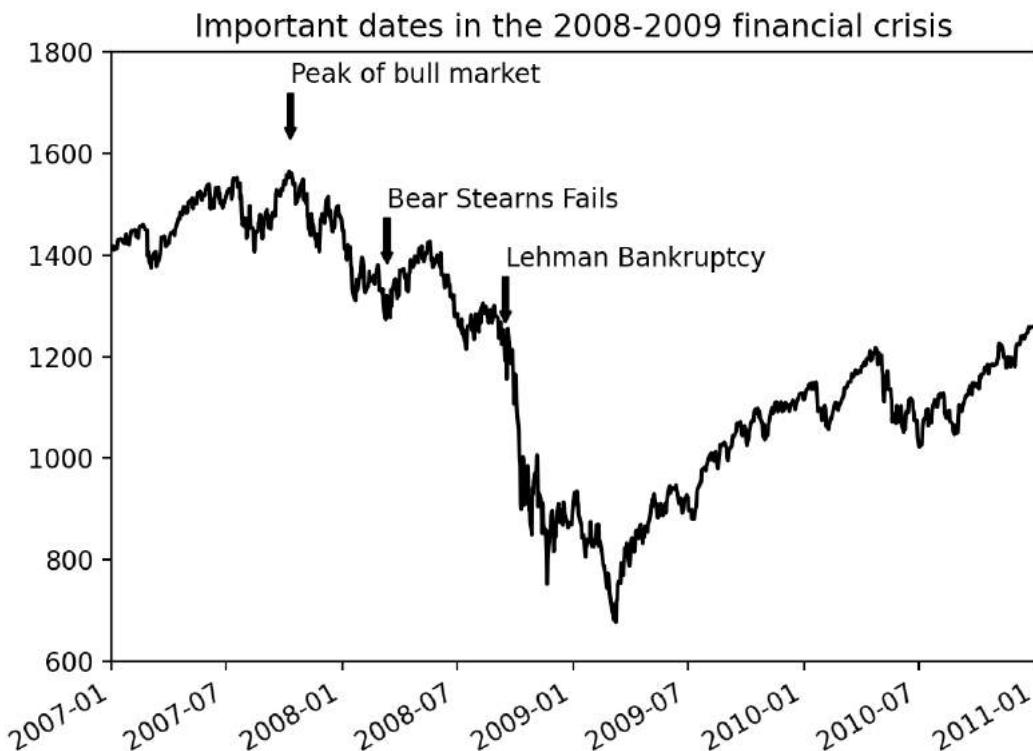
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
data = pd.read_csv('examples/spx.csv', index_col=0, parse_dates=True)
spx = data['SPX']
spx.plot(ax=ax, style='k-')

crisis_data = [
    (datetime(2007, 10, 11), 'Peak of bull market'),
    (datetime(2008, 3, 12), 'Bear Stearns Fails'),
    (datetime(2008, 9, 15), 'Lehman Bankruptcy')
]

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')

# Zoom in on 2007-2010
ax.set_xlim(['1/1/2007', '1/1/2011'])
ax.set_ylim([600, 1800])
ax.set_title('Important dates in the 2008-2009 financial crisis')

```



Out[50]:

```
Text(0.5, 1.0, 'Important dates in the 2008-2009 financial crisis')
```

Bu grafikte vurgulanması gereken birkaç önemli nokta vardır: `ax.annotate` yöntemi, belirtilen x ve y koordinatlarında etiketler çizebilir. Matplotlib'in varsayılanını kullanmak yerine çizimin başlangıç ve bitiş sınırlarını manuel olarak ayarlayabiliriz.

l olarak ayarlamak için `set_xlim` ve `set_ylim` yöntemlerini kullanıyoruz. Son olarak, `ax.set_title` arsaya bir ana başlık ekler.

Şekil çizmek biraz daha özen gerektirir. `matplotlib`, yamalar olarak adlandırılan birçok genel şekli temsil eden nesnelere sahiptir. Bunlardan `Rectangle` ve `Circle` gibi bazıları `matplotlib.pyplot` içinde bulunur, ancak tam `matplotlib.patches` içinde bulunur.

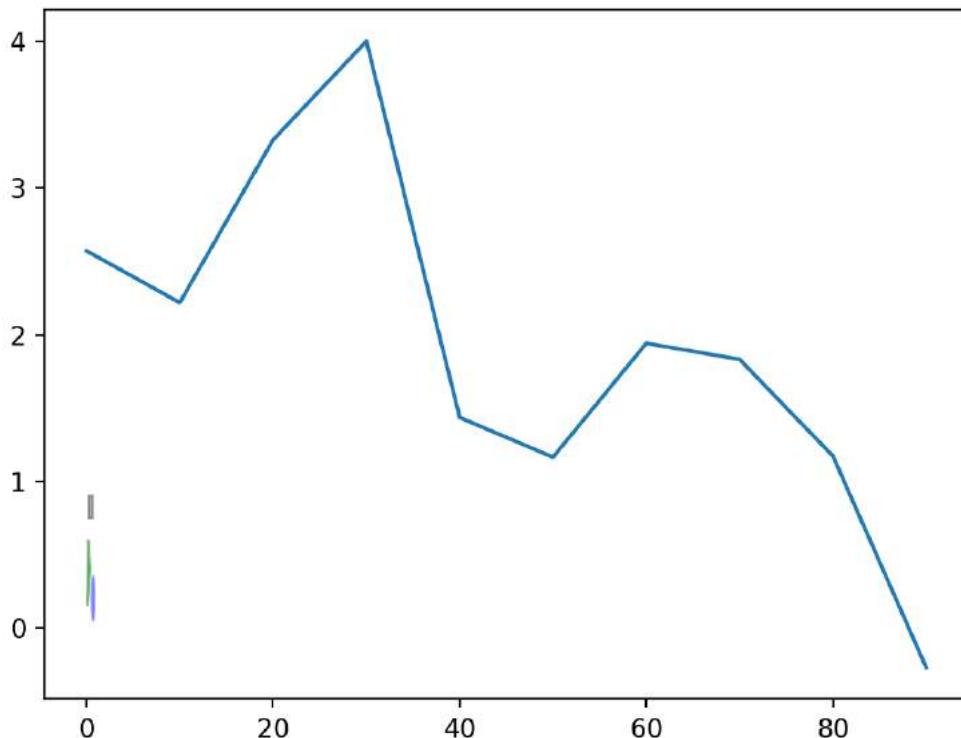
Bir grafiğe şekil eklemek için, yama nesnesi `shp`'yi oluşturursunuz ve `ax.add_patch(shp)` 'yi çağırarak onu bir alt grafiğe eklersiniz.

In [51]:

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

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)

ax.add_patch(rect)
ax.add_patch(circ)
ax.add_patch(pgon)
```



Out[51]:

<matplotlib.patches.Polygon at 0x7ff86a072a30>

Grafikleri Dosyaya Kaydetme

Etkin şekli plt.savefig kullanarak dosyaya kaydedebilirsiniz. Bu yöntem, şekil nesnesinin savefig örnek yöntemine eşdeğerdir. Örneğin, bir şeklin SVG ve rsiyonunu kaydetmek için yalnızca şunu yazmanız gereklidir.

In [52]:

```
plt.savefig('figpath.svg')
```

Dosya türü, dosya uzantısından çıkarılır. Yani bunun yerine .pdf kullanırsanız, bir PDF alırsınız. Grafik yayınılmak için sık sık kullandığım birkaç önlük:

emli seçenek var: iç başına nokta çözünürlüğünü kontrol eden dpi ve gerçek şeklin etrafındaki beyaz boşluğu kesebilen bbox_inches. PNG ile aynı grafiği, arsa çevresinde minimum beyaz boşlukla ve 400 DPI'da elde etmek için şunu yaparsınız:

In [53]:

```
plt.savefig('figpath.png', dpi=400, bbox_inches='tight')
```

savefig'in diske yazması gerekmektedir; herhangi bir dosya benzeri nesneye de yazabilir, örneğin

BytesIO:

In [54]:

```
from io import BytesIO
buffer = BytesIO()
plt.savefig(buffer)
plot_data = buffer.getvalue()
```

In [55]:

```
Image("img/picture60.png")
```

Out[55]:

Table 9-2. Figure.savefig options

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 Yapılandırması

matplotlib, öncelikli olarak şekillerin yayına hazırlanmasına yönelik renk şemaları ve varsayılan ayarlarla yapılandırılmış olarak gelir. Neyse ki, varsayılan davranışların neredeyse tamamı, şekil boyutu, alt alan aralığı, renkler, yazı tipi boyutları, ızgara stilleri ve benzerlerini yöneten kapsamlı bir küresel parametreler kümesi aracılığıyla özelleştirilebilir. Yapılandırmayı Python'dan programlı olarak değiştirmenin bir yolu rc yöntemini kullanmaktadır; örneğin, genel varsayılan şekil boyutunu 10×10 olarak ayarlamak için şunu girebilirsiniz:

In [56]:

```
plt.rc('figure', figsize=(10, 10))
```

RC'nin ilk argümanı, 'figure', 'axes', 'xtick', 'ytick', 'grid', 'legend' ve ya diğerleri gibi özelleştirmek istediğiniz bileşendir. Bundan sonra, yeni parametreleri gösteren bir dizi anahtar kelime argümanı takip edilebilir. Programınızdaki seçenekleri yazmanın kolay bir yolu, bir diktedir.

```
font_options = {'family' : 'monospace',
                'weight' : 'bold',
                'size' : 'small'}
plt.rc('font', **font_options)
```

9.2 Pandas ve seaborn ile plotlama

matplotlib oldukça düşük seviyeli bir araç olabilir. Temel bileşenlerinden bir çizim oluşturursunuz: veri ekranı (yani çizim türü: çizgi, çubuk, kutu, dağılım, kontur vb.)

Pandas'da, satır ve sütun etiketleriyle birlikte birden çok veri sütunumuz olabilir. Pandas, Veri Çerçeveesi ve Seri nesnelerinden görselleştirmeler oluşturmayı basitleştiren yerleşik yöntemlere sahiptir. Diğer bir kitaplık ise Michael Waskom tarafından oluşturulan bir istatistiksel grafik kitabı olan seaborn'dur. Seaborn, birçok yaygın görselleştirme türü oluşturmayı basitleştirir.

Çizgi Grafikleri

Seri ve Veri Çerçeveisinin her biri, bazı temel çizim türlerini yapmak için bir çizim özniteliğine sahiptir. Varsayılan olarak, plot () çizgi grafikleri yapar.

In [57]:

```
s = pd.Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10))
```

In [58]:

```
s.plot()
```

Out[58]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff86a056880>
```

In [59]:

Image("img/picture61.png")

Out[59]:

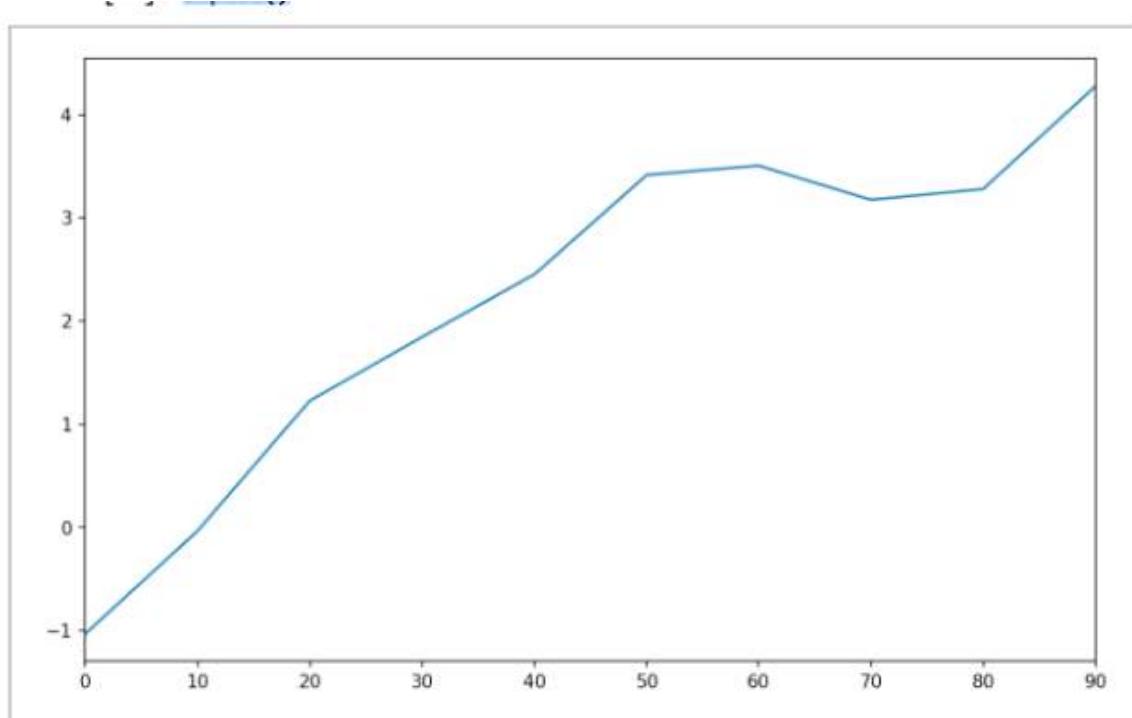


Figure 9-13. Simple Series plot

Series nesnesinin dizini, x ekseninde çizim yapmak için matplotlib'e aktarılır, ancak `use_index = False` değerini ileterek bunu devre dışı bırakabilirsiniz.

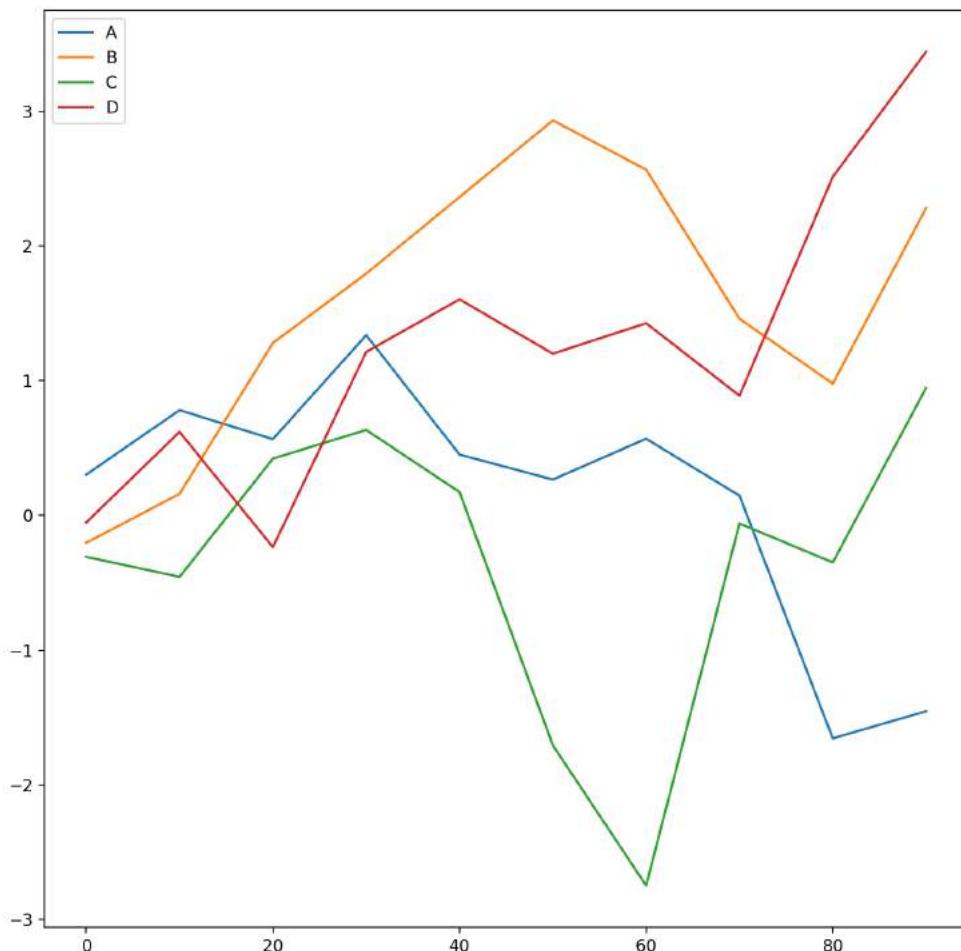
Pandas çizim yöntemlerinin çoğu, bir matplotlib alt çizim nesnesi olabilecek istege bağlı bir `ax` parametresini kabul eder. Bu, size bir grid düzendeinde alt alanların daha esnek bir şekilde yerleştirilmesini sağlar.

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]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff86154c6a0>
```

In [62]:

Image("img/picture62.png")

Out[62]:

Table 9-3. Series.plot method arguments

Argument	Description
label	Label for plot legend
ax	matplotlib subplot object to plot on; if nothing passed, uses active matplotlib subplot
style	Style string, like 'ko--', to be passed to matplotlib
alpha	The plot fill opacity (from 0 to 1)
kind	Can be 'area', 'bar', 'barh', 'density', 'hist', 'kde', 'line', 'pie'
logy	Use logarithmic scaling on the y-axis
use_index	Use the object index for tick labels
rot	Rotation of tick labels (0 through 360)
xticks	Values to use for x-axis ticks
yticks	Values to use for y-axis ticks
xlim	x-axis limits (e.g., [0, 10])
ylim	y-axis limits
grid	Display axis grid (on by default)

DataFrame, sütunların nasıl işlendiği konusunda biraz esneklik sağlayan bir dizi seçenekse sahiptir; örneğin, hepsini aynı alt üzerinde işaretlemek veya ayrı alt grafikler oluşturmak için. Bunlar hakkında daha fazla bilgi iç in Tablo 9-4'e bakın.

In [63]:

Image("img/picture63.png")

Out[63]:

Table 9-4. DataFrame-specific plot arguments

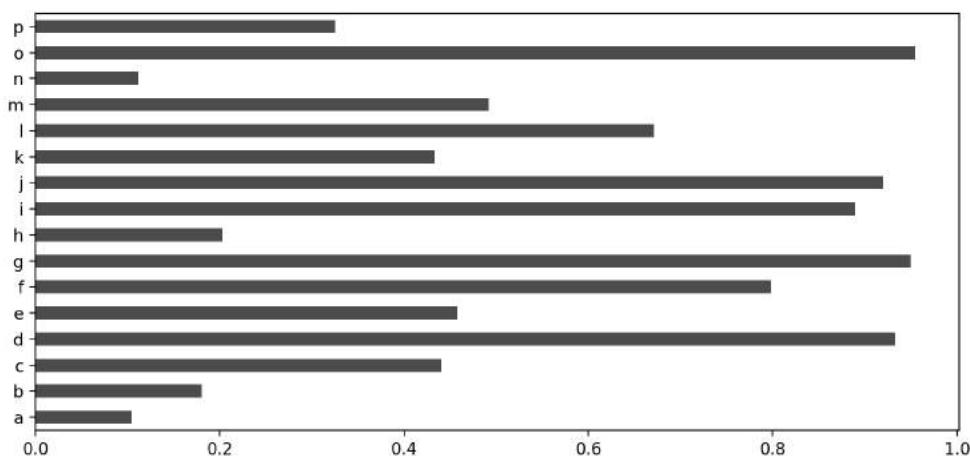
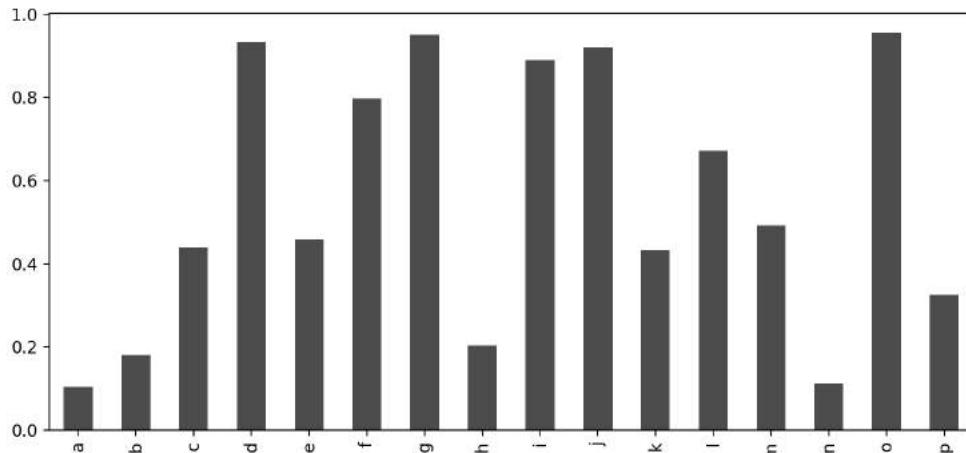
Argument	Description
subplots	Plot each DataFrame column in a separate subplot
sharex	If subplots=True, share the same x-axis, linking ticks and limits
sharey	If subplots=True, share the same y-axis
figsize	Size of figure to create as tuple
title	Plot title as string
legend	Add a subplot legend (True by default)
sort_columns	Plot columns in alphabetical order; by default uses existing column order

Çubuk Grafikler

Plot.bar () ve plot.bart (), sırasıyla dikey ve yatay çubuk grafikleri oluşturur. Bu durumda, Series veya DataFrame dizini x (bar) veya y (bart) işaretleri olarak kullanılacaktır.

In [64]:

```
fig, axes = plt.subplots(2, 1)
data = pd.Series(np.random.rand(16), index=list('abcdefghijklmno'))
data.plot.bar(ax=axes[0], color='k', alpha=0.7)
data.plot.bart(ax=axes[1], color='k', alpha=0.7)
```



Out[64]:

<matplotlib.axes._subplots.AxesSubplot at 0x7ff84df8be20>

In [65]:

```
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 [66]:

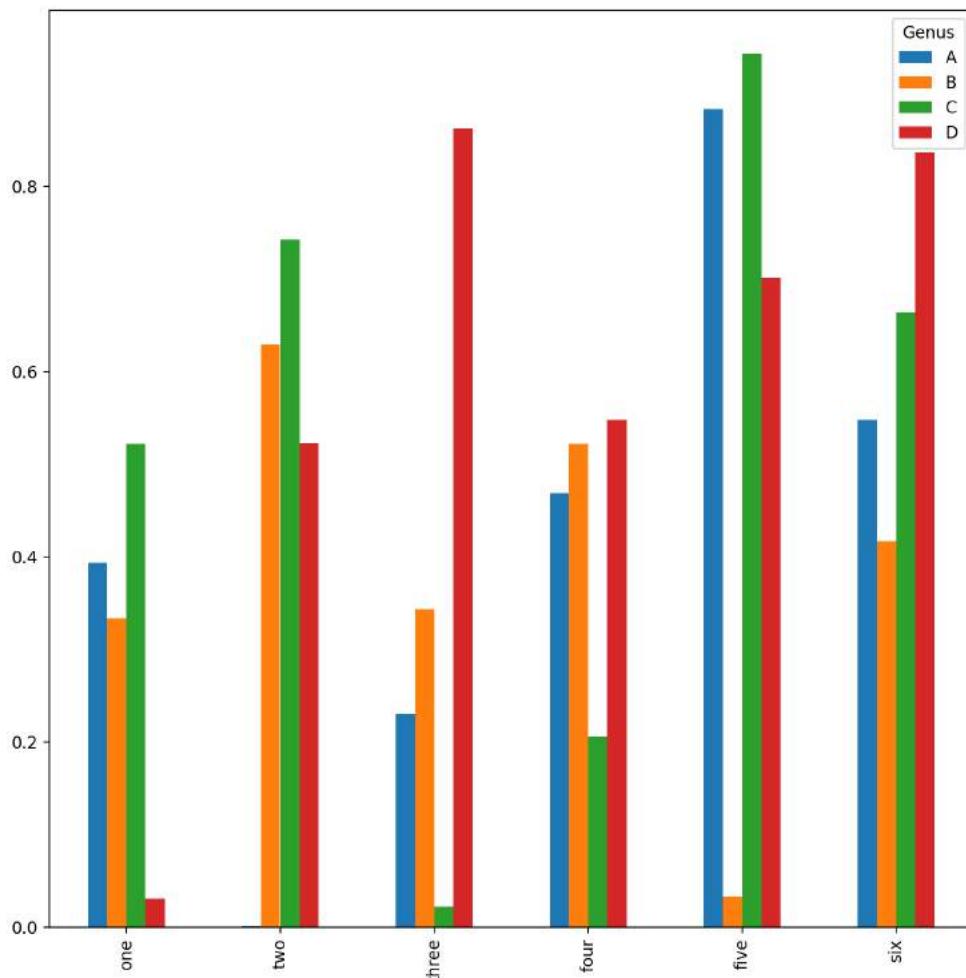
```
df
```

Out[66]:

Genus	A	B	C	D
one	0.393550	0.333993	0.522495	0.030785
two	0.001277	0.629919	0.743181	0.522823
three	0.230003	0.343547	0.022736	0.863640
four	0.468790	0.522169	0.205630	0.548133
five	0.883813	0.033031	0.943529	0.702137
six	0.547888	0.417075	0.663827	0.836742

In [67]:

```
df.plot.bar()
```



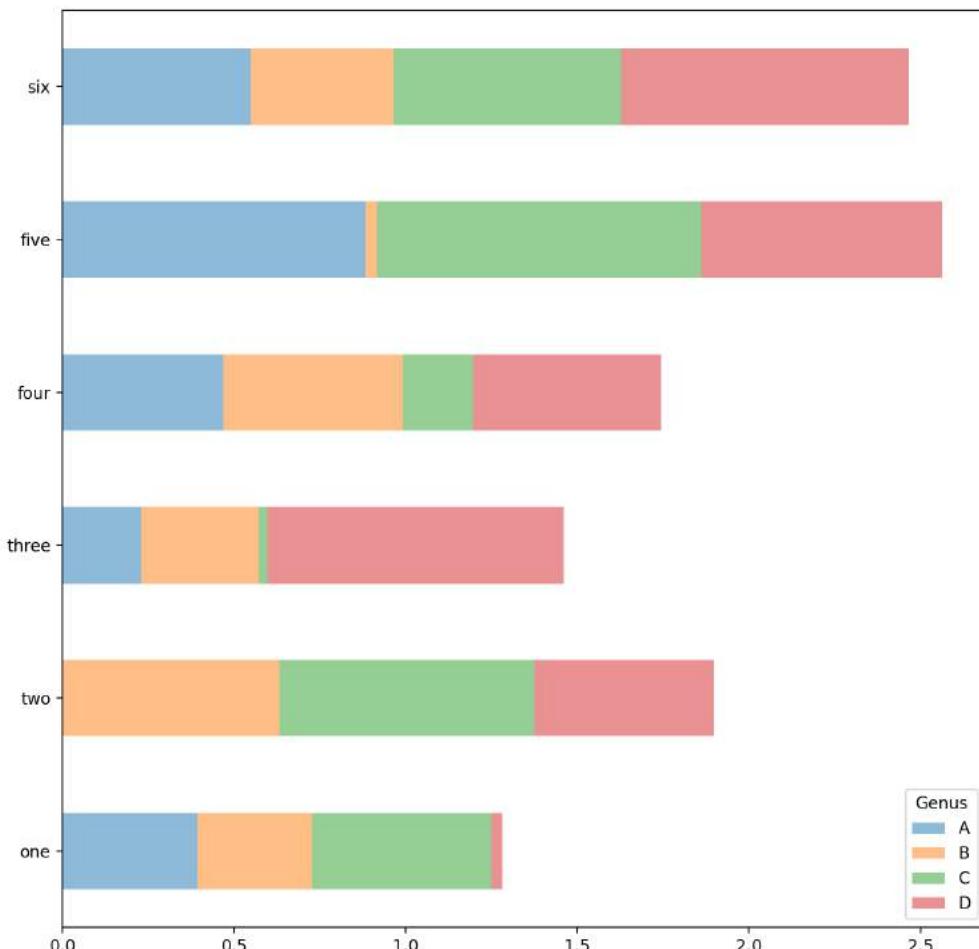
Out[67]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff84e265fd0>
```

Stacked = True geçerek bir DataFrame'den yığınlanmış çubuk grafikleri oluşturuyoruz, bu da her satırdaki değerin birlikte istiflenmesine neden oluyor.

In [68]:

```
df.plot.barh(stacked=True, alpha=0.5)
```



Out[68]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff84e357b50>
```

Daha önce kullanılan devrilme veri kümesine dönersek, her gün her bir parti boyutu için veri noktalarının yüzdesini gösteren yiğilmiş bir çubuk grafiği

yapmak istediğimizi varsayıyalım. Verileri `read_csv` kullanarak yükleyelim ve gün ve parti boyutuna göre bir çapraz tablo oluşturalım.

In [69]:

```
tips = pd.read_csv('examples/tips.csv')
```

In [70]:

```
party_counts = pd.crosstab(tips['day'], tips['size'])
```

In [71]:

```
party_counts
```

Out[71]:

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

In [72]:

```
party_counts = party_counts.loc[:, 2:5]
```

In [73]:

```
party_pcts = party_counts.div(party_counts.sum(1), axis=0)
```

In [74]:

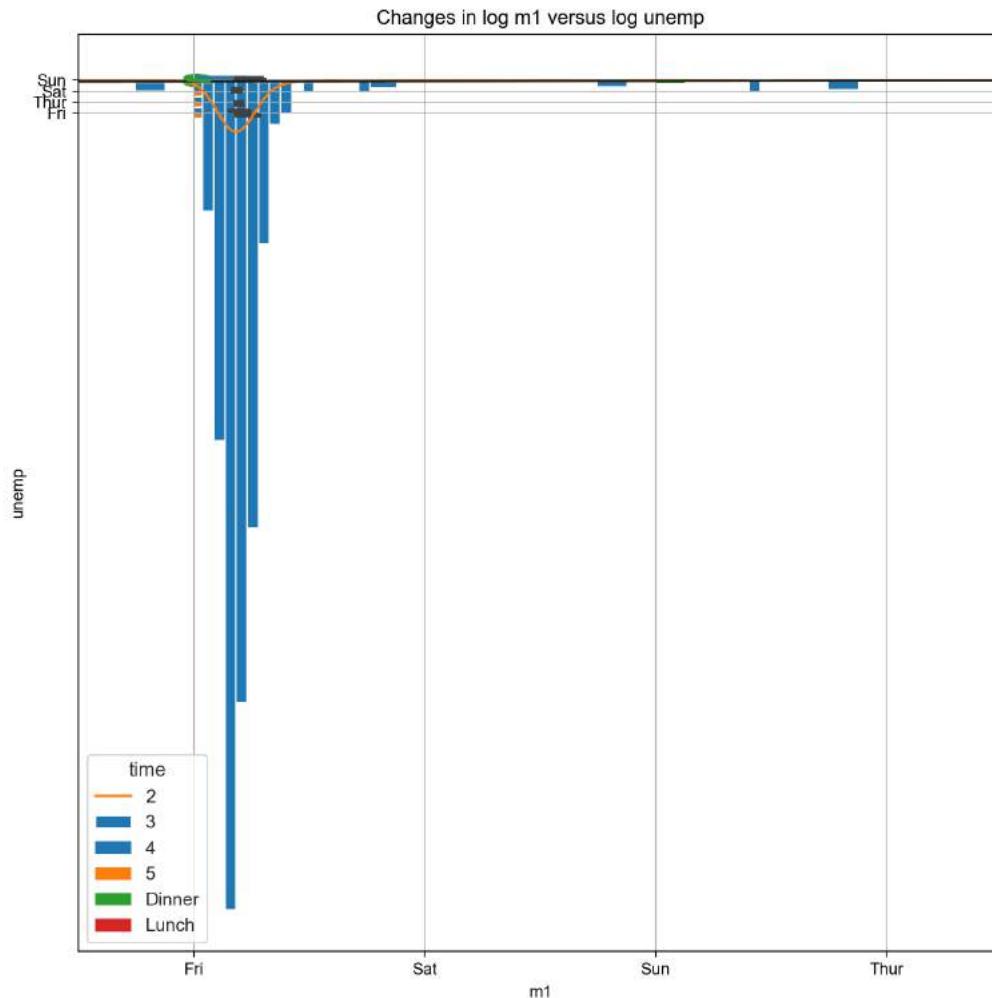
```
party_pcts
```

Out[74]:

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 [75]:

```
party_pcts.plot.bar()
```



Out[75]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff84e3147f0>
```

Dolayısıyla, bu veri kümesinde hafta sonu parti boyutlarının arttığını görebilirsiniz.

Bir olay örgüsü oluşturmadan önce toplama veya özetleme gerektiren verilerle, seaborn paketini kullanmak işleri çok daha basit hale getirebilir. Şimdi seaborn ile günlük devrilme yüzdesine bakalım.

In [76]:

```
!conda install -c anaconda seaborn -y
```

Collecting package metadata (current_repodata.json): done
Solving environment: done

All requested packages already installed.

In [77]:

```
import seaborn as sns
```

In [78]:

```
tips['tip_pct'] = tips['tip'] / (tips['total_bill'] - tips['tip'])
```

In [79]:

```
tips.head()
```

Out[79]:

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01		No	Sun	Dinner	2 0.063204
1	10.34	1.66		No	Sun	Dinner	3 0.191244
2	21.01	3.50		No	Sun	Dinner	3 0.199886
3	23.68	3.31		No	Sun	Dinner	2 0.162494
4	24.59	3.61		No	Sun	Dinner	4 0.172069

In [80]:

```
sns.barplot(x='tip_pct', y='day', data=tips, orient='h')
```

Out[80]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff84e3147f0>
```

Seaborn'daki çizim fonksiyonları, bir pandanın Veri Çerçevesi olabilen bir veri argümanı alır. Diğer argümanlar sütun adlarına atıfta bulunur. Gün içindeki her bir değer için birden fazla gözlem olduğundan, çubuklar ortalama tip_pct değeridir. Çubukların üzerine çizilen siyah çizgiler, % 95 güven aralığıni temsil eder (bu, isteğe bağlı argümanlarla yapılandırılabilir).

In [81]:

```
sns.barplot(x='tip_pct', y='day', hue='time', data=tips, orient='h')
```

Out[81]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff84e3147f0>
```

Seaborn'un çizimlerin estetiğini otomatik olarak değiştirdiğine dikkat edin: varsayılan renk paleti, çizim arka planı ve ızgara çizgi renkleri. Seaborn.set'i kullanarak farklı arsa görünümleri arasında geçiş yapabilirsiniz.

In [82]:

```
sns.set(style="whitegrid")
```

Histogramlar ve Yoğunluk Grafikleri

Histogram, değer frekansının ayrık bir görüntüsünü veren bir tür çubuk grafiktir. Veri noktaları ayrı, eşit aralıklı bölmelere bölünür ve her bölmedeki veri noktalarının sayısı çizilir. Önceki bahış verilerini kullanarak, Serideki plot.hist yöntemini kullanarak toplam faturanın bahış yüzdelerinin histogramını oluşturabiliriz.

In [83]:

```
tips['tip_pct'].plot.hist(bins=50)
```

Out[83]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff84e3147f0>
```

In [84]:

```
Image("img/picture64.png")
```

Out[84]:

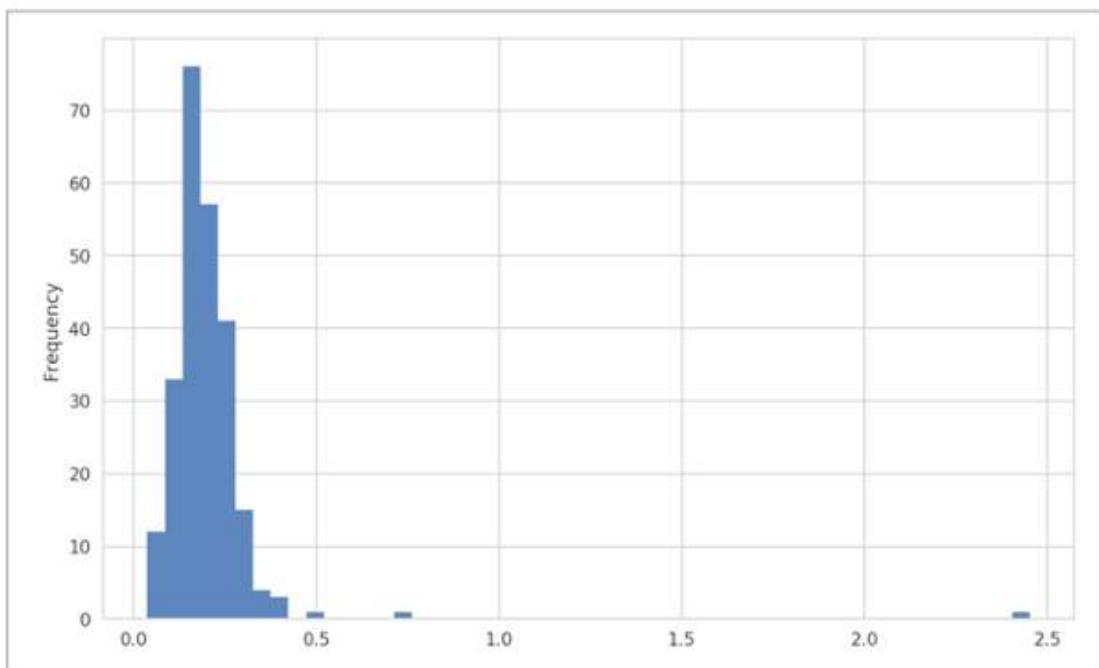


Figure 9-21. Histogram of tip percentages

İlgili bir çizim türü, gözlemlenen verileri oluşturmuş olabilecek sürekli bir olasılık dağılımının bir tahminini hesaplayarak oluşturulan bir yoğunluk g

rafiğidir. Genel prosedür, bu dağılımı "çekirdeklerin" bir karışımı olarak tanımlanır - yani, normal dağılım gibi daha basit dağılımlar. Bu nedenle, yoğunluk grafikleri aynı zamanda çekirdek yoğunluğu tahmini (KDE) grafikleri olarak da bilinir. Plot.kde'yi kullanmak, geleneksel normallerin karışımını tahminini kullanarak bir yoğunluk grafiği yapar.

In [85]:

```
tips['tip_pct'].plot.density()
```

Out[85]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff84e3147f0>
```

In [86]:

```
Image("img/picture65.png")
```

Out[86]:

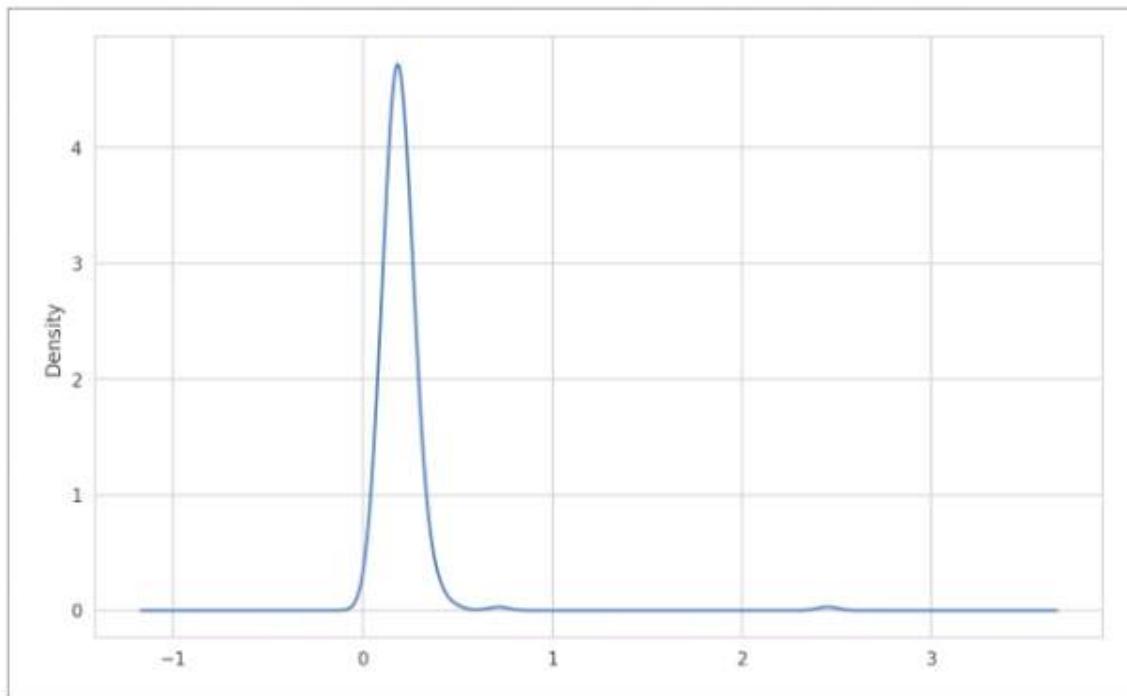


Figure 9-22. Density plot of tip percentages

Seaborn, hem histogramı hem de sürekli yoğunluk tahminini eşzamanlı olarak çizebilen distplot yöntemiyle histogramları ve yoğunluk çizimlerini daha da kolaylaştırır. Örnek olarak, iki farklı standart normal dağılımdan gelen çekimlerden oluşan iki modlu bir dağılımı düşünün.

In [87]:

```
comp1 = np.random.normal(0, 1, size=200)
comp2 = np.random.normal(10, 2, size=200)
values = pd.Series(np.concatenate([comp1, comp2]))
sns.distplot(values, bins=100, color='k')
```

/Users/veyseldogan/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
warnings.warn(msg, FutureWarning)

Out[87]:

<matplotlib.axes._subplots.AxesSubplot at 0x7ff84e3147f0>

In [88]:

Image("img/picture66.png")

Out[88]:

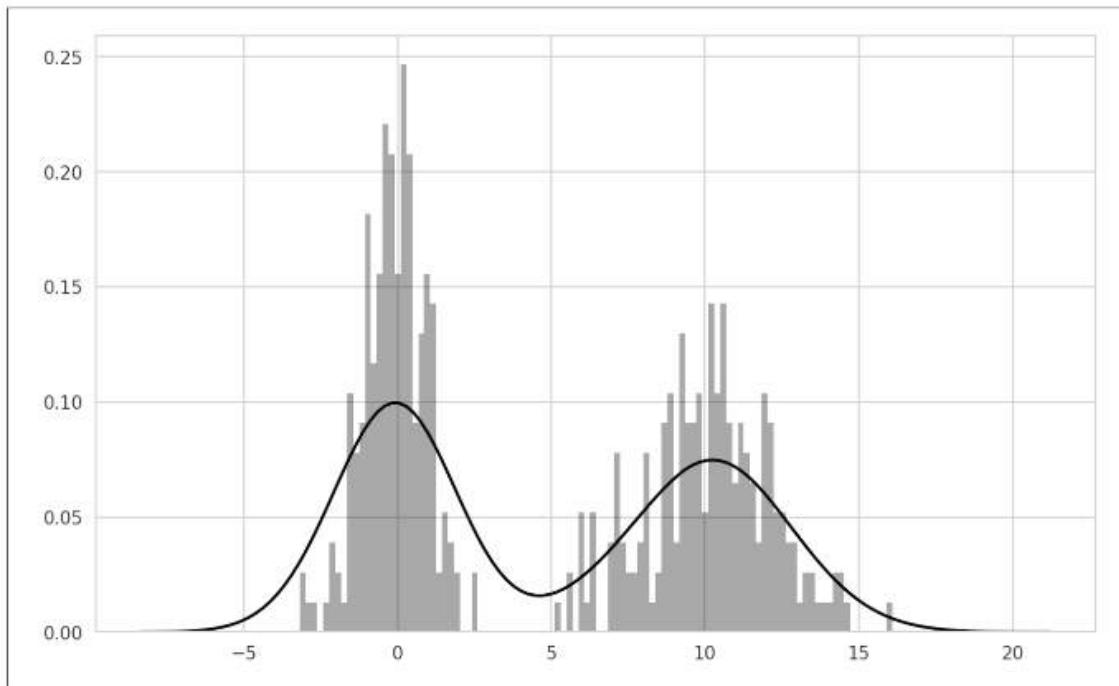


Figure 9-23. Normalized histogram of normal mixture with density estimate

Dağılım veya Nokta Grafikleri

Nokta grafikleri veya dağılım grafikleri, iki tek boyutlu veri serisi arasındaki ilişkiyi incelemenin yararlı bir yolu olabilir. Örneğin, burada istatistik modelleri projesinden macrodata veri kümesini yükliyoruz, birkaç değişken seçiyoruz ve ardından günlük farklılıklarını hesaplıyoruz.

In [89]:

```
macro = pd.read_csv('examples/macrodata.csv')
data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]
trans_data = np.log(data).diff().dropna()
trans_data[-5:]
```

Out[89]:

	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

In [90]:

```
sns.regplot('m1', 'unemp', data=trans_data)
plt.title('Changes in log %s versus log %s' % ('m1', 'unemp'))
```

/Users/veyseldogan/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```

Out[90]:

```
Text(0.5, 1.0, 'Changes in log m1 versus log unemp')
```

In [91]:

Image("img/picture67.png")

Out[91]:

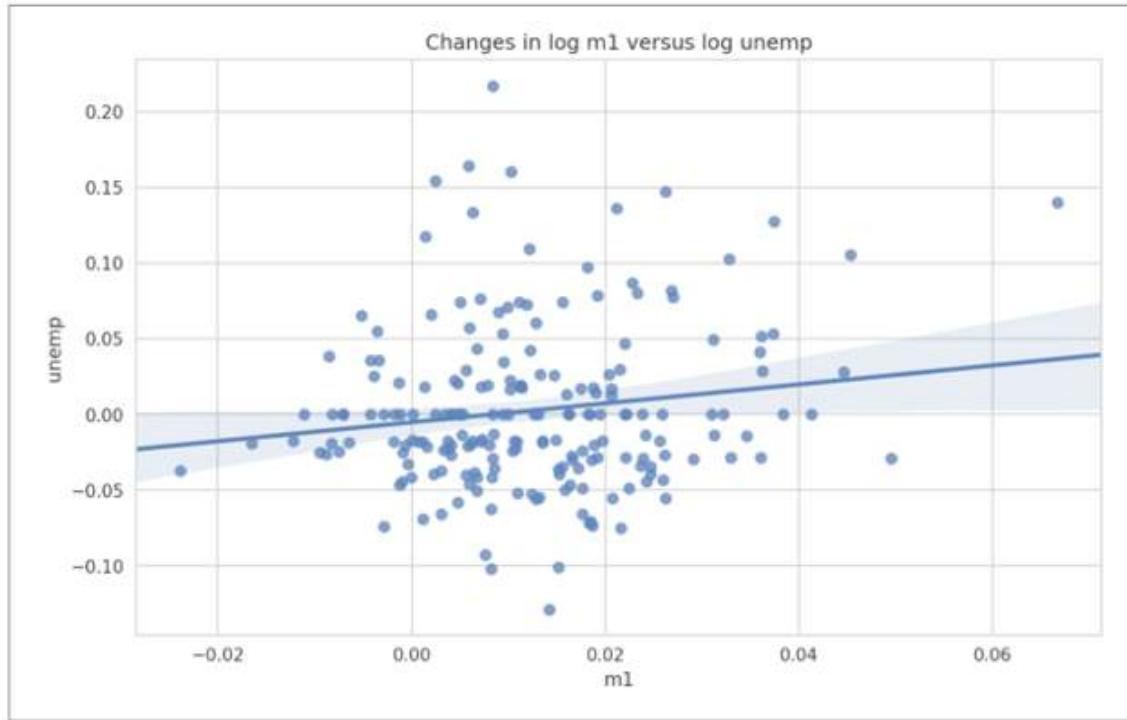


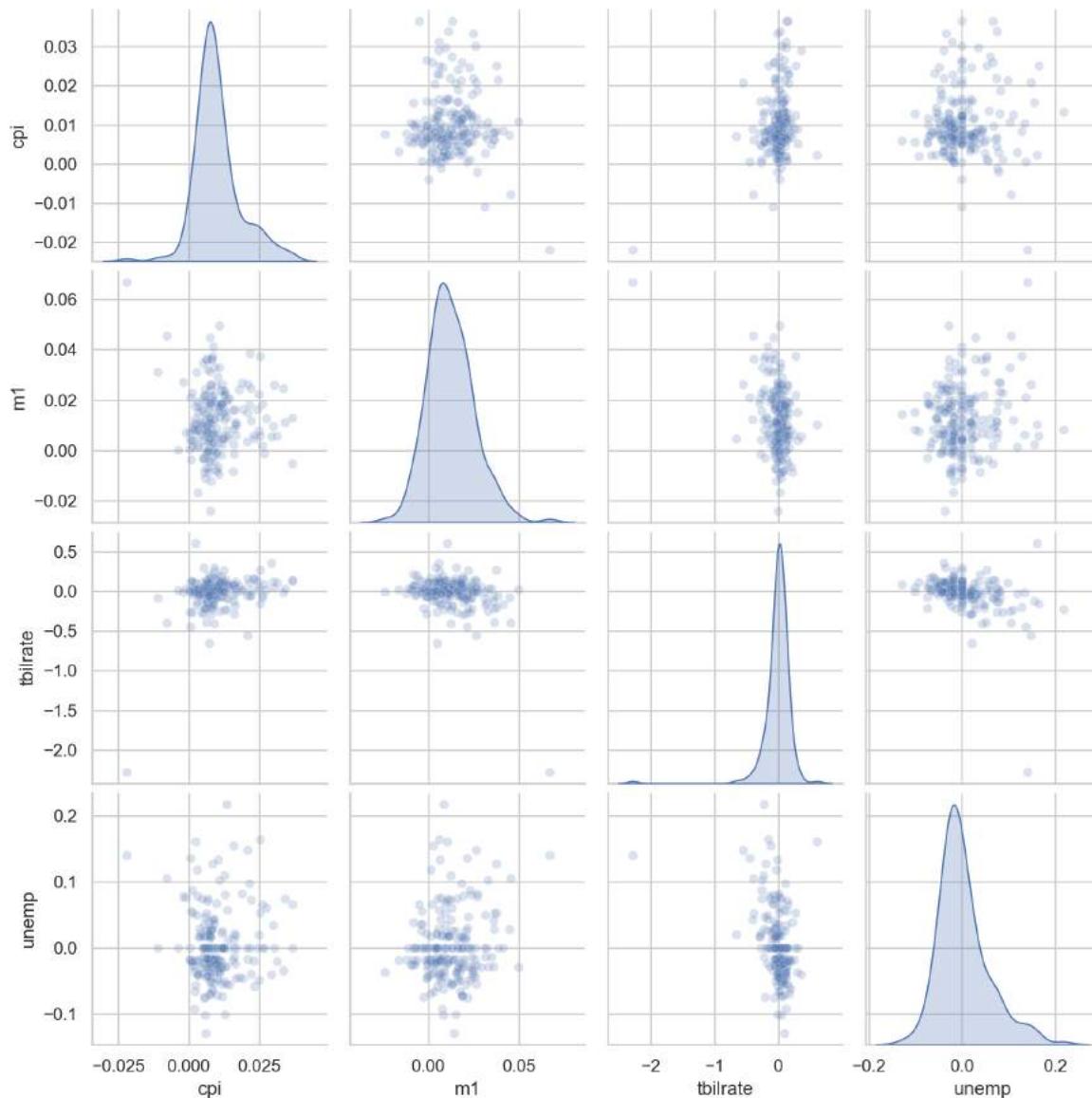
Figure 9-24. A seaborn regression/scatter plot

Keşif amaçlı veri analizinde, bir grup değişken arasındaki tüm dağılım grafiğine bakabilmek yararlıdır; bu, çiftler grafiği veya dağılım grafiği matrisi olarak bilinir. Sıfırdan böyle bir çizim yapmak biraz istir, bu nedenle seaborn, diyagonal boyunca her bir değişkenin histogramlarını veya yoğunluk

tahminlerini yerleştirmeyi destekleyen uygun bir çift diyagram işlevine sahiptir.

In [92]:

```
sns.pairplot(trans_data, diag_kind='kde', plot_kws={'alpha': 0.2})
```



Out[92]:

```
<seaborn.axisgrid.PairGrid at 0x7ff8588b3fa0>
```

In [97]:

Image("img/picture68.png")

Out[97]:

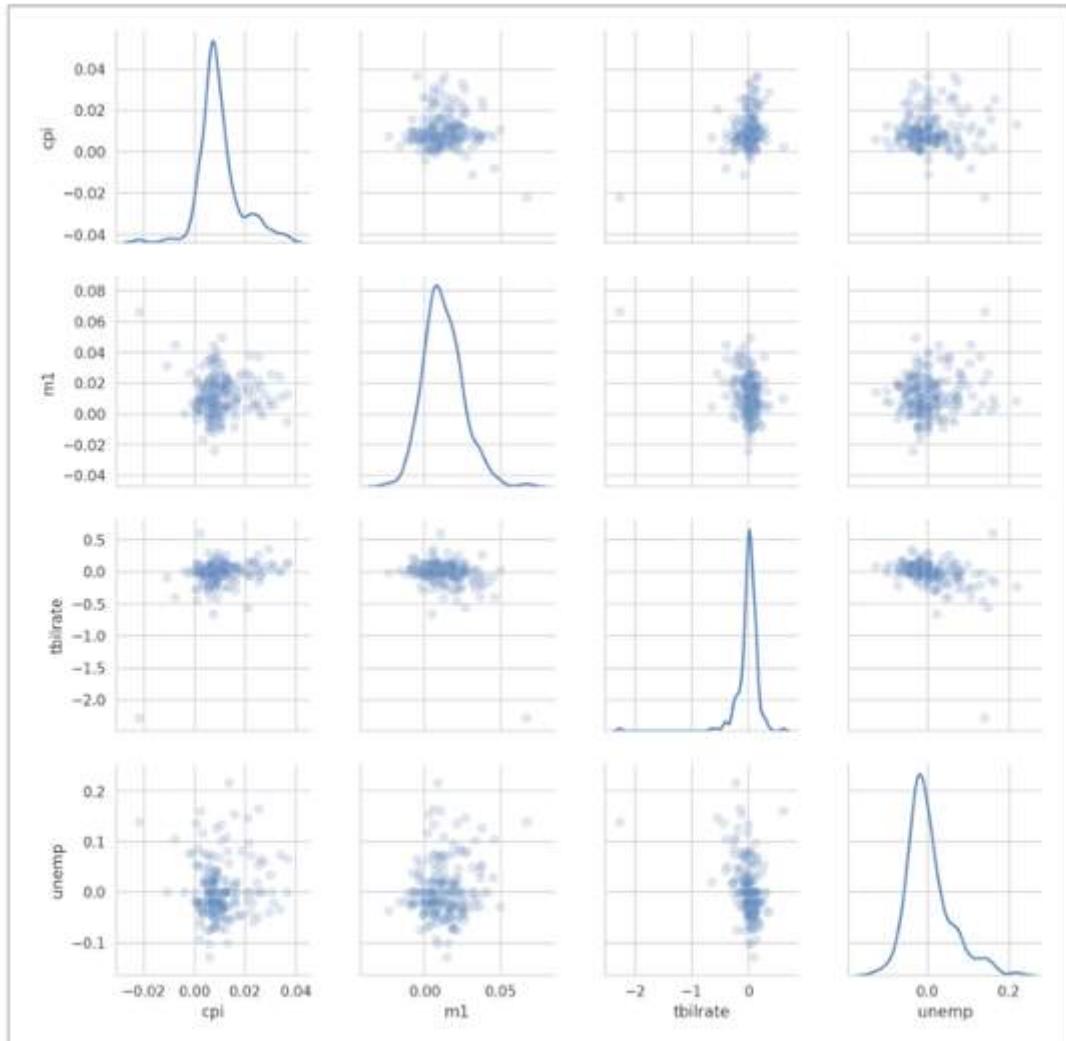


Figure 9-25. Pair plot matrix of statsmodels macro data

Plot_kws argümanını fark edebilirsiniz. Bu, konfigürasyon seçeneklerini çapraz olmayan elemanlar üzerindeki bireysel çizim çağrılarına aktarmamızı sağlar.

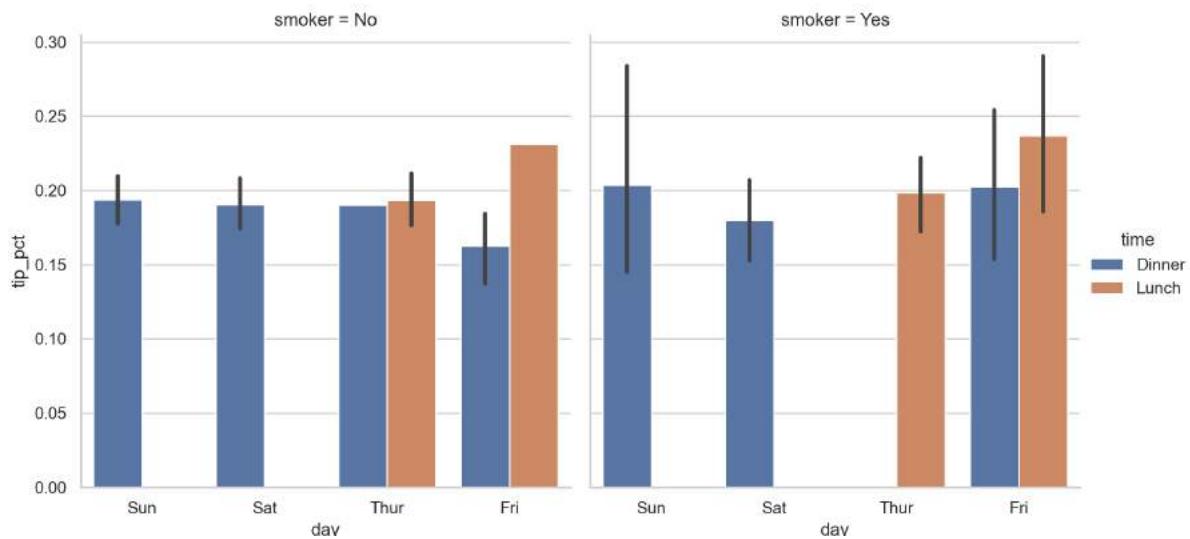
Facet Grids and Kategorik Veriler

Ek gruplama boyutlarımızın olduğu veri kümeleri ne olacak? Verileri birçok kategorik değişkenle görselleştirmenin bir yolu, yüz ızgarası kullanmaktır. Seaborn, birçok çeşit yönlü grafik oluşturmayı basitleştiren kullanışlı bir yerleşik fonksiyon faktör grafiğine sahiptir.

In [98]:

```
sns.factorplot(x='day', y='tip_pct', hue='time', col='smoker',
                 kind='bar', data=tips[tips.tip_pct < 1])
```

```
/Users/veyseldogan/opt/anaconda3/lib/python3.8/site-packages/seaborn/categorical.py:3704: UserWarning: The `factorplot` function has been renamed to `catplot`. The original name will be removed in a future release. Please update your code. Note that the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.
warnings.warn(msg)
```



Out[98]:

```
<seaborn.axisgrid.FacetGrid at 0x7ff83c2af550>
```

In [99]:

```
Image("img/picture69.png")
```

Out[99]:

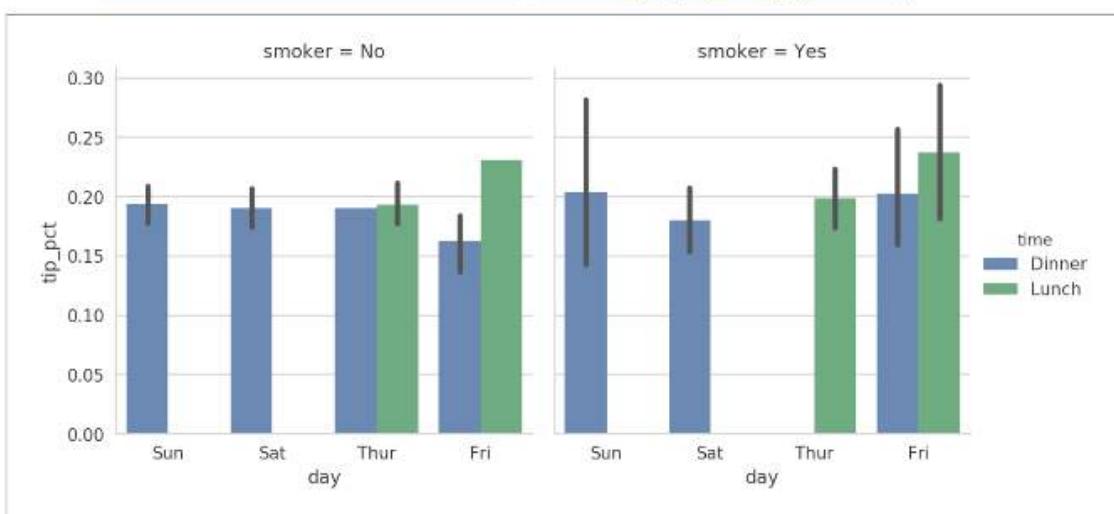
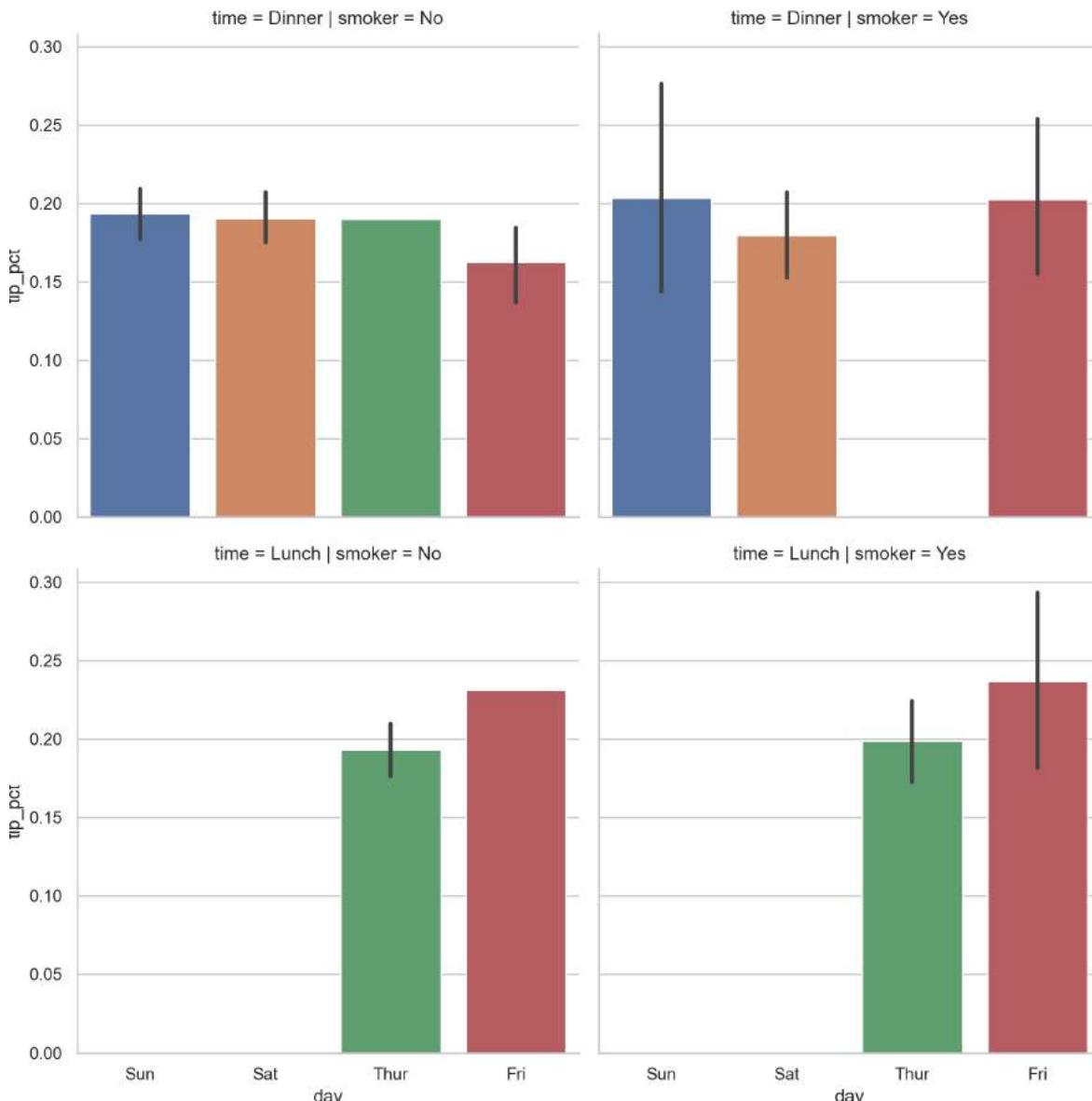


Figure 9-26. Tipping percentage by day/time/smoker

In [100]:

```
sns.factorplot(x='day', y='tip_pct', row='time',
                col='smoker',
                kind='bar', data=tips[tips.tip_pct < 1])
```

/Users/veyseldogan/opt/anaconda3/lib/python3.8/site-packages/seaborn/categorical.py:3704: UserWarning: The `factorplot` function has been renamed to `catplot`. The original name will be removed in a future release. Please update your code. Note that the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.
warnings.warn(msg)



Out[100]:

```
<seaborn.axisgrid.FacetGrid at 0x7ff84c8ddf10>
```

In [101]:

Image("img/picture70.png")

Out[101]:

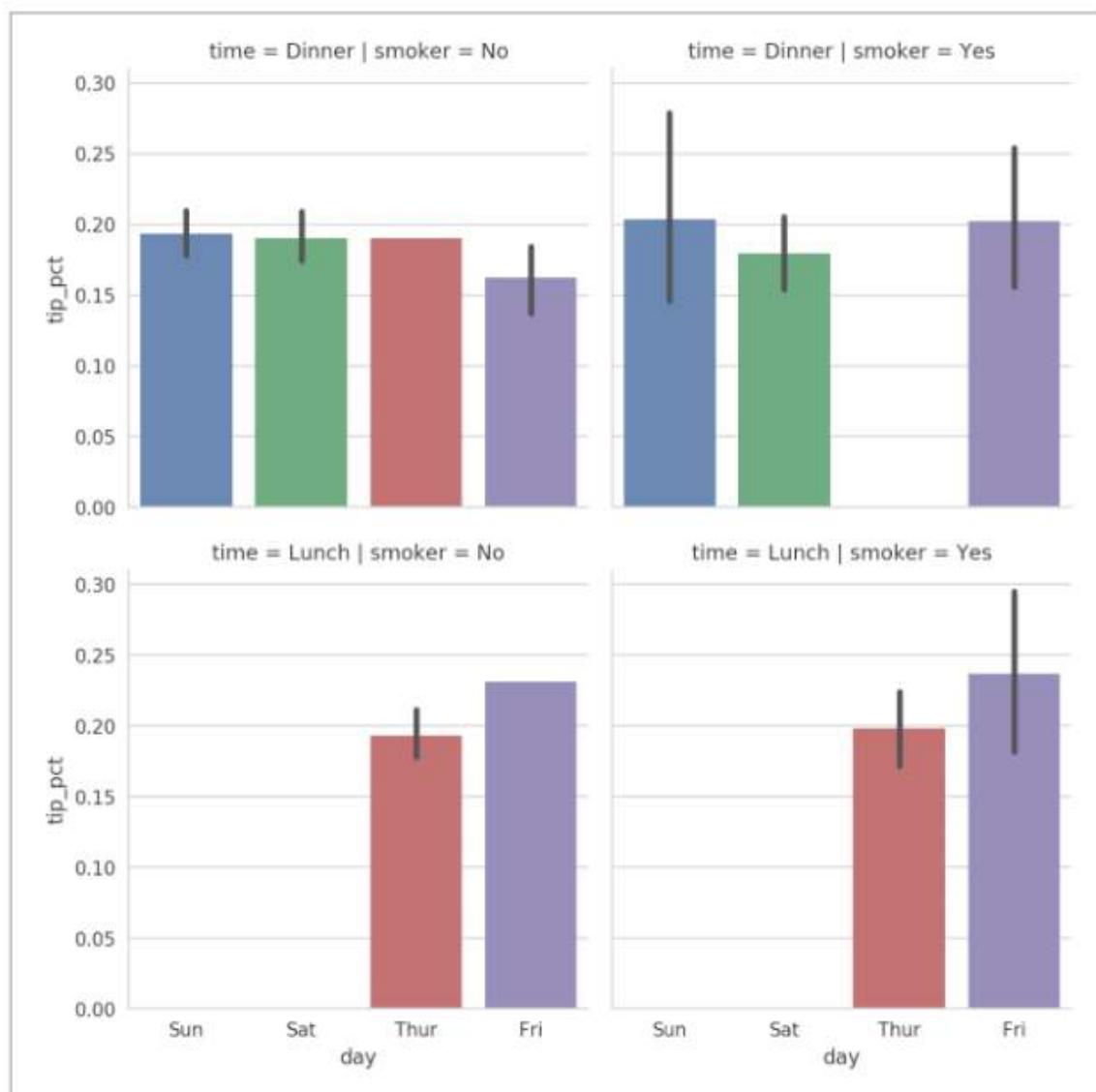


Figure 9-27. tip_pct by day; facet by time/smoker

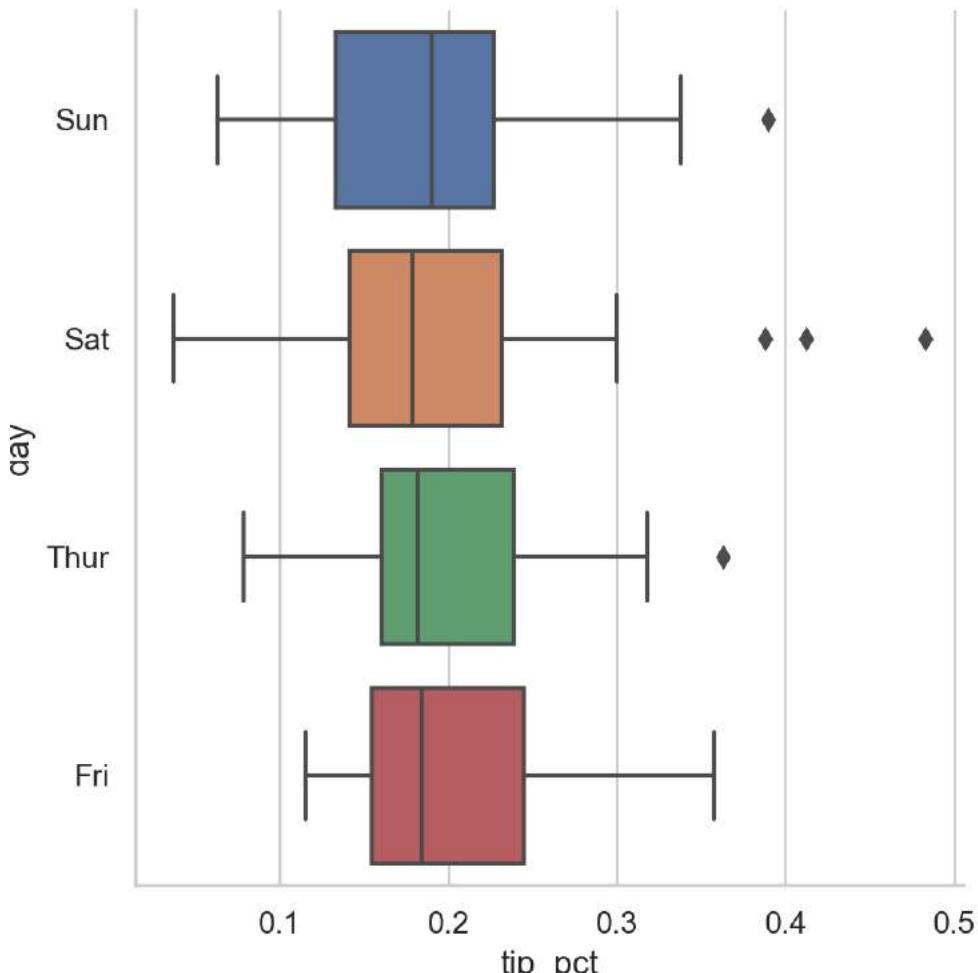
faktör grafiği, görüntülemeye çalışığınız şeye bağlı olarak yararlı olabilecek diğer çizim türlerini destekler. Örneğin, kutu grafikleri (medyan, çeyrek

kler ve dışları gösteren) etkili bir görselleştirme türü olabilir.

In [102]:

```
sns.factorplot(x='tip_pct', y='day', kind='box',
                 data=tips[tips.tip_pct < 0.5])
```

```
/Users/veyseldogan/opt/anaconda3/lib/python3.8/site-packages/seaborn/c
ategorical.py:3704: UserWarning: The `factorplot` function has been re
named to `catplot`. The original name will be removed in a future rele
ase. Please update your code. Note that the default `kind` in `factorp
lot` (`'point'`) has changed `'strip'` in `catplot`.
warnings.warn(msg)
```



Out[102]:

```
<seaborn.axisgrid.FacetGrid at 0x7ff83d3da0a0>
```

In [103]:

Image("img/picture71.png")

Out[103]:

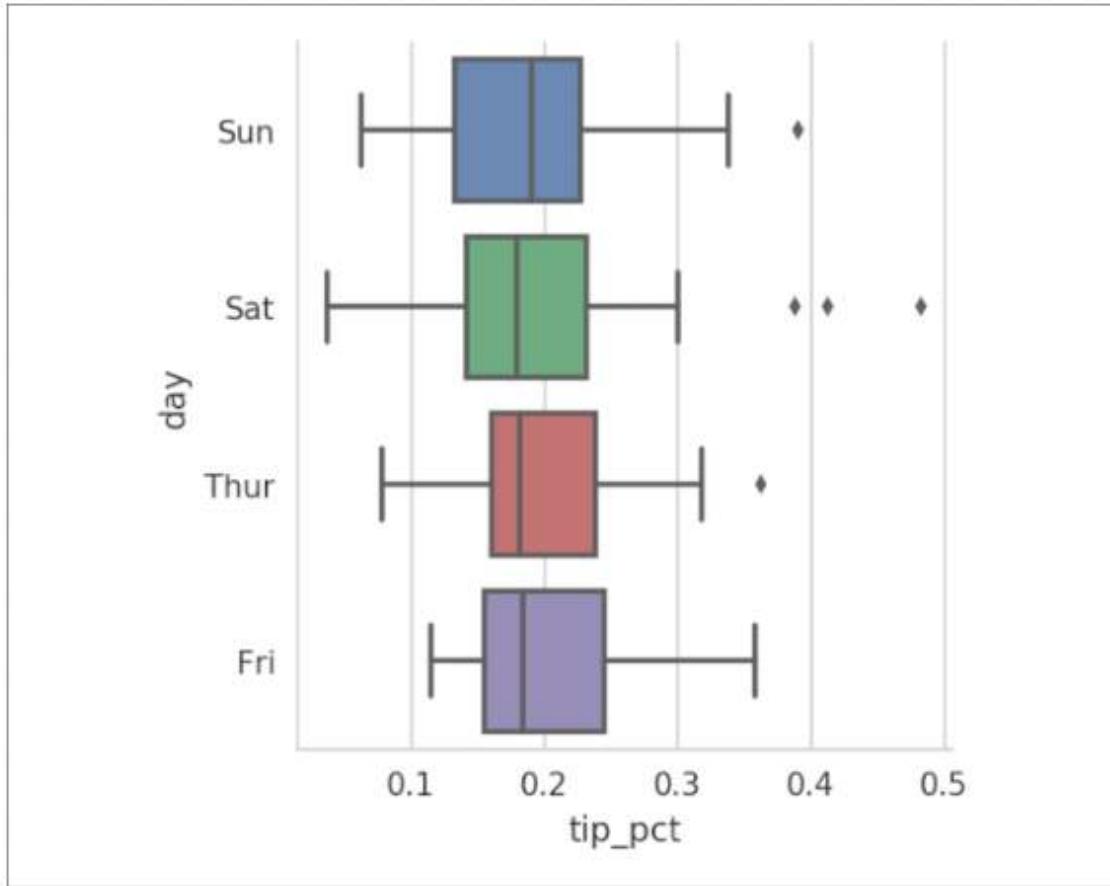


Figure 9-28. Box plot of tip_pct by day

9.3 Diğer Python Görselleştirme Araçları

Açık kaynakta yaygın olduğu gibi, Python'da grafik oluşturmak için çok sayıd a seçenek vardır. Birçok geliştirme çabası web'de yayınlanmak üzere etkileşimi li grafikler oluşturmaya odaklanmıştır. Bokeh ve Plotly gibi araçlarla, Python'da bir web tarayıcısı için tasarlanmış dinamik, etkileşimli grafikler be lirlemek artık mümkün.

Veri Toplama ve Grup İşlemleri

Bir veri kümesini kategorize etmek ve bir toplama veya dönüştürme olsun, her gruba bir işlev uygulamak, genellikle bir veri analizi iş akışının kritik bir bileşenidir. Bir veri kümesini yükledikten, birleştirildikten ve hazırladıktan sonra, raporlama veya görselleştirme amacıyla grup istatistiklerini veya muhtemelen pivot tabloları hesaplamamanız gerekebilir. Pandas, veri kümelerin doğal bir şekilde dilimlemenize, parçalara ayırmaya ve özetlemenize olanak tanıyan esnek bir grup arabirimini sağlar.

İlişkisel veri tabanlarının ve SQL'in popüler olmasının bir nedeni, verilerin birleştirilebilmesi, filtrelenmesi, dönüştürülmesi ve toplanabilme kolaylığıdır. Bununla birlikte, SQL gibi sorgu dilleri, gerçekleştirilebilecek grup işlemleri türlerinde bir şekilde sınırlıdır. Göreceğiniz gibi, Python ve pandasın ifade gücüyle, bir pandas nesnesini veya NumPy dizisini kabul eden herhangi bir işlevi kullanarak oldukça karmaşık grup işlemleri gerçekleştirebiliriz. Bu bölümde aşağıdakilerin nasıl yapılacağını öğreneceksiniz:

- Pandas nesnesini bir veya daha fazla anahtar kullanarak parçalara ayırin (işlevler, diziler veya DataFrame sütun adları biçiminde)
- Sayım, ortalama veya standart sapma veya kullanıcı tanımlı bir işlev gibi grup özet istatistiklerini hesaplayın
- Grup içi dönüşümleri veya normalleştirme, doğrusal regresyon, sıralama veya alt küme seçimi gibi diğer işlemleri uygulayın
- Özet tabloları ve çapraz tabloları hesaplayın
- Nicelik analizi ve diğer istatistiksel grup analizlerini gerçekleştirin

10.1 GroupBy Mechanics

bir DataFrame satırları (`eksen = 0`) veya sütunları (`eksen = 1`) üzerinde gruplanabilir. Bu yapıldıktan sonra, her gruba yeni bir değer üreten bir işlev uygulanır. Son olarak, tüm bu fonksiyon uygulamalarının sonuçları bir sonuç nesnesi olarak birleştirilir. Ortaya çıkan nesnenin şekli genellikle verilere ne yapıldığına bağlı olacaktır.

In [1]:

```
from IPython.display import Image
Image("img/picture72.png")
```

Out[1]:

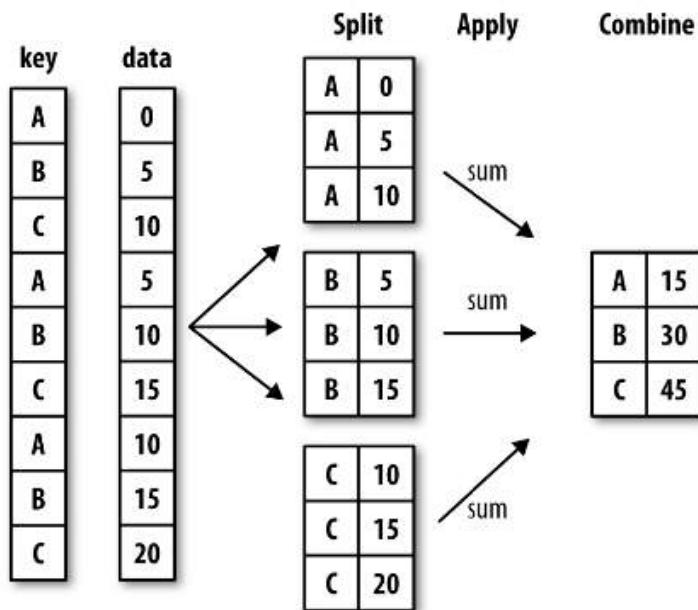


Figure 10-1. Illustration of a group aggregation

Her gruplama anahtarı birçok biçimde olabilir ve anahtarların hepsinin aynı türde olması gerekmek:

- Gruplanmakta olan eksenle aynı uzunlukta olan bir değerler listesi veya dizisi
- DataFrame'deki bir sütun adını belirten bir değer
- Gruplanan eksendeki değerler ile grup isimleri arasında bir yazışma veren bir dikté veya Seri
- Eksen dizininde veya dizindeki tek tek etiketlerde çağrılabilecek bir işlev

Son üç yöntemin, nesneyi bölmek için kullanılacak bir dizi üretmek için kisayollar olduğunu unutmayın.

In [2]:

```
import pandas as pd
import numpy as np
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 [3]:

df

Out[3]:

	key1	key2	data1	data2
0	a	one	-0.399197	-1.392406
1	a	two	-0.583593	0.041499
2	b	one	-1.378632	0.182173
3	b	two	-0.205171	-0.352032
4	a	one	-0.183798	2.131064

Key1'deki etiketleri kullanarak data1 sütununun ortalamasını hesaplamak istediginizi varsayalim. Bunu yapmanin c̄esitli yolları var. Birincisi data1'e erişmek ve groupby'yi key1'deki sütun (a Series) ile aramaktır.

In [4]:

grouped = df['data1'].groupby(df['key1'])

In [5]:

grouped

Out[5]:

<pandas.core.groupby.generic.SeriesGroupBy object at 0x7fe837dc8850>

Bu gruplanmış değişken artık bir GroupBy nesnesidir. Grup anahtarı df['key1'] ile ilgili bazı ara veriler dışında henüz hiçbir şey hesaplamadı. Burada ki fikir, bu nesnenin daha sonra grupların her birine bazı işlemler uygulama için gereken tüm bilgilere sahip olmasıdır. Örneğin, grubu hesaplamak, GroupBy'nin ortalama yöntemini çağırabileceğimiz anlamına gelir.

In [6]:

grouped.mean()

Out[6]:

```
key1
a    -0.388863
b    -0.791901
Name: data1, dtype: float64
```

Bunun yerine birden çok diziyi bir liste olarak geçirseydik, farklı bir şey elde ederdik.

In [7]:

means = df['data1'].groupby([df['key1'], df['key2']]).mean()

In [8]:

means

Out[8]:

```
key1  key2
a    one    -0.291498
      two    -0.583593
b    one    -1.378632
      two    -0.205171
Name: data1, dtype: float64
```

Burada verileri iki key kullanarak grupladık ve ortaya çıkan Seri artık gözlemlenen benzersiz key çiftlerinden oluşan hiyerarşik bir dizine sahip.

In [9]:

means.unstack()

Out[9]:

key2	one	two
key1		
a	-0.291498	-0.583593
b	-1.378632	-0.205171

Bu örnekte, grup anahtarlarının tümü Seridir, ancak bunlar doğru uzunlukta herhangi bir dizi olabilir.

In [10]:

```
states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
```

In [11]:

```
years = np.array([2005, 2005, 2006, 2005, 2006])
```

In [12]:

```
df['data1'].groupby([states, years]).mean()
```

Out[12]:

```
California 2005    -0.583593
             2006    -1.378632
Ohio        2005    -0.302184
             2006    -0.183798
Name: data1, dtype: float64
```

Çoğunlukla gruplama bilgileri, üzerinde çalışmak istediğiniz verilerle aynı DataFrame'de bulunur. Bu durumda, sütun adlarını (dizeler, sayılar veya diğer Python nesneleri) grup anahtarları olarak geçirebilirsiniz.

In [13]:

```
df.groupby('key1').mean()
```

Out[13]:

	data1	data2
key1		
a	-0.388863	0.260053
b	-0.791901	-0.084929

In [14]:

```
df.groupby(['key1', 'key2']).mean()
```

Out[14]:

	data1	data2
key1	key2	
a	one	-0.291498 0.369329
	two	-0.583593 0.041499
b	one	-1.378632 0.182173
	two	-0.205171 -0.352032

İlk durumda df.groupby ('key1'). Yani () sonuçta key2 sütunu olmadığını fark etmiş olabilirsiniz. Df ['key2'] sayısal veri olmadığından, sorunlu bir sütü

n olduğu söylenir ve bu nedenle sonuçtan çıkarılır. Varsayılan olarak, tüm sızısal sütunlar toplanır, ancak bir alt kümeye göre filtreleme yapmak da mümkün.

Groupby kullanmanın amacı ne olursa olsun, genel olarak kullanışlı bir GroupBy yöntemi boyuttur ve grup boyutlarını içeren bir Seri döndürür.

In [15]:

```
df.groupby(['key1', 'key2']).size()
```

Out[15]:

key1	key2	
a	one	2
	two	1
b	one	1
	two	1

dtype: int64

Bir grup anahtarındaki eksik değerlerin sonuçtan çıkarılacağını unutmayın.

Gruplar Üzerinde Yineleme

GroupBy nesnesi yinelemeyi destekler, grup adını ve veri yığısını içeren bir 2-tuple dizisi oluşturur. Aşağıdakileri göz önünde bulundur:

In [16]:

```
for name, group in df.groupby('key1'):
    print(name)
    print(group)
```

a	key1	key2	data1	data2
0	a	one	-0.399197	-1.392406
1	a	two	-0.583593	0.041499
4	a	one	-0.183798	2.131064
b	key1	key2	data1	data2
2	b	one	-1.378632	0.182173
3	b	two	-0.205171	-0.352032

Birden fazla anahtar olması durumunda, demetteki ilk öğe, anahtar değerlerinin bir demeti olacaktır.

In [17]:

```
for (k1, k2), group in df.groupby(['key1', 'key2']):
    print((k1, k2))
    print(group)
```

```
('a', 'one')
  key1 key2      data1      data2
0     a  one -0.399197 -1.392406
4     a  one -0.183798  2.131064
('a', 'two')
  key1 key2      data1      data2
1     a  two -0.583593  0.041499
('b', 'one')
  key1 key2      data1      data2
2     b  one -1.378632  0.182173
('b', 'two')
  key1 key2      data1      data2
3     b  two -0.205171 -0.352032
```

Elbette, veri parçalarıyla istediğiniz yapmayı seçebilirsiniz. Veri parçalarını bir sınırını tek satırlık olarak hesaplamaktır.

In [18]:

```
pieces = dict(list(df.groupby('key1')))
```

In [19]:

```
pieces['b']
```

Out[19]:

	key1	key2	data1	data2
2	b	one	-1.378632	0.182173
3	b	two	-0.205171	-0.352032

Varsayılan olarak, eksen = 0 üzerindeki grplara göre gruplandırılabilirsiniz, ancak diğer eksenlerin herhangi birinde gruplayabilirsiniz. Örneğin, burada df örneğimizin sütunlarını şu şekilde dtype ile gruplayabiliriz:

In [20]:

```
df.dtypes
```

Out[20]:

```
key1      object
key2      object
data1     float64
data2     float64
dtype: object
```

In [21]:

```
grouped = df.groupby(df.dtypes, axis=1)
```

Grupları şu şekilde yazdırabiliriz:

In [22]:

```
for dtype, group in grouped:
    print(dtype)
    print(group)
```

```
float64
      data1      data2
0 -0.399197 -1.392406
1 -0.583593  0.041499
2 -1.378632  0.182173
3 -0.205171 -0.352032
4 -0.183798  2.131064
object
      key1  key2
0      a   one
1      a   two
2      b   one
3      b   two
4      a   one
```

Sütun veya Sütun Alt Kümesini Seçme

Bir DataFrame'den bir sütun adı veya sütun adları dizisi ile oluşturulan bir GroupBy nesnesinin dizine alınması, toplama için sütun alt kümeleme etkisine sahiptir. Bunun anlamı şudur ki:

In [23]:

```
df.groupby('key1')['data1']
df.groupby('key1')[['data2']]
```

Out[23]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fe837e24cd0>
```

In [24]:

```
df['data1'].groupby(df['key1'])
df[['data2']].groupby(df['key1'])
```

Out[24]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fe836f308e0>
```

Özellikle büyük veri kümeleri için, yalnızca birkaç sütunun toplanması istenebilir. Örneğin, önceki veri kümesinde, yalnızca data2 sütunu için araçları

hesaplamak ve sonucu DataFrame olarak almak için şunu yazabiliriz:

In [25]:

```
df.groupby(['key1', 'key2'])[['data2']].mean()
```

Out[25]:

data2

key1	key2	data2
a	one	0.369329
	two	0.041499
b	one	0.182173
	two	-0.352032

Bu indeksleme işlemi tarafından döndürülen nesne, bir liste veya dizi geçirilirse gruplanmış bir DataFrame veya skaler olarak yalnızca tek bir sütun adı geçirilirse gruplanmış bir Seridir.

In [26]:

```
s_grouped = df.groupby(['key1', 'key2'])['data2']
```

In [27]:

```
s_grouped
```

Out[27]:

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x7fe837e24130>
```

In [28]:

```
s_grouped.mean()
```

Out[28]:

key1	key2	data2
a	one	0.369329
	two	0.041499
b	one	0.182173
	two	-0.352032

Name: data2, dtype: float64

Diktler ve Serilerle Gruplama

Gruplama bilgileri, bir diziden farklı bir biçimde olabilir.

In [29]:

```
people = pd.DataFrame(np.random.randn(5, 5),
                      columns=['a', 'b', 'c', 'd', 'e'],
                      index=['Joe', 'Steve', 'Wes', 'Jim', 'Travis'])
```

In [30]:

```
people.iloc[2:3, [1, 2]] = np.nan # Birkaç NA değeri ekleyin
```

In [31]:

```
people
```

Out[31]:

	a	b	c	d	e
Joe	-0.445252	0.557591	0.720730	-0.950052	-0.690401
Steve	0.598614	-0.280635	-0.311785	0.261914	-0.568365
Wes	-1.452790	NaN	NaN	-0.146601	-0.134484
Jim	0.227724	-0.507082	0.959123	-0.435780	-1.416611
Travis	0.904868	-1.395759	0.744964	1.073613	0.168507

Şimdi, sütunlar için bir grup yazışmam olduğumu ve sütunları gruptara göre toplamak istediğimi varsayıyalım.

In [32]:

```
mapping = {'a': 'red', 'b': 'red', 'c': 'blue',
           'd': 'blue', 'e': 'red', 'f': 'orange'}
```

Şimdi, bu dikteden groupby'ye geçmek için bir dizi oluşturabilirsiniz, ancak bunun yerine dikteyi geçebiliriz.

In [33]:

```
by_column = people.groupby(mapping, axis=1)
```

In [34]:

```
by_column.sum()
```

Out[34]:

	blue	red
Joe	-0.229322	-0.578062
Steve	-0.049870	-0.250385
Wes	-0.146601	-1.587274
Jim	0.523343	-1.695969
Travis	1.818577	-0.322384

Aynı işlevsellik, sabit boyutlu bir eşleme olarak görülebilen Seriler için de geçerlidir.

In [35]:

```
map_series = pd.Series(mapping)
```

In [36]:

```
map_series
```

Out[36]:

```
a      red
b      red
c    blue
d    blue
e      red
f  orange
dtype: object
```

In [37]:

```
people.groupby(map_series, axis=1).count()
```

Out[37]:

	blue	red
Joe	2	3
Steve	2	3
Wes	1	2
Jim	2	3
Travis	2	3

Fonksiyonları Gruplama

Python işlevlerini kullanmak, bir dikte veya Seriye kiyasla bir grup eşlemesini tanımlamanın daha genel bir yoludur. Grup anahtarı olarak iletilen herhangi bir işlev, her dizin değeri için bir kez çağrırlar ve dönüş değerleri grub adları olarak kullanılır. Daha somut olarak, bir önceki bölümde yer alan ve indeks değerleri olarak insanların adlarının bulunduğu DataFrame örneğini düşünün. Adların uzunluğuna göre grüplamak istedığınızı varsayıyalım; dizi uzunluklarını hesaplayabilseniz de, yalnızca len işlevini geçirmek daha kolaydır:

In [38]:

```
people.groupby(len).sum()
```

Out[38]:

	a	b	c	d	e
3	-1.670318	0.050508	1.679853	-1.532433	-2.241497
5	0.598614	-0.280635	-0.311785	0.261914	-0.568365
6	0.904868	-1.395759	0.744964	1.073613	0.168507

Her şey dahili olarak dizilere dönüştürüldüğünden, fonksiyonları diziler, diktler veya Seriler ile karıştırmak sorun olmaz.

In [39]:

```
key_list = ['one', 'one', 'one', 'two', 'two']
```

In [40]:

```
people.groupby([len, key_list]).min()
```

Out[40]:

	a	b	c	d	e
3	one	-1.452790	0.557591	0.720730	-0.950052
	two	0.227724	-0.507082	0.959123	-0.435780
5	one	0.598614	-0.280635	-0.311785	0.261914
6	two	0.904868	-1.395759	0.744964	1.073613
					0.168507

Dizin Düzeylerine Göre Gruplama

Hiyerarşik olarak indekslenmiş veri kümeleri için son bir kolaylık, bir ekse'n indeksinin seviyelerinden birini kullanarak bir araya getirme yeteneğidir. Bir örneğe bakalım:

In [41]:

```
columns = pd.MultiIndex.from_arrays([[ 'US', 'US', 'US', 'JP', 'JP'],
[1, 3, 5, 1, 3]],
names=[ 'cty', 'tenor'])
```

In [42]:

```
hier_df = pd.DataFrame(np.random.randn(4, 5), columns=columns)
```

In [43]:

```
hier_df
```

Out[43]:

cty	US			JP		
	tenor	1	3	5	1	3
0	-0.143661	0.669795	0.874048	1.175805	0.930104	
1	1.530062	-1.740579	-1.241398	-0.109193	0.268877	
2	-1.170875	-0.158375	-1.354285	-1.331365	0.226714	
3	-0.525477	-0.362784	-0.002955	-0.391789	0.028888	

Seviyeye göre grüplamak için, seviye anahtar kelimesini kullanarak seviye numarasını veya adını yazın.

In [44]:

```
hier_df.groupby(level='cty', axis=1).count()
```

Out[44]:

cty	JP	US
0	2	3
1	2	3
2	2	3
3	2	3

10.2 Veri Toplama

Toplamlar, dizilerden skaler değerler üreten herhangi bir veri dönüşümünü ifade eder. Önceki örnekler, ortalama, sayı, min ve toplam dahil olmak üzere birkaçını kullanmıştır. Bir GroupBy nesnesinde mean () çağrıdığınızda neler olduğunu merak edebilirsiniz. Tablo 10-1'de bulunanlar gibi birçok yaygın toplamanın optimize edilmiş uygulamaları vardır. Ancak, yalnızca bu yöntemle re sınırlı değilsiniz.

In [45]:

Image("img/picture73.png")

Out[45]:

Table 10-1. Optimized groupby methods

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

Kendi tasarıladığınız toplamaları kullanabilir ve ayrıca gruplanmış nesnede tanımlanan herhangi bir yöntemi de çağırabilirsiniz. Örneğin, nicel değerin bir Seri veya DataFrame sütunlarının örnek niceliklerini hesapladığı hatırlayabilirsiniz.

Quantile, GroupBy için açıkça uygulanmaya da, bir Series yöntemidir ve bu nedenle kullanılabilir. GroupBy dahili olarak, Serileri verimli bir şekilde dilimler, her parça için `piece.quantile (0.9)` 'u çağırır ve ardından bu sonuçları bir araya getirerek sonuç nesnesi oluşturur.

In [46]:

df

Out[46]:

	key1	key2	data1	data2
0	a	one	-0.399197	-1.392406
1	a	two	-0.583593	0.041499
2	b	one	-1.378632	0.182173
3	b	two	-0.205171	-0.352032
4	a	one	-0.183798	2.131064

In [47]:

```
grouped = df.groupby('key1')
```

In [48]:

```
grouped['data1'].quantile(0.9)
```

Out[48]:

```
key1
a   -0.226878
b   -0.322517
Name: data1, dtype: float64
```

Kendi toplama işlevlerinizi kullanmak için, bir diziyi toplayan herhangi bir işlevi toplama veya toplama yöntemine iletin.

In [49]:

```
def peak_to_peak(arr):
    return arr.max() - arr.min()
```

In [50]:

```
grouped.agg(peak_to_peak)
```

Out[50]:

	data1	data2
key1		
a	0.399794	3.523470
b	1.173461	0.534204

Tanımlama gibi bazı yöntemlerin, toplama olmasalar bile, kesinlikle işe yarıdığını fark edebilirsiniz.

In [51]:

```
grouped.describe()
```

Out[51]:

	data1					data2				
	count	mean	std	min	25%	50%	75%	max	count	i
key1										
a	3.0	-0.388863	0.200097	-0.583593	-0.491395	-0.399197	-0.291498	-0.183798	3.0	
b	2.0	-0.791901	0.829762	-1.378632	-1.085267	-0.791901	-0.498536	-0.205171	2.0	

Sütun Bazında ve Çok Fonksiyonlu Uygulama

Önceki örneklerden bahış veri kümesine dönelim. Read_csv ile yükledikten sonra, bir devrilme yüzdesi sütunu tip_pct ekliyoruz.

In [52]:

```
tips = pd.read_csv('examples/tips.csv')
```

In [53]:

```
# Toplam faturanın bahış yüzdesini ekleyin
tips['tip_pct'] = tips['tip'] / tips['total_bill']
```

In [54]:

```
tips[:6]
```

Out[54]:

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01		No	Sun	Dinner	0.059447
1	10.34	1.66		No	Sun	Dinner	0.160542
2	21.01	3.50		No	Sun	Dinner	0.166587
3	23.68	3.31		No	Sun	Dinner	0.139780
4	24.59	3.61		No	Sun	Dinner	0.146808
5	25.29	4.71		No	Sun	Dinner	0.186240

Daha önce gördüğünüz gibi, bir Seriyi veya bir DataFrame'in tüm sütunlarını toplamak, istenen işlevle toplamayı kullanma veya ortalama veya std gibi bir yöntemi çağırma meselesidir. Ancak, sütuna bağlı olarak farklı bir işlev veya aynı anda birden çok işlev kullanarak toplama yapmak isteyebilirsiniz. Neyse ki bunu yapmak mümkündür. İlk olarak, ipuçlarını güne ve sigara içen kişiye göre gruplayalım:

In [55]:

```
grouped = tips.groupby(['day', 'smoker'])
```

In [56]:

```
grouped_pct = grouped['tip_pct']
```

In [57]:

```
grouped_pct.agg('mean')
```

Out[57]:

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 bir işlevler veya işlev adları listesi iletirseniz, işlevlerden alınan sütun adlarıyla bir DataFrame geri alırsınız.

In [58]:

```
grouped_pct.agg(['mean', 'std', peak_to_peak])
```

Out[58]:

	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 gruplarını bağımsız olarak değerlendirmek için bir toplama işlevleri listesi geçirdik.

GroupBy'nin sütunlara verdiği adları kabul etmenize gerek yoktur; özellikle lambda işlevlerinin '`<lambda>`' adı vardır, bu da onları tanımlamayı zorlaştırır (bir işlevin `__name__` özniteliğine bakarak kendiniz görebilirsiniz). Bu nedenle, bir `(ad, işlev)` tuple listesini iletirseniz, her demetin ilk ögesi DataFrame sütun adları olarak kullanılacaktır (2-başlıkların bir listesini sıralı bir eşleme olarak düşünüebilirsiniz).

In [59]:

```
grouped_pct.agg([('foo', 'mean'), ('bar', np.std)])
```

Out[59]:

		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

Bir DataFrame ile, tüm sütunlara veya sütun başına farklı işlevlere uygulana cak bir işlev listesi belirleyebileceğiniz için daha fazla seçenekiniz vardır. Başlangıç olarak, tip_pct ve total_bill sütunları için aynı üç istatistiği hesaplamak istediğimizi varsayıyalım.

In [60]:

```
functions = ['count', 'mean', 'max']
```

In [61]:

```
result = grouped[['tip_pct', 'total_bill']].agg(functions)
```

<ipython-input-61-426117f1e146>:1: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.

```
result = grouped[['tip_pct', 'total_bill']].agg(functions)
```

In [62]:

result

Out[62]:

		tip_pct			total_bill		
		count	mean	max	count	mean	max
day	smoker						
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

Gördüğünüz gibi, sonuçta elde edilen DataFrame hiyerarşik sütunlara sahiptir, aynı her bir sütunu ayrı ayrı toplarsanız ve anahtarlar bağımsız değişken olarak sütun adlarını kullanarak sonuçları birbirine yapıştırmak için concat kullanırsınız.

In [63]:

```
result['tip_pct']
```

Out[63]:

		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

Daha önce olduğu gibi, özel adlara sahip bir tuples listesi aktarılabilir:

In [64]:

```
ftuples = [('Durchschnitt', 'mean'), ('Abweichung', np.var)]
```

In [65]:

```
grouped[['tip_pct', 'total_bill']].agg(ftuples)
```

<ipython-input-65-c2b2ffd79840>:1: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.

```
grouped[['tip_pct', 'total_bill']].agg(ftuples)
```

Out[65]:

		tip_pct		total_bill	
		Durchschnitt	Abweichung	Durchschnitt	Abweichung
day	smoker				
Fri	No	0.151650	0.000791	18.420000	25.596333
	Yes	0.174783	0.002631	16.813333	82.562438
Sat	No	0.158048	0.001581	19.661778	79.908965
	Yes	0.147906	0.003767	21.276667	101.387535
Sun	No	0.160113	0.001793	20.506667	66.099980
	Yes	0.187250	0.023757	24.120000	109.046044
Thur	No	0.160298	0.001503	17.113111	59.625081
	Yes	0.163863	0.001551	19.190588	69.808518

Şimdi, potansiyel olarak farklı işlevleri bir veya daha fazla sütuna uygulamak istediğinizizi varsayıyalım. Bunu yapmak için, şu ana kadar listelenen işlev özelliklerinden herhangi birine sütun adlarının bir eşlemesini içeren bir dict to agg iletin.

In [66]:

```
grouped.agg({'tip' : np.max, 'size' : 'sum'})
```

Out[66]:

		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 [67]:

```
grouped.agg({'tip_pct' : ['min', 'max', 'mean', 'std'],
              'size' : 'sum'})
```

Out[67]:

		tip_pct				size
		min	max	mean	std	sum
day	smoker					
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

Bir DataFrame, yalnızca en az bir sütuna birden çok işlev uygulandığında hiyerarşik sütunlara sahip olacaktır.

Satır Dizinleri Olmadan Toplu Verileri Döndürme

Şimdkiye kadarki tüm örneklerde, toplanan veriler, benzersiz grup anahtarları kombinasyonlarından oluşan potansiyel olarak hiyerarşik bir dizinle geri gelir. Bu her zaman arzu edilmediğinden, bu davranışını çoğu durumda groupby'ye as_index = False değerini ileterek devre dışı bırakabilirsiniz.

In [68]:

```
tips.groupby(['day', 'smoker'], as_index=False).mean()
```

Out[68]:

	day	smoker	total_bill	tip	size	tip_pct
0	Fri	No	18.420000	2.812500	2.250000	0.151650
1	Fri	Yes	16.813333	2.714000	2.066667	0.174783
2	Sat	No	19.661778	3.102889	2.555556	0.158048
3	Sat	Yes	21.276667	2.875476	2.476190	0.147906
4	Sun	No	20.506667	3.167895	2.929825	0.160113
5	Sun	Yes	24.120000	3.516842	2.578947	0.187250
6	Thur	No	17.113111	2.673778	2.488889	0.160298
7	Thur	Yes	19.190588	3.030000	2.352941	0.163863

Elbette, sonucu reset_index'i çağırarak bu biçimde elde etmek her zaman mümkündür. As_index = False yönteminin kullanılması bazı gereksiz hesaplamaları önler.

10.3 Uygula: Genel bölme-uygula-birleştir

Bu bölümün geri kalanının konusu olan en genel amaçlı GroupBy yöntemi uygulanır. Şekil 10-2'de gösterildiği gibi, uygula, işlenmekte olan nesneyi parçalara böler, her parçada geçirilen işlevi çağırır ve ardından parçaları birleştirmeye çalışır.

In [69]:

Image("img/picture74.png")

Out[69]:

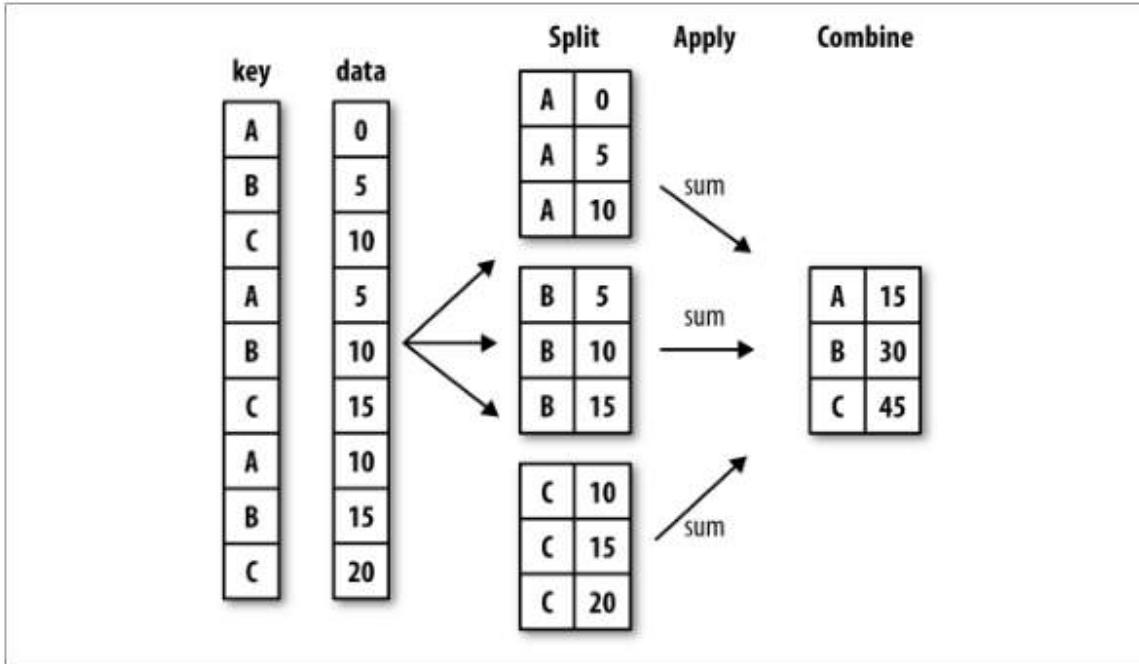


Figure 10-2. Illustration of a group aggregation

Önceki bahis veri kümesine dönersek, gruba göre ilk beş tip_pct değerini se çmek istediğinizizi varsayıyalım. İlk olarak, belirli bir sütundaki en büyük de ğerlere sahip satırları seçen bir işlev yazın.

In [70]:

```
def top(df, n=5, column='tip_pct'):
    return df.sort_values(by=column)[-n:]
```

In [71]:

```
top(tips, n=6)
```

Out[71]:

	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 içenlere göre gruplandırırsak, söylesek ve bu işlevle uygulayın çağırırsak, aşağıdakileri elde ederiz:

In [72]:

```
tips.groupby('smoker').apply(top)
```

Out[72]:

		total_bill	tip	smoker	day	time	size	tip_pct
	smoker							
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

Burada ne oldu? Top işlevi DataFrame'den her satır grubunda çağrıılır ve ardından sonuçlar pandas.concat kullanılarak birbirine yapıştırılır ve parçalar grup adlarıyla etiketlenir. Bu nedenle sonuç, iç seviyesi orijinal DataFrame'den dizin değerlerini içeren hiyerarşik bir dizine sahiptir.

Başka argümanlar veya anahtar sözcükler alan bir işlevi uygulamaya geçirirseniz, bunları işlevden sonra iletебilirsiniz.

In [73]:

```
tips.groupby(['smoker', 'day']).apply(top, n=1, column='total_bill')
```

Out[73]:

			total_bill	tip	smoker	day	time	size	tip_pct
smoker	day								
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 [74]:

```
result = tips.groupby('smoker')['tip_pct'].describe()
```

In [75]:

```
result
```

Out[75]:

	count	mean	std	min	25%	50%	75%	max
smoker								
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 [76]:

```
result.unstack('smoker')
```

Out[76]:

```
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
```

Grup içinde Tanımla gibi bir yöntemi çağırıldığınızda, bu aslında şunun kısaltmasıdır:

In [77]:

```
f = lambda x: x.describe()
grouped.apply(f)
```

Out[77]:

			total_bill	tip	size	tip_pct	
day	smoker	No	count	4.000000	4.000000	4.00	4.000000
Fri		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 Bastırma

Önceki örneklerde, ortaya çıkan nesnenin, orijinal nesnenin her bir parçasının dizinleri ile birlikte grup anahtarlarından oluşan bir hiyerarşik dizine sahip olduğunu görüyorsunuz. Bunu group_keys = False öğesini groupby'ye ile terek devre dışı bırakabilirsiniz.

In [78]:

```
tips.groupby('smoker', group_keys=False).apply(top)
```

Out[78]:

	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

Nicelik ve Bucket Analizi

In [79]:

```
frame = pd.DataFrame({'data1': np.random.randn(1000),
                      'data2': np.random.randn(1000)})
```

In [80]:

```
quartiles = pd.cut(frame.data1, 4)
```

In [81]:

```
quartiles[:10]
```

Out[81]:

```
0      (-0.121, 1.471]
1      (-1.713, -0.121]
2      (-0.121, 1.471]
3      (-1.713, -0.121]
4      (-0.121, 1.471]
5      (1.471, 3.063]
6      (-3.312, -1.713]
7      (-0.121, 1.471]
8      (-0.121, 1.471]
9      (-1.713, -0.121]
Name: data1, dtype: category
Categories (4, interval[float64]): [(-3.312, -1.713] < (-1.713, -0.121] < (-0.121, 1.471] < (1.471, 3.063]]
```

Kesmeyle döndürülen Kategorik nesne doğrudan groupby'ye aktarılabilir. Böylece, data2 sütunu için şu şekilde bir dizi istatistik hesaplayabiliriz.

In [82]:

```
def get_stats(group):
    return {'min': group.min(), 'max': group.max(),
            'count': group.count(), 'mean': group.mean()}
```

In [83]:

```
grouped = frame.data2.groupby(quartiles)
```

In [84]:

```
grouped.apply(get_stats).unstack()
```

Out[84]:

	min	max	count	mean
data1				
(-3.312, -1.713]	-2.035234	2.185491	35.0	0.107557
(-1.713, -0.121]	-3.317860	2.903081	402.0	-0.052531
(-0.121, 1.471]	-3.215114	2.457543	494.0	-0.022288
(1.471, 3.063]	-2.051406	2.529880	69.0	0.199493

In [85]:

```
grouping = pd.qcut(frame.data1, 10, labels=False)
```

In [86]:

```
grouped = frame.data2.groupby(grouping)
```

In [87]:

```
grouped.apply(get_stats).unstack()
```

Out[87]:

	min	max	count	mean
data1				
0	-2.040926	2.364725	100.0	0.079744
1	-2.322944	2.107722	100.0	-0.057665
2	-2.027871	2.903081	100.0	0.036003
3	-3.317860	2.201497	100.0	-0.167696
4	-2.103088	2.037555	100.0	-0.145220
5	-3.215114	2.287420	100.0	0.117786
6	-2.444351	2.295160	100.0	-0.109730
7	-1.423907	2.139066	100.0	-0.079800
8	-2.924371	2.457543	100.0	0.006001
9	-2.051406	2.529880	100.0	0.174594

Örnek: Eksik Değerleri Gruba Özgü Değerlerle Doldurma

Eksik verileri temizlerken, bazı durumlarda veri gözlemlerini dropna kullanarak değiştirirsiniz, ancak diğerlerinde boş (NA) değerleri sabit bir değer veya verilerden türetilen bir değer kullanarak atamak (doldurmak) isteyebilirsiniz. fillna, kullanılacak doğru araçtır; örneğin, burada NA değerlerini ortalamaya dolduralım.

In [88]:

```
s = pd.Series(np.random.randn(6))
```

In [89]:

```
s[::2] = np.nan
```

In [90]:

```
s
```

Out[90]:

```
0      NaN
1    0.387622
2      NaN
3   -1.140526
4      NaN
5   -0.772338
dtype: float64
```

In [91]:

```
s.fillna(s.mean())
```

Out[91]:

```
0    -0.508414
1     0.387622
2    -0.508414
3    -1.140526
4    -0.508414
5    -0.772338
dtype: float64
```

Gruba göre değiştirmek için doldurma değerine ihtiyacınız olduğunu varsayıyalım. Bunu yapmanın bir yolu, verileri grüplamak ve her veri yığınında `fillna`'yı çağırın bir işlevle uygula kullanmaktır. Doğu ve batı bölgelerine bölünmüş ABD eyaletleriyle ilgili bazı örnek veriler.

In [92]:

```
states = ['Ohio', 'New York', 'Vermont', 'Florida',
          'Oregon', 'Nevada', 'California', 'Idaho']
```

In [93]:

```
group_key = ['East'] * 4 + ['West'] * 4
```

In [94]:

```
data = pd.Series(np.random.randn(8), index=states)
```

In [95]:

```
data
```

Out[95]:

```
Ohio        0.175345
New York   -0.806473
Vermont    -0.546296
Florida    1.068155
Oregon     0.245295
Nevada     -0.957322
California 1.217878
Idaho      -0.238817
dtype: float64
```

`['Doğu'] * 4` sözdiziminin, `['Doğu']` 'daki öğelerin dört kopyasını içeren bir liste oluşturduğuna dikkat edin. Listeleri birbirine eklemek onları birleştirir.

Verilerdeki bazı değerleri eksik olarak ayarlayalım.

In [96]:

```
data[['Vermont', 'Nevada', 'Idaho']] = np.nan
```

In [97]:

```
data
```

Out[97]:

```
Ohio          0.175345
New York     -0.806473
Vermont       NaN
Florida       1.068155
Oregon        0.245295
Nevada        NaN
California    1.217878
Idaho         NaN
dtype: float64
```

In [98]:

```
data.groupby(group_key).mean()
```

Out[98]:

```
East      0.145676
West      0.731587
dtype: float64
```

NA değerlerini şu şekilde grup araçlarını kullanarak doldurabiliriz:

In [99]:

```
fill_mean = lambda g: g.fillna(g.mean())
```

In [100]:

```
data.groupby(group_key).apply(fill_mean)
```

Out[100]:

```
Ohio          0.175345
New York     -0.806473
Vermont       0.145676
Florida       1.068155
Oregon        0.245295
Nevada        0.731587
California    1.217878
Idaho         0.731587
dtype: float64
```

Başka bir durumda, kodunuzda gruba göre değişen önceden tanımlanmış dolgu değerleriniz olabilir. Grupların dahili olarak ayarlanmış bir ad öz niteliği olduğundan, bunu kullanabiliriz.

In [101]:

```
fill_values = {'East': 0.5, 'West': -1}
```

In [102]:

```
fill_func = lambda g: g.fillna(fill_values[g.name])
```

In [103]:

```
data.groupby(group_key).apply(fill_func)
```

Out[103]:

Ohio	0.175345
New York	-0.806473
Vermont	0.500000
Florida	1.068155
Oregon	0.245295
Nevada	-1.000000
California	1.217878
Idaho	-1.000000

dtype: float64

Örnek: Rastgele Örnekleme ve Permütasyon

Monte Carlo simülasyonu veya başka bir uygulama için büyük bir veri kümelerinde rastgele bir örnek (değiştirilerek veya değiştirilmeden) çekmek istediği zamanı varsayıyalım. "Çekilişleri" gerçekleştirmenin birkaç yolu vardır; burada Seri için örnek yöntemi kullanıyoruz.

In [104]:

```
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)

deck = pd.Series(card_val, index=cards)
```

Şimdi, indeksi kart adlarını içeren ve değerlerini Blackjack ve diğer oyunlarda kullanılan uzunluklu 52 bir Serimiz var (işleri basitleştirmek için sadece as 'A' 1 olsun).

In [105]:

```
deck[:13]
```

Out[105]:

```
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
```

In [106]:

```
def draw(deck, n=5):  
    return deck.sample(n)
```

In [107]:

```
draw(deck)
```

Out[107]:

```
KH      10  
QS      10  
5H      5  
4C      4  
8H      8  
dtype: int64
```

Her takımdan iki rastgele kart istedığınızı varsayalım. Takım, her kart adının son karakteri olduğu için, buna göre gruplayabilir ve uygulayabiliriz.

In [108]:

```
get_suit = lambda card: card[-1]
```

In [109]:

```
deck.groupby(get_suit).apply(draw, n=2)
```

Out[109]:

C	8C	8
	5C	5
D	3D	3
	JD	10
H	6H	6
	10H	10
S	9S	9
	7S	7

dtype: int64

Alternatif olarak şunları yazabiliriz:

In [110]:

```
deck.groupby(get_suit, group_keys=False).apply(draw, n=2)
```

Out[110]:

5C	5
3C	3
9D	9
7D	7
KH	10
8H	8
2S	2
5S	5

dtype: int64

Örnek: Grup Ağırlıklı Ortalama ve Korelasyon

Gruplandırmamanın bölme-uygulama-birleştirme paradigmı altında, bir DataFrame'deki sütunlar veya bir grup ağırlıklı ortalama gibi iki Seri arasında işlemler mümkündür. Örnek olarak, grup anahtarlarını, değerlerini ve bazı ağırlıkları içeren bu veri kümесini alın.

In [111]:

```
import pandas as pd
import numpy as np
df = pd.DataFrame({'category': ['a', 'a', 'a', 'a',
                                 'b', 'b', 'b', 'b'],
                    'data': np.random.randn(8),
                    'weights': np.random.rand(8)})
```

In [112]:

df

Out[112]:

	category	data	weights
0	a	0.586089	0.054759
1	a	0.707313	0.852027
2	a	1.699787	0.824191
3	a	1.136030	0.878106
4	b	-1.745034	0.812521
5	b	-2.010998	0.897366
6	b	-1.096052	0.016913
7	b	0.528596	0.945391

Kategoriye göre grup ağırlıklı ortalama şu şekilde olacaktır:

In [113]:

grouped = df.groupby('category')

In [114]:

get_wavg = lambda g: np.average(g['data'], weights=g['weights'])

In [115]:

grouped.apply(get_wavg)

Out[115]:

```
category
a    1.162572
b   -1.025857
dtype: float64
```

Başka bir örnek olarak, orijinal olarak Yahoo! Birkaç hisse senedi ve S&P 500 endeksi (SPX simgesi) için gün sonu fiyatlarını içeren finans:

In [116]:

close_px = pd.read_csv('examples/stock_px_2.csv', parse_dates=True, index_col=0)

In [117]:

```
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 [118]:

```
close_px[-4:]
```

Out[118]:

	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

İlgili duyulan bir görev, SPX ile günlük getirilerin (yüzde değişimlerden hesaplanan) yıllık korelasyonlarından oluşan bir DataFrame'i hesaplamak olabilir. Bunu yapmanın bir yolu olarak, önce her bir sütunun 'SPX' sütunuyla ikili korelasyonunu hesaplayan bir işlev oluşturuyoruz.

In [119]:

```
spx_corr = lambda x: x.corrwith(x['SPX'])
```

Daha sonra, pct_change kullanarak close_px'deki yüzde değişimini hesaplıyoruz.

In [120]:

```
rets = close_px.pct_change().dropna()
```

Son olarak, her bir tarih saat etiketinin year özelliğini döndüren tek satırlık bir fonksiyonla her satır etiketinden çıkarılabilen bu yüzde değişikliklerini yıla göre grupperediriyoruz.

In [121]:

```
get_year = lambda x: x.year
```

In [122]:

```
by_year = rets.groupby(get_year)
```

In [123]:

```
by_year.apply(spx_corr)
```

Out[123]:

	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 ve Microsoft arasındaki yıllık ilişkiye bakıyoruz.

In [124]:

```
by_year.apply(lambda g: g['AAPL'].corr(g['MSFT']))
```

Out[124]:

```
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

Önceki örnekle aynı temada, işlev bir pandas nesnesi veya skaler değer döndürdüğü sürece, grup bazında daha karmaşık istatistiksel analiz gerçekleştirmek için groupby'yi kullanabilirsiniz. Örneğin, her veri yığınında sıradan bir en küçük kareler (OLS) regresyonu uygulayan aşağıdaki regress fonksiyonunu (istatistik modelleri ekonometri kitabılığını kullanarak) tanımlayabilirsiniz.

In [125]:

```
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
```

Şimdi, SPX iadelerinde AAPL'nin yıllık doğrusal regresyonunu çalıştmak için şunu çalıştırın:

In [126]:

```
by_year.apply(regress, 'AAPL', ['SPX'])
```

Out[126]:

	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

10.4 Pivot Tablolar ve Çapraz Tablolar

Pivot tablo, elektronik tablo programlarında ve diğer veri analizi yazılımlarında sıkılıkla bulunan bir veri özetleme aracıdır. Bir veri tablosunu bir ve ya daha fazla anahtarla toplar, verileri grup anahtarlarının bazıları satırlar boyunca ve bazıları da sütunlar boyunca olacak şekilde bir dikdörtgen şeklinde düzenler. Python ile pandalarla pivot tablolar, hiyerarşik indeksleme kullanan yeniden şekillendirme işlemleriyle birlikte bu bölümde açıklanan groupby özelliği ile mümkün hale getirilir. DataFrame'in bir pivot_table yöntemi vardır ve ayrıca üst düzey bir pandas.pivot_table işlevi vardır. Pivot_table, groupby için kullanışlı bir arayüz sağlamanın yanı sıra, kenar boşlukları olarak da bilinen kısmi toplamlar ekleyebilir.

Bahşiş veri kümesine dönersek, güne göre düzenlenmiş ve satırlarda sigara içen bir grup araçları tablosu (varsayılan pivot_table toplama türü) hesaplama istedığınızı varsayıyalım.

In [127]:

```
tips.pivot_table(index=['day', 'smoker'])
```

Out[127]:

		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 üretilmiş olabilir. Şimdi, sadece tip_pct ve size ve ayrıca zamana göre grüplamak istedığımızı varsayıyalım. Sigara içenleri tablo sütunlarına ve günü satırlara koyalım.

In [128]:

```
tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
                 columns='smoker')
```

Out[128]:

		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` yazarak bu tabloyu kısmi toplamları içerecek şekilde genişletebiliriz. Bu, Tüm satır ve sütun etiketlerini ekleme etkisine sahiptir; karşılık gelen değerler, tek bir katmandaki tüm verilerin grup istatistikleri olur.

In [129]:

```
tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
                 columns='smoker', margins=True)
```

Out[129]:

		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

Burada, Tüm değerler, sigara içen ile içmeyen (Tüm sütunlar) veya satırlarda ki iki gruplandırma düzeyinden herhangi biri (Tümü satırı) hesaba katılmadan ortalamalarıdır.

Farklı bir toplama işlevi kullanmak için bunu toplama işlevine aktarın. Örneğin, 'count' veya len size grup boyutlarının çapraz tablolarını (sayım veya sıklığı) verecektir.

In [130]:

```
tips.pivot_table('tip_pct', index=['time', 'smoker'], columns='day',
                 aggfunc=len, margins=True)
```

Out[130]:

		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 yazmak isteyebilirsiniz.

In [131]:

```
tips.pivot_table('tip_pct', index=['time', 'size', 'smoker'],
                 columns='day', aggfunc='mean', fill_value=0)
```

Out[131]:

		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
	6	No	0.000000	0.000000	0.103799	0.000000
		Yes	0.000000	0.000000	0.000000	0.181728
Lunch	1	No	0.000000	0.000000	0.000000	0.000000
		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

Pivot_table yöntemlerinin özeti için Tablo 10-2'ye bakın.

In [132]:

```
from IPython.display import Image
Image("img/picture75.png")
```

Out[132]:

Table 10-2. pivot_table options

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

Çapraz tablo (veya kısaca çapraz tablo), grup frekanslarını hesaplayan özel bir pivot tablodur. İşte bir örnek:

In [133]:

data

Out[133]:

```
Ohio          0.175345
New York     -0.806473
Vermont       NaN
Florida       1.068155
Oregon        0.245295
Nevada        NaN
California   1.217878
Idaho         NaN
dtype: float64
```

Bazı anket analizlerinin bir parçası olarak, bu verileri milliyet ve teslimi yet açısından özetlemek isteyebiliriz. Bunu yapmak için pivot_table'ı kullanabilirsiniz, ancak pandas.crosstab işlevi daha uygun olabilir.

```
pd.crosstab(data.Nationality, data.Handedness, margins=True)
```

Handedness Nationality	Left-handed	Right-handed	All
Japan	2	3	5
USA	1	4	5
All	3	7	10

Çapraz tablonun ilk iki bağımsız değişkeninin her biri bir dizi veya Seri ve ya bir dizi listesi olabilir.

In [134]:

```
pd.crosstab([tips.time, tips.day], tips.smoker, margins=True)
```

Out[134]:

	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

Time Series

Zaman serisi verileri, finans, ekonomi, ekoloji, sinirbilim ve fizik gibi birçok farklı alanda önemli bir yapılandırılmış veri biçimidir. Zamanın birçok noktasında gözlemlenen veya ölçülen her şey bir zaman serisi oluşturur. Çoğu zaman serisi sabit frekanstır, yani veri noktalarının her 15 saniyede bir, her 5 dakikada bir veya ayda bir gibi bazı kurallara göre düzenli aralıklarla meydana geldiği anlamına gelir. Zaman serileri, sabit bir zaman birimi veya birimler arasında fark olmaksızın da düzensiz olabilir. Zaman serisi verilerini nasıl işaretlediğiniz ve bunlara atıfta bulunduğuuz uygulamaya bağlıdır ve aşağıdakilerden birine sahip olabilirsiniz:

- Zaman damgaları, zamandaki belirli anlar
- Ocak 2007 veya 2010 yılının tamamı gibi sabit dönemler
- Başlangıç ve bitiş zaman damgası ile gösterilen zaman aralıkları. Dönemle, özel aralık durumları olarak düşünülebilir
- Deney veya geçen süre; her zaman damgası, belirli bir başlangıç zamanına göre bir zaman ölçüsüdür (örneğin, fırına yerleştirildikten sonra her saniye pişen bir kurabiyenin çapı)

Pandas, 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 ve parçalara ayıracı, bir araya getirebilir ve yeniden örnekleyebilirsiniz. Bu araçlardan bazıları özellikle finans ve ekonomi uygulamaları için kullanışlıdır, ancak bunları kesinlikle sunucu günlüğü verilerini analiz etmek için de kullanabilirsiniz.

11.1 Date and Time Data Types and Tools

Python standart kitaplığı, tarih ve saat verileri için veri türlerinin yanı sıra takvimle ilgili işlevselligi içerir. Tarih saat, saat ve takvim modülleri, başlanacak ana yerlerdir. Datetime.datetime türü veya kısaca datetime türü yaygın olarak kullanılır.

In [1]:

```
from datetime import datetime
```

In [2]:

```
now = datetime.now()
```

In [3]:

```
now
```

Out[3]:

```
datetime.datetime(2020, 12, 8, 22, 1, 12, 953932)
```

In [4]:

```
now.year, now.month, now.day
```

Out[4]:

```
(2020, 12, 8)
```

datetime hem tarihi hem de saatı 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
```

Bir datetime nesnesine bir timedelta veya birden fazla ekleyebilir (veya çıkış artabilirsiniz).

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)
```

Tablo 11-1, datetime modülündeki veri türlerini özetlemektedir.

In [13]:

```
from IPython.display import Image
Image("img/picture76.png")
```

Out[13]:

Table 11-1. Types in datetime module

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

Dize ve Tarih Saat Arasında Dönüşüm

Daha sonra tanıtabileceğim datetime nesnelerini ve pandas Timestamp nesnelerini, bir biçim belirtimini ileterek str veya strftime yöntemini kullanarak dizer olarak biçimlendirebilirsiniz.

In [14]:

```
stamp = datetime(2011, 1, 3)
```

In [15]:

`str(stamp)`

Out[15]:

'2011-01-03 00:00:00'

In [16]:

`stamp.strftime('%Y-%m-%d')`

Out[16]:

'2011-01-03'

In [17]:

`Image("img/picture77.png")`

Out[17]:

Table 11-2. Datetime format specification (ISO C89 compatible)

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)

Date time.strptime kullanarak dizeleri tarihlere dönüştürmek için aynı biçim kodlarını kullanabilirsiniz.

In [18]:

`value = '2011-01-03'`

In [19]:

```
datetime.strptime(value, '%Y-%m-%d')
```

Out[19]:

```
datetime.datetime(2011, 1, 3, 0, 0)
```

In [20]:

```
datestrs = ['7/6/2011', '8/6/2011']
```

In [21]:

```
[datetime.strptime(x, '%m/%d/%Y') for x in datestrs]
```

Out[21]:

```
[datetime.datetime(2011, 7, 6, 0, 0), datetime.datetime(2011, 8, 6, 0, 0)]
```

`datetime.strptime`, bilinen bir biçimde sahip bir tarihi ayırtımanın iyi bir yoludur. Ancak, özellikle yaygın tarih formatları için her seferinde bir format spesifikasyonu yazmak zorunda kalmak biraz can sıkıcı olabilir. Bu durumda, üçüncü taraf `dateutil` paketindeki `parser.parse` yöntemini kullanabilirsiniz.

In [22]:

```
from dateutil.parser import parse
```

In [23]:

```
parse('2011-01-03')
```

Out[23]:

```
datetime.datetime(2011, 1, 3, 0, 0)
```

`dateutil`, insan tarafından anlaşılır tarih temsillerinin çoğunu ayırtırabilir.

In [24]:

```
parse('Jan 31, 1997 10:45 PM')
```

Out[24]:

```
datetime.datetime(1997, 1, 31, 22, 45)
```

Uluslararası yerel ayarlarda, aydan önce görünen gün çok yaygındır, bu nedenle bunu belirtmek için `dayfirst = True` seçeneğini geçebilirsiniz.

In [25]:

```
parse('6/12/2011', dayfirst=True)
```

Out[25]:

```
datetime.datetime(2011, 12, 6, 0, 0)
```

Pandalar genellikle, ister eksen dizini olarak ister bir DataFrame'de bir sü tun olarak kullanılsın, tarih dizileriyle çalışmaya yönelikdir. To_datetime yöntemi birçok farklı tarih temsilini ayırtırır. ISO 8601 gibi standart ta rih biçimleri çok hızlı bir şekilde ayırtırılabilir.

In [26]:

```
datestrs = ['2011-07-06 12:00:00', '2011-08-06 00:00:00']
```

In [27]:

```
import pandas as pd
import numpy as np
pd.to_datetime(datestrs)
```

Out[27]:

```
DatetimeIndex(['2011-07-06 12:00:00', '2011-08-06 00:00:00'], dtype='datetime64[ns]', freq=None)
```

Eksik olarak kabul edilmesi gereken değerleri de işler.

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])
```

datetime nesneleri ayrıca diğer ülkelerdeki veya dillerdeki sistemler için bir dizi yerel ayara özgü biçimlendirme seçeneğine sahiptir. Örneğin, kısaltılmış ay adları İngilizce sistemlere kıyasla Almanca veya Fransızca sistemlerde farklı olacaktır. Liste için Tablo 11-3'e bakın.

In [32]:

```
Image("img/picture78.png")
```

Out[32]:

Table 11-3. Locale-specific date formatting

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')

11.2 Zaman Serisi Temelleri

Pandas temel bir tür zaman serisi nesnesi, zaman damgaları tarafından indekslenen ve genellikle pandaların dışında Python dizeleri veya datetime nesnele ri olarak temsil edilen bir Seridir.

In [33]:

```
from datetime import datetime
```

In [34]:

```
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 [35]:

```
ts = pd.Series(np.random.randn(6), index=dates)
```

In [36]:

```
ts
```

Out[36]:

```
2011-01-02    -0.167618
2011-01-05    -0.139078
2011-01-07    -0.955696
2011-01-08    -0.830644
2011-01-10    -1.236246
2011-01-12    -0.726282
dtype: float64
```

Başlık altında, bu datetime nesneleri bir DatetimeIndex değerine yerleştirilmiştir.

In [37]:

```
ts.index
```

Out[37]:

```
DatetimeIndex(['2011-01-02', '2011-01-05', '2011-01-07', '2011-01-08',
                 '2011-01-10', '2011-01-12'],
                dtype='datetime64[ns]', freq=None)
```

Diğer Seriler gibi, farklı şekilde indekslenmiş zaman serileri arasındaki aritmetik işlemler tarihlere otomatik olarak hizalanır.

In [38]:

```
ts + ts[::2]
```

Out[38]:

```
2011-01-02    -0.335236
2011-01-05      NaN
2011-01-07   -1.911393
2011-01-08      NaN
2011-01-10   -2.472491
2011-01-12      NaN
dtype: float64
```

Ts [:: 2] 'nin ts cinsinden her ikinci elemanı seçtiğini hatırlayın.

Pandas, Nanosaniyede NumPy'nin datetime64 veri türünü kullanarak zaman damgarını depolar

Çözüm:

In [39]:

```
ts.index.dtype
```

Out[39]:

```
dtype('M8[ns]')
```

DatetimeIndex'teki skaler değerler pandaların Timestamp nesneleridir.

In [40]:

```
stamp = ts.index[0]
```

In [41]:

```
stamp
```

Out[41]:

```
Timestamp('2011-01-02 00:00:00')
```

Bir Zaman Damgası, datetime nesnesini kullanacağınız herhangi bir yerde değiştirilebilir. Ek olarak, frekans bilgilerini (varsı) depolayabilir ve saat dilimi dönüşümlerinin ve diğer türden işlemlerin nasıl yapılacağını anlar.

İndeksleme, Seçim, Alt Kümeleme

In [42]:

```
stamp = ts.index[2]
```

In [43]:

```
ts[stamp]
```

Out[43]:

```
-0.9556964358265075
```

Kolaylık sağlamak için, tarih olarak yorumlanabilen bir dize de iletебilirsiniz.

In [44]:

```
ts['1/10/2011']
```

Out[44]:

```
-1.2362455523428892
```

In [45]:

ts['20110110']

Out[45]:

-1.2362455523428892

Daha uzun zaman serileri için, veri dilimlerini kolayca seçmek için bir yıl veya yalnızca bir yıl ve ay geçilebilir.

In [46]:

```
longer_ts = pd.Series(np.random.randn(1000),
                      index=pd.date_range('1/1/2000', periods=1000))
```

In [47]:

longer_ts

Out[47]:

2000-01-01	0.591602
2000-01-02	0.048478
2000-01-03	-1.819338
2000-01-04	0.790559
2000-01-05	0.446850
...	
2002-09-22	-0.827081
2002-09-23	-0.647824
2002-09-24	0.107585
2002-09-25	-1.903806
2002-09-26	-0.152246

Freq: D, Length: 1000, dtype: float64

In [48]:

longer_ts['2001']

Out[48]:

2001-01-01	0.451057
2001-01-02	-1.051888
2001-01-03	-0.077032
2001-01-04	0.957333
2001-01-05	-0.800397
...	
2001-12-27	1.436362
2001-12-28	-0.758598
2001-12-29	-1.679734
2001-12-30	-0.396577
2001-12-31	0.308144

Freq: D, Length: 365, dtype: float64

Burada, '2001' dizisi bir yıl olarak yorumlanır ve bu zaman periyodunu seçer. Bu, ayı belirtirseniz de işe yarar.

In [49]:

```
longer_ts['2001-05']
```

Out[49]:

```
2001-05-01    2.069407
2001-05-02    0.474534
2001-05-03   -0.498526
2001-05-04    1.394368
2001-05-05   -1.533434
2001-05-06   -0.429792
2001-05-07    1.430014
2001-05-08   -0.674394
2001-05-09    0.132574
2001-05-10   -0.519418
2001-05-11    1.503926
2001-05-12    1.604574
2001-05-13   -0.158515
2001-05-14    0.035668
2001-05-15   -0.327299
2001-05-16    1.115009
2001-05-17   -0.305507
2001-05-18   -0.662547
2001-05-19   -0.261305
2001-05-20    1.535290
2001-05-21    0.206806
2001-05-22   -0.312340
2001-05-23    0.237264
2001-05-24    0.966096
2001-05-25   -0.944367
2001-05-26    0.275228
2001-05-27    1.667173
2001-05-28    0.042950
2001-05-29   -0.293588
2001-05-30   -1.078286
2001-05-31   -0.427982
Freq: D, dtype: float64
```

Datetime nesneleriyle dilimleme de işe yarar.

In [50]:

```
ts[datetime(2011, 1, 7):]
```

Out[50]:

```
2011-01-07   -0.955696
2011-01-08   -0.830644
2011-01-10   -1.236246
2011-01-12   -0.726282
dtype: float64
```

Çoğu zaman serisi verisi kronolojik olarak sıralandığından, bir aralık sorusu gerçekleştirmek için bir zaman serisinde yer almayan zaman damgalarıyla dilimleme yapabilirsiniz.

In [51]:

ts['1/6/2011':'1/11/2011']

Out[51]:

```
2011-01-07    -0.955696
2011-01-08    -0.830644
2011-01-10    -1.236246
dtype: float64
```

Daha önce olduğu gibi, bir dize tarihi, tarih saatı veya zaman damgası geçir ebilirsiniz. Bu şekilde dilimlemenin, NumPy dizilerini dilimlemek gibi kayna k zaman serileri üzerinde görünümler oluşturduğunu unutmayın. Bu, hiçbir ver inin kopyalanmayacağı ve dilim üzerindeki değişikliklerin orijinal verilere yansıtılacağı anlamına gelir.

Bir Seriyi iki tarih arasında dilimleyen eşdeğer bir örnek yöntemi vardır: truncate:

In [52]:

ts.truncate(after='1/9/2011')

Out[52]:

```
2011-01-02    -0.167618
2011-01-05    -0.139078
2011-01-07    -0.955696
2011-01-08    -0.830644
dtype: float64
```

Tüm bunlar DataFrame için de geçerlidir ve satırları üzerinde indeksleme yap ar.

In [53]:

dates = pd.date_range('1/1/2000', periods=100, freq='W-WED')

In [54]:

```
long_df = pd.DataFrame(np.random.randn(100, 4),
                      index=dates,
                      columns=['Colorado', 'Texas',
                               'New York', 'Ohio'])
```

long_df.loc['5-2001']

Yinelenen Endeksli Zaman Serileri

Bazı uygulamalarda, belirli bir zaman damgasına düşen birden çok veri gözlemi olabilir. İşte bir örnek:

In [55]:

```
dates = pd.DatetimeIndex(['1/1/2000', '1/2/2000', '1/2/2000',
                           '1/2/2000', '1/3/2000'])
```

In [56]:

```
dup_ts = pd.Series(np.arange(5), index=dates)
```

In [57]:

```
dup_ts
```

Out[57]:

```
2000-01-01    0
2000-01-02    1
2000-01-02    2
2000-01-02    3
2000-01-03    4
dtype: int64
```

`is_unique` özelliğini kontrol ederek indeksin benzersiz olmadığını söyleyebiliriz.

In [58]:

```
dup_ts.index.is_unique
```

Out[58]:

```
False
```

Bu zaman serisine indeksleme, bir zaman damgasının yinelenip yinelenmediğine bağlı olarak artık ya skaler değerler ya da dilimler üretecektir.

In [59]:

```
dup_ts['1/3/2000'] # kopyalanmamış
```

Out[59]:

```
4
```

In [60]:

```
dup_ts['1/2/2000'] # kopyalandı
```

Out[60]:

```
2000-01-02    1
2000-01-02    2
2000-01-02    3
dtype: int64
```

Benzersiz olmayan zaman damgalarına sahip verileri toplamak istedığınızı var sayalım. Bunu yapmanın bir yolu groupby kullanmak ve pass level=0:

In [61]:

```
grouped = dup_ts.groupby(level=0)
```

In [62]:

```
grouped.mean()
```

Out[62]:

```
2000-01-01    0
2000-01-02    2
2000-01-03    4
dtype: int64
```

In [63]:

```
grouped.count()
```

Out[63]:

```
2000-01-01    1
2000-01-02    3
2000-01-03    1
dtype: int64
```

11.3 Tarih Aralıkları, Frekanslar ve Kaydırma

Pandas jenerik zaman serilerinin düzensiz olduğu varsayıılır; yani sabit bir frekansları yoktur. Birçok uygulama için bu yeterlidir. Bununla birlikte, bir zaman serisine eksik değerler eklemek anlamına gelse bile, genellikle günlük, aylık veya 15 dakikada bir gibi sabit bir sıklığa göre çalışmak istenir. Neyse ki pandalar, yeniden örneklemeye, frekanslar çıkarma ve sabit frekanslı tarih aralıkları oluşturma için eksiksiz bir standart zaman serisi frekansları ve araçları paketine sahiptir. Örneğin, resample işlevini çağırarak örnek zaman serilerini sabit günlük sıklığa dönüştürebilirsiniz.

In [64]:

ts

Out[64]:

```
2011-01-02    -0.167618
2011-01-05    -0.139078
2011-01-07    -0.955696
2011-01-08    -0.830644
2011-01-10    -1.236246
2011-01-12    -0.726282
dtype: float64
```

In [65]:

resampler = ts.resample('D')

'D' dizisi günlük sıklık olarak yorumlanır.

Tarih Aralıkları Oluşturma

`pandas.date_range`, belirli bir frekansa göre belirtilen uzunlukta bir DatetimeIndex oluşturmaktan sorumludur.

In [66]:

index = pd.date_range('2012-04-01', '2012-06-01')

In [67]:

index

Out[67]:

```
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')
```

Varsayılan olarak `tarih_aralığı`, günlük zaman damgaları oluşturur. Yalnızca

bir başlangıç veya bitiş tarihi geçerseniz, oluşturmak için birkaç dönem gerekeniz gerekir.

In [68]:

```
pd.date_range(start='2012-04-01', periods=20)
```

Out[68]:

```
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', '2012-04-2
0'],
                dtype='datetime64[ns]', freq='D')
```

In [69]:

```
pd.date_range(end='2012-06-01', periods=20)
```

Out[69]:

```
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', '2012-06-0
1'],
                dtype='datetime64[ns]', freq='D')
```

Başlangıç ve bitiş tarihleri, oluşturulan tarih dizini için katı sınırlar tanımlar. Örneğin, her ayın son iş gününü içeren bir tarih endeksi istiyorsanız, 'BM' sıklığını (ayın iş sonu; Tablo 11-4'te daha eksiksiz frekanslar listesine bakın) ve yalnızca veya tarih aralığının içi dahil edilecektir.

In [70]:

```
pd.date_range('2000-01-01', '2000-12-01', freq='BM')
```

Out[70]:

```
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')
```

In [71]:

Image("img/picture79.png")

Out[71]:

Table 11-4. Base time series frequencies (not comprehensive)

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

date_range varsayılan olarak başlangıç veya bitiş zaman damgasının saatini (varsa) korur.

In [72]:

```
pd.date_range('2012-05-02 12:56:31', periods=5)
```

Out[72]:

```
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')
```

Bazen saat bilgileriyle başlangıç veya bitiş tarihleriniz olur, ancak bir kural olarak gece yarısına normalleştirilmiş bir dizi zaman damgası oluşturmak isteyebilirsiniz. Bunu yapmak için bir normalleştirme seçeneği var.

In [73]:

```
pd.date_range('2012-05-02 12:56:31', periods=5, normalize=True)
```

Out[73]:

```
DatetimeIndex(['2012-05-02', '2012-05-03', '2012-05-04', '2012-05-05',
                 '2012-05-06'],
                dtype='datetime64[ns]', freq='D')
```

Frekanslar ve Tarih Farkları

Pandalardaki frekanslar, bir temel frekans ve bir çarptan oluşan. Temel frekanslar tipik olarak, aylık için 'M' veya saatlik için 'H' gibi bir dize türkmen adıyla belirtilir. Her temel frekans için, genellikle tarih ofseti olarak adlandırılan tanımlanmış bir nesne vardır. Örneğin, saatlik sıkılık, Saat sınıfı ile temsil edilebilir.

In [74]:

```
from pandas.tseries.offsets import Hour, Minute
```

In [75]:

```
hour = Hour()
```

In [76]:

```
hour
```

Out[76]:

```
<Hour>
```

Bir tamsayı ileterek bir ofsetin katını tanımlayabilirsiniz.

In [77]:

```
four_hours = Hour(4)
```

In [78]:

```
four_hours
```

Out[78]:

```
<4 * Hours>
```

Çoğu uygulamada, 'H' veya '4H' gibi bir dize takma adı kullanmak yerine, bu nesnelerden birini açıkça oluşturmanız gerekmektedir. Temel frekanstan önce bir tamsayı koymak bir kat oluşturur.

In [79]:

```
pd.date_range('2000-01-01', '2000-01-03 23:59', freq='4h')
```

Out[79]:

```
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 ekleyerek birleştirilebilir.

In [80]:

```
Hour(2) + Minute(30)
```

Out[80]:

```
<150 * Minutes>
```

Benzer şekilde, etkin bir şekilde aynı ifadeye ayırtılacak "1s30dak" gibi frekans dizelerini iletебilirsiniz.

In [81]:

```
pd.date_range('2000-01-01', periods=10, freq='1h30min')
```

Out[81]:

```
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')
```

Bazı frekanslar zaman içinde eşit aralıklı olmayan noktaları tanımlar. Örneğin, 'A' (takvim ayı sonu) ve 'BM' (son iş / ayın hafta içi günü) bir aydaki gün sayısına ve ikinci durumda ayın hafta sonunda bitip bitmediğine bağlıdır. Bunları bağlantılı offsetler olarak adlandırıyoruz.

Ayların haftası tarihleri

Yararlı bir sıklık sınıfı, WOM ile başlayan "ayın haftası"dır. Bu, her ayın üçüncü Cuma günü gibi tarihler almanızı sağlar.

In [82]:

```
rng = pd.date_range('2012-01-01', '2012-09-01', freq='WOM-3FRI')
```

In [83]:

```
list(rng)
```

Out[83]:

```
[Timestamp('2012-01-20 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-02-17 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-03-16 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-04-20 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-05-18 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-06-15 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-07-20 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-08-17 00:00:00', freq='WOM-3FRI')]
```

Kaydırma (Öncü ve Gecikme) Verileri

"Kaydırma", verilerin zaman içinde geriye ve ileriye taşınması anlamına gelir. Hem Series hem de DataFrame, ileriye veya geriye doğru saf geçişler yapmak için bir kaydırma yöntemine sahiptir ve dizini değiştirmeden bırakır.

In [84]:

```
ts = pd.Series(np.random.randn(4),
               index=pd.date_range('1/1/2000', periods=4, freq='M'))
```

In [85]:

```
ts
```

Out[85]:

```
2000-01-31    -0.352418
2000-02-29     0.463366
2000-03-31    -0.398486
2000-04-30    -1.149113
Freq: M, dtype: float64
```

In [86]:

```
ts.shift(2)
```

Out[86]:

```
2000-01-31      NaN
2000-02-29      NaN
2000-03-31    -0.352418
2000-04-30     0.463366
Freq: M, dtype: float64
```

In [87]:

```
ts.shift(-2)
```

Out[87]:

```
2000-01-31    -0.398486
2000-02-29    -1.149113
2000-03-31      NaN
2000-04-30      NaN
Freq: M, dtype: float64
```

Bu şekilde değiştığımızda, eksik veriler zaman serisinin başında veya sonundan ortaya çıkar.

Kaymanın yaygın bir kullanımı, DataFrame sütunları olarak bir zaman serisindeki veya çoklu zaman serilerindeki yüzde değişiklikleri hesaplamaktır. Bu şekilde ifade edilir:

In [88]:

```
ts / ts.shift(1) - 1
```

Out[88]:

```
2000-01-31      NaN
2000-02-29    -2.314819
2000-03-31    -1.859981
2000-04-30     1.883698
Freq: M, dtype: float64
```

Saf kaymalar indeksi değiştirmeden bıraktığı için bazı veriler atılır. Böylece

ce, frekans biliniyorsa, basitçe veriler yerine zaman damgalarını ilerletmek için kaydırma yapabilir.

In [89]:

```
ts.shift(2, freq='M')
```

Out[89]:

```
2000-03-31    -0.352418
2000-04-30     0.463366
2000-05-31    -0.398486
2000-06-30    -1.149113
Freq: M, dtype: float64
```

In [90]:

```
ts.shift(3, freq='D')
```

Out[90]:

```
2000-02-03    -0.352418
2000-03-03     0.463366
2000-04-03    -0.398486
2000-05-03    -1.149113
dtype: float64
```

In [91]:

```
ts.shift(1, freq='90T')
```

Out[91]:

```
2000-01-31 01:30:00    -0.352418
2000-02-29 01:30:00     0.463366
2000-03-31 01:30:00    -0.398486
2000-04-30 01:30:00    -1.149113
Freq: M, dtype: float64
```

Kaydırma ile tarihleri değiştirmeye:

In [92]:

```
from pandas.tseries.offsets import Day, MonthEnd
```

In [93]:

```
now = datetime(2011, 11, 17)
```

In [94]:

```
now + 3 * Day()
```

Out[94]:

```
Timestamp('2011-11-20 00:00:00')
```

MonthEnd gibi bağlantılı bir offset eklerseniz, ilk artış, sıklık kuralına göre bir tarihi sonraki tarihe "ileri alır".

In [95]:

```
now + MonthEnd()
```

Out[95]:

```
Timestamp('2011-11-30 00:00:00')
```

In [96]:

```
now + MonthEnd(2)
```

Out[96]:

```
Timestamp('2011-12-31 00:00:00')
```

Bağlantılı uzaklıklar, sırasıyla geri alma ve geri alma yöntemlerini kullanarak tarihleri açıkça ileri veya geri "yuvarlayabilir".

In [97]:

```
offset = MonthEnd()
```

In [98]:

```
offset.rollforward(now)
```

Out[98]:

```
Timestamp('2011-11-30 00:00:00')
```

In [99]:

```
offset.rollback(now)
```

Out[99]:

```
Timestamp('2011-10-31 00:00:00')
```

```
Timestamp('2011-10-31 00:00:00')
```

Tarih offsetlerinin bir kullanımı da, şu yöntemleri groupby ile kullanmaktadır:

In [100]:

```
ts = pd.Series(np.random.randn(20),
               index=pd.date_range('1/15/2000', periods=20, freq='4d'))
```

In [101]:

ts

Out[101]:

```
2000-01-15    -0.333279
2000-01-19     2.311530
2000-01-23     0.182716
2000-01-27     1.192940
2000-01-31     1.753089
2000-02-04     1.421851
2000-02-08     0.268416
2000-02-12     0.588780
2000-02-16     1.817362
2000-02-20    -0.187322
2000-02-24    -0.319280
2000-02-28    -1.174072
2000-03-03    -0.489599
2000-03-07     1.427535
2000-03-11    -0.321368
2000-03-15    -1.005643
2000-03-19     0.191880
2000-03-23     0.386196
2000-03-27     0.694630
2000-03-31    -0.157693
Freq: D, dtype: float64
```

In [102]:

ts.groupby(offset.rollforward).mean()

Out[102]:

```
2000-01-31     1.021399
2000-02-29     0.345105
2000-03-31     0.090742
dtype: float64
```

In [103]:

ts.resample('M').mean()

Out[103]:

```
2000-01-31     1.021399
2000-02-29     0.345105
2000-03-31     0.090742
Freq: M, dtype: float64
```

11.4 Saat Dilimi İşleme

Zaman dilimleriyle çalışmak, genellikle zaman serisi manipülasyonunun en rahatsız edici kısımlarından biri olarak kabul edilir. Sonuç olarak, birçok zaman serisi kullanıcısı, Greenwich Ortalama Zamanının halefi olan ve mevcut uluslararası standart olan eşgüdümü evrensel saatte zaman serileriyle çalışma'yı sefer. Saat dilimleri, UTC'den farklar olarak ifade edilir; örneğin, New York yaz saati uygulaması sırasında UTC'nin dört saat gerisinde ve yılın geri kalanında beş saat geridedir.

Python'da, saat dilimi bilgileri, dünya saat dilimi bilgilerinin bir derlemesi olan Olson veritabanını ortaya çıkaran üçüncü taraf pytz kitaplığından (pip veya conda ile kurulabilir) gelir. Bu, özellikle tarihsel veriler için önemlidir, çünkü gün ışığından yararlanma saati (DST) geçiş tarihleri (ve hatta UTC farkları) yerel yönetimlerin kaprislerine bağlı olarak birçok kez değiştirilmiştir. Birleşik Devletler'de, DST geçiş süreleri 1900'den beri birçok kez değiştirildi!

Pytz kitabı hakkında ayrıntılı bilgi için bu kitaplığın belgelerine bakmanız gereklidir. Bu kitap söz konusu olduğunda, pandalar pytz'in işlevsellliğini sarmalar, böylece saat dilimi adlarının dışındaki API'sini göz ardı edebilirsiniz. Saat dilimi adları etkileşimli olarak ve aşağıdaki belgelerde bulunabilir.

In [104]:

```
import pytz
```

In [105]:

```
pytz.common_timezones[-5:]
```

Out[105]:

```
['US/Eastern', 'US/Hawaii', 'US/Mountain', 'US/Pacific', 'UTC']
```

Pytz'den bir saat dilimi nesnesi almak için pytz.timezone kullanın.

In [106]:

```
tz = pytz.timezone('America/New_York')
```

In [107]:

```
tz
```

Out[107]:

```
<DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>
```

Pandalar'daki yöntemler, saat dilimi adlarını veya bu nesneleri kabul eder.

Saat Dilimi Yerelleştirme ve Dönüştürme

Varsayılan olarak, pandas'daki zaman serileri yerel saat dilimidir. Örneğin, aşağıdaki zaman serilerini düşünün:

In [108]:

```
import pandas as pd
import numpy as np
rng = pd.date_range('3/9/2012 9:30', periods=6, freq='D')
```

In [109]:

```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
```

In [110]:

```
ts
```

Out[110]:

```
2012-03-09 09:30:00    -0.178143
2012-03-10 09:30:00    -0.857762
2012-03-11 09:30:00     0.779415
2012-03-12 09:30:00    -0.083602
2012-03-13 09:30:00    -0.116134
2012-03-14 09:30:00     2.037980
Freq: D, dtype: float64
```

In [111]:

```
print(ts.index.tz)
```

```
None
```

Tarih aralıkları, bir saat dilimi ayarıyla oluşturulabilir.

In [112]:

```
pd.date_range('3/9/2012 9:30', periods=10, freq='D', tz='UTC')
```

Out[112]:

```
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')
```

In [113]:

```
ts
```

Out[113]:

```
2012-03-09 09:30:00    -0.178143
2012-03-10 09:30:00    -0.857762
2012-03-11 09:30:00     0.779415
2012-03-12 09:30:00    -0.083602
2012-03-13 09:30:00    -0.116134
2012-03-14 09:30:00     2.037980
Freq: D, dtype: float64
```

In [114]:

```
ts_utc = ts.tz_localize('UTC')
```

In [115]:

```
ts_utc
```

Out[115]:

```
2012-03-09 09:30:00+00:00    -0.178143
2012-03-10 09:30:00+00:00    -0.857762
2012-03-11 09:30:00+00:00     0.779415
2012-03-12 09:30:00+00:00    -0.083602
2012-03-13 09:30:00+00:00    -0.116134
2012-03-14 09:30:00+00:00     2.037980
Freq: D, dtype: float64
```

In [116]:

```
ts_utc.index
```

Out[116]:

```
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 saat dilimine yerleştirildiğinde, `tz_convert` ile başka bir saat dilimine dönüştürülebilir.

In [117]:

```
ts_utc.tz_convert('America/New_York')
```

Out[117]:

```
2012-03-09 04:30:00-05:00 -0.178143
2012-03-10 04:30:00-05:00 -0.857762
2012-03-11 05:30:00-04:00 0.779415
2012-03-12 05:30:00-04:00 -0.083602
2012-03-13 05:30:00-04:00 -0.116134
2012-03-14 05:30:00-04:00 2.037980
Freq: D, dtype: float64
```

America / New_York saat diliminde bir DST geçişini destekleyen önceki zaman serileri durumunda, EST'ye yerelleştirebilir ve örneğin UTC veya Berlin saatine dönüştürebiliriz.

In [118]:

```
ts_eastern = ts.tz_localize('America/New_York')
```

In [119]:

```
ts_eastern.tz_convert('UTC')
```

Out[119]:

```
2012-03-09 14:30:00+00:00 -0.178143
2012-03-10 14:30:00+00:00 -0.857762
2012-03-11 13:30:00+00:00 0.779415
2012-03-12 13:30:00+00:00 -0.083602
2012-03-13 13:30:00+00:00 -0.116134
2012-03-14 13:30:00+00:00 2.037980
Freq: D, dtype: float64
```

In [120]:

```
ts_eastern.tz_convert('Europe/Berlin')
```

Out[120]:

```
2012-03-09 15:30:00+01:00 -0.178143
2012-03-10 15:30:00+01:00 -0.857762
2012-03-11 14:30:00+01:00 0.779415
2012-03-12 14:30:00+01:00 -0.083602
2012-03-13 14:30:00+01:00 -0.116134
2012-03-14 14:30:00+01:00 2.037980
Freq: D, dtype: float64
```

`tz_localize` ve `tz_convert`, DatetimeIndex'eki örnek yöntemleridir.

In [121]:

```
ts.index.tz_localize('Asia/Shanghai')
```

Out[121]:

```
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='D')
```

Saat Dilimi ile İşlemler – Aware Zaman Damgası Nesneleri

Zaman serileri ve tarih aralıklarına benzer şekilde, ayrı Zaman Damgası nesneleri de benzer şekilde basitten saat dilimine göre yerelleştirilebilir ve bir saat diliminden diğerine dönüştürülebilir.

In [122]:

```
stamp = pd.Timestamp('2011-03-12 04:00')
```

In [123]:

```
stamp_utc = stamp.tz_localize('utc')
```

In [124]:

```
stamp_utc.tz_convert('America/New_York')
```

Out[124]:

```
Timestamp('2011-03-11 23:00:00-0500', tz='America/New_York')
```

Zaman Damgasını oluştururken de bir saat dilimi yazabilirsiniz.

In [125]:

```
stamp_moscow = pd.Timestamp('2011-03-12 04:00', tz='Europe/Moscow')
```

In [126]:

```
stamp_moscow
```

Out[126]:

```
Timestamp('2011-03-12 04:00:00+0300', tz='Europe/Moscow')
```

Saat dilimine duyarlı Timestamp nesneleri, Unix döneminden (1 Ocak 1970) bu yana bir UTC zaman damgası değerini dahili olarak nano saniye olarak depolar; bu UTC değeri, saat dilimi dönüşümleri arasında değişmez.

In [127]:

```
stamp_utc.value
```

Out[127]:

```
12999024000000000000
```

In [128]:

```
stamp_utc.tz_convert('America/New_York').value
```

Out[128]:

```
12999024000000000000
```

Pandas'ın DateOffset nesnelerini kullanarak zaman aritmetiği gerçekleştirirken, pandas mümkün olduğunda gün ışığından yararlanma saatini geçişlerine uyar. Burada, DST geçişlerinden hemen önce (ileri ve geri) oluşan zaman damgaları oluşturuyoruz. İlk olarak, DST'ye geçmeden 30 dakika önce.

In [129]:

```
from pandas.tseries.offsets import Hour
```

In [130]:

```
stamp = pd.Timestamp('2012-03-12 01:30', tz='US/Eastern')
```

In [131]:

```
stamp
```

Out[131]:

```
Timestamp('2012-03-12 01:30:00-0400', tz='US/Eastern')
```

In [132]:

```
stamp + Hour()
```

Out[132]:

```
Timestamp('2012-03-12 02:30:00-0400', tz='US/Eastern')
```

Ardından, DST'den çıkmadan 90 dakika önce:

In [133]:

```
stamp = pd.Timestamp('2012-11-04 00:30', tz='US/Eastern')
```

In [134]:

stamp

Out[134]:

Timestamp('2012-11-04 00:30:00-0400', tz='US/Eastern')

In [135]:

stamp + 2 * Hour()

Out[135]:

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. Zaman damgaları UTC'de başlık altında saklandığından, bu basit bir işlemidir ve herhangi bir dönüşüm yapılmasını gerektirmez.

In [136]:

rng = pd.date_range('3/7/2012 9:30', periods=10, freq='B')

In [137]:

ts = pd.Series(np.random.randn(len(rng)), index=rng)

In [138]:

ts

Out[138]:

2012-03-07 09:30:00	-0.078952
2012-03-08 09:30:00	-1.509267
2012-03-09 09:30:00	0.216251
2012-03-12 09:30:00	0.294584
2012-03-13 09:30:00	0.043956
2012-03-14 09:30:00	0.502256
2012-03-15 09:30:00	-0.252668
2012-03-16 09:30:00	-1.026749
2012-03-19 09:30:00	-0.549725
2012-03-20 09:30:00	0.037510

Freq: B, dtype: float64

In [139]:

ts1 = ts[:7].tz_localize('Europe/London')

In [140]:

ts2 = ts1[2:].tz_convert('Europe/Moscow')

In [141]:

```
result = ts1 + ts2
```

In [142]:

```
result.index
```

Out[142]:

```
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='B')
```

11.5 Periyotlar ve Periyot Aritmetiği

Dönemler, günler, aylar, çeyrekler veya yıllar gibi zaman aralıklarını temsil eder. Period sınıfı, bir dize veya tamsayı ve Tablo 11-4'ten bir sıklık gerektiren bu veri türünü temsil eder.

In [143]:

```
p = pd.Period(2007, freq='A-DEC')
```

In [144]:

```
p
```

Out[144]:

```
Period('2007', 'A-DEC')
```

Bu durumda, Dönem nesnesi 1 Ocak 2007 ile 31 Aralık 2007 arasındaki tam zaman aralığını temsil eder. Elverişli bir şekilde, tam sayıları dönemlerden eklemek ve çıkarmak, sıklıklarına göre değişme etkisine sahiptir.

In [145]:

```
p+5
```

Out[145]:

```
Period('2012', 'A-DEC')
```

In [146]:

p-2

Out[146]:

Period('2005', 'A-DEC')

İki dönemin aynı frekansı varsa, aralarındaki birim sayısı farkıdır.

In [147]:

pd.Period('2014', freq='A-DEC') - p

Out[147]:

<7 * YearEnds: month=12>

Düzenli dönem aralıkları, period_range işlevi ile oluşturulabilir.

In [148]:

rng = pd.period_range('2000-01-01', '2000-06-30', freq='M')

In [149]:

rng

Out[149]:

PeriodIndex(['2000-01', '2000-02', '2000-03', '2000-04', '2000-05', '2000-06'], dtype='period[M]', freq='M')

PeriodIndex sınıfı, bir dönem dizisini depolar ve herhangi bir pandas veri yapısında bir eksen dizini görevi görebilir.

In [150]:

pd.Series(np.random.randn(6), index=rng)

Out[150]:

2000-01	-0.856101
2000-02	0.818814
2000-03	-0.643821
2000-04	1.393822
2000-05	0.151194
2000-06	0.979933

Freq: M, dtype: float64

Bir dizi dizenz varsa, PeriodIndex sınıfını da kullanabilirsiniz.

In [151]:

```
values = ['2001Q3', '2002Q2', '2003Q1']
```

In [152]:

```
index = pd.PeriodIndex(values, freq='Q-DEC')
```

In [153]:

```
index
```

Out[153]:

```
PeriodIndex(['2001Q3', '2002Q2', '2003Q1'], dtype='period[Q-DEC]', freq='Q-DEC')
```

Periyot Frekans Dönüşümü

Periyotlar ve PeriodIndex nesneleri asfreq yöntemi ile başka bir frekansa dönüştürülebilir. Örnek olarak, bir yıllık dönemimiz olduğunu ve bunu yılın başında veya sonunda aylık bir döneme dönüştürmek istediğimizi varsayalım.

In [154]:

```
p = pd.Period('2007', freq='A-DEC')
```

In [155]:

```
p
```

Out[155]:

```
Period('2007', 'A-DEC')
```

In [156]:

```
p.asfreq('M', how='start')
```

Out[156]:

```
Period('2007-01', 'M')
```

In [157]:

```
p.asfreq('M', how='end')
```

Out[157]:

```
Period('2007-12', 'M')
```

Dönemi ('2007', 'A-DEC') aylık dönemlere bölünmüş bir zaman aralığını gösteren bir tür imleç olarak düşünebilirsiniz. Bunun bir açıklaması için Şekil 11-1'e bakın. Aralık dışında bir ayda sona eren bir mali yıl için, karşılık gelen aylık alt dönemler farklıdır.

In [158]:

```
p = pd.Period('2007', freq='A-JUN')
```

In [159]:

```
p
```

Out[159]:

```
Period('2007', 'A-JUN')
```

In [160]:

```
p.asfreq('M', 'start')
```

Out[160]:

```
Period('2006-07', 'M')
```

In [161]:

```
p.asfreq('M', 'end')
```

Out[161]:

```
Period('2007-06', 'M')
```

In [162]:

```
from IPython.display import Image
Image("img/picture80.png")
```

Out[162]:

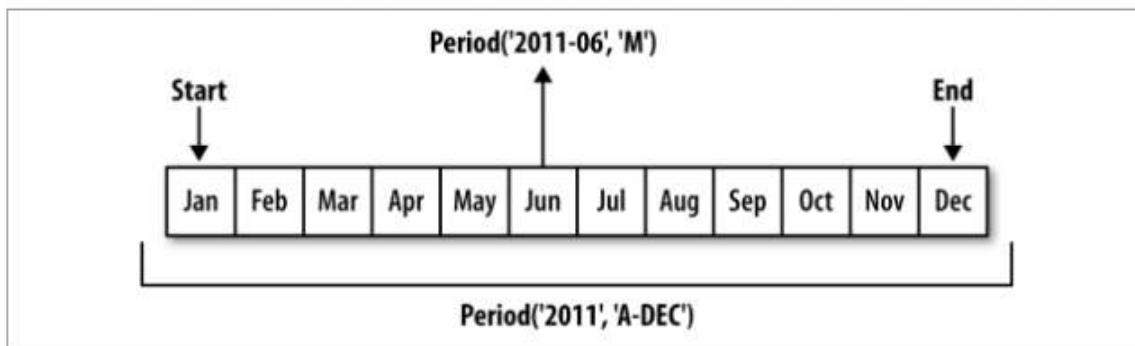


Figure 11-1. Period frequency conversion illustration

Yüksek frekanstan düşük frekansa geçiş yaparken pandalar, alt periyodun "ait olduğu" yere bağlı olarak süper dönemi belirler. Örneğin, A-JUN sıklığında, Ağustos-2007 ayı aslında 2008 döneminin bir parçasıdır.

In [163]:

```
p = pd.Period('Aug-2007', 'M')
```

In [164]:

```
p.asfreq('A-JUN')
```

Out[164]:

```
Period('2008', 'A-JUN')
```

Tüm PeriodIndex nesneleri veya zaman serileri, aynı semantikle benzer şekilde dönüştürülebilir.

In [165]:

```
rng = pd.period_range('2006', '2009', freq='A-DEC')
```

In [166]:

```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
```

In [167]:

```
ts
```

Out[167]:

2006	0.451662
2007	1.939921
2008	1.433454
2009	1.907176

Freq: A-DEC, dtype: float64

In [168]:

```
ts.asfreq('M', how='start')
```

Out[168]:

2006-01	0.451662
2007-01	1.939921
2008-01	1.433454
2009-01	1.907176

Freq: M, dtype: float64

Burada yıllık dönemler, her bir yıllık döneme giren ilk aya karşılık gelen aylık dönemlerle değiştirilir. Bunun yerine her yılın son iş gününü istersek, 'B' sıklığını kullanabilir ve dönemin sonunu istediğimizi belirtebiliriz.

In [169]:

```
ts.asfreq('B', how='end')
```

Out[169]:

```
2006-12-29    0.451662
2007-12-31    1.939921
2008-12-31    1.433454
2009-12-31    1.907176
Freq: B, dtype: float64
```

Üç Aylık Dönem Frekansları

Üç aylık veriler muhasebe, finans ve diğer alanlarda standarttır. Çeyreklik verilerin çoğu, mali yıl sonuna göre rapor edilir, tipik olarak yılın 12 ayından birinin son takvimi veya iş günüdür. Dolayısıyla 2012Q4 dönemi mali yıl sonuna bağlı olarak farklı bir anlama sahiptir. Pandalar, Q-JAN ile Q-DEC ile 12 olası üç aylık frekansı destekler.

In [170]:

```
p = pd.Period('2012Q4', freq='Q-JAN')
```

In [171]:

```
p
```

Out[171]:

```
Period('2012Q4', 'Q-JAN')
```

2012 yılının Ocak ayında sona eren mali yıl durumunda, 4. Çeyrek Kasım'dan Ocak'a kadar devam eder ve bunu günlük sıklığı dönüştürerek kontrol edebilirsiniz. Bir gösterim için Şekil 11-2. ye bakın.

In [172]:

```
Image("img/picture81.png")
```

Out[172]:

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			

Figure 11-2. Different quarterly frequency conventions

In [173]:

```
p.asfreq('D', 'start')
```

Out[173]:

```
Period('2011-11-01', 'D')
```

In [174]:

```
p.asfreq('D', 'end')
```

Out[174]:

```
Period('2012-01-31', 'D')
```

Böylece, kolay dönem aritmetiği yapmak mümkünür; örneğin, çeyreğin ikinci i la son iş gününde saat 16: 00'da zaman damgası almak için şunları yapabilirsiniz:

In [175]:

```
p4pm = (p.asfreq('B', 'e') - 1).asfreq('T', 's') + 16 * 60
```

In [176]:

```
p4pm
```

Out[176]:

```
Period('2012-01-30 16:00', 'T')
```

In [177]:

```
p4pm.to_timestamp()
```

Out[177]:

```
Timestamp('2012-01-30 16:00:00')
```

`Period_range` kullanarak üç aylık aralıklar oluşturabilirsiniz. Aritmetik de aynıdır.

In [178]:

```
rng = pd.period_range('2011Q3', '2012Q4', freq='Q-JAN')
```

In [179]:

```
ts = pd.Series(np.arange(len(rng)), index=rng)
```

In [180]:

ts

Out[180]:

```
2011Q3      0
2011Q4      1
2012Q1      2
2012Q2      3
2012Q3      4
2012Q4      5
Freq: Q-JAN, dtype: int64
```

In [181]:

```
new_rng = (rng.asfreq('B', 'e') - 1).asfreq('T', 's') + 16 * 60
```

In [182]:

```
ts.index = new_rng.to_timestamp()
```

Zaman Damgalarını Dönemlere (ve Geri) Dönüştürme

Zaman damgalarına göre indekslenen Series ve DataFrame nesneleri, to_period yöntemiyle dönemlere dönüştürülebilir.

In [183]:

```
rng = pd.date_range('2000-01-01', periods=3, freq='M')
```

In [184]:

```
ts = pd.Series(np.random.randn(3), index=rng)
```

In [185]:

ts

Out[185]:

```
2000-01-31    0.921619
2000-02-29   -1.238203
2000-03-31    0.203244
Freq: M, dtype: float64
```

In [186]:

```
pts = ts.to_period()
```

In [187]:

pts

Out[187]:

```
2000-01    0.921619
2000-02   -1.238203
2000-03    0.203244
Freq: M, dtype: float64
```

Dönemler, çakışmayan zaman aralıklarına atıfta bulunduğuundan, bir zaman damga ası belirli bir sıklık için yalnızca tek bir döneme ait olabilir. Yeni PeriodIndex sıklığı varsayılan olarak zaman damgalarından çıkarılırken, istediğiniz herhangi bir sıklığı belirtebilirsiniz. Ayrıca sonuçta yinelenen dönemler olmasına ilgili bir sorun yoktur.

In [188]:

```
rng = pd.date_range('1/29/2000', periods=6, freq='D')
```

In [189]:

```
ts2 = pd.Series(np.random.randn(6), index=rng)
```

In [190]:

ts2

Out[190]:

```
2000-01-29    1.420410
2000-01-30    0.007182
2000-01-31    0.197205
2000-02-01   -0.483437
2000-02-02   -0.384188
2000-02-03   -1.199151
Freq: D, dtype: float64
```

In [191]:

```
ts2.to_period('M')
```

Out[191]:

```
2000-01    1.420410
2000-01    0.007182
2000-01    0.197205
2000-02   -0.483437
2000-02   -0.384188
2000-02   -1.199151
Freq: M, dtype: float64
```

Zaman damgalarına geri dönmek için to_timestamp kullanın.

In [192]:

```
pts = ts2.to_period()
```

In [193]:

```
pts
```

Out[193]:

```
2000-01-29    1.420410
2000-01-30    0.007182
2000-01-31    0.197205
2000-02-01   -0.483437
2000-02-02   -0.384188
2000-02-03   -1.199151
Freq: D, dtype: float64
```

In [194]:

```
pts.to_timestamp(how='end')
```

Out[194]:

```
2000-01-29 23:59:59.999999999    1.420410
2000-01-30 23:59:59.999999999    0.007182
2000-01-31 23:59:59.999999999    0.197205
2000-02-01 23:59:59.999999999   -0.483437
2000-02-02 23:59:59.999999999   -0.384188
2000-02-03 23:59:59.999999999   -1.199151
Freq: D, dtype: float64
```

Dizilerden PeriodIndex Oluşturma

Sabit frekans veri kümeleri bazen birden çok sütuna yayılmış zaman aralığı b
ilgileriyle depolanır. Örneğin, bu makroekonomik veri setinde yıl ve üç aylı
k dönem farklı sütunlardadır.

In [195]:

```
data = pd.read_csv('examples/macrodata.csv')
```

In [196]:

```
data.head(5)
```

Out[196]:

	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbillrate	unemp
0	1959.0	1.0	2710.349	1707.4	286.898	470.045	1886.9	28.98	139.7	2.82	5.8
1	1959.0	2.0	2778.801	1733.7	310.859	481.301	1919.7	29.15	141.7	3.08	5.1
2	1959.0	3.0	2775.488	1751.8	289.226	491.260	1916.4	29.35	140.5	3.82	5.3
3	1959.0	4.0	2785.204	1753.7	299.356	484.052	1931.3	29.37	140.0	4.33	5.6
4	1960.0	1.0	2847.699	1770.5	331.722	462.199	1955.5	29.54	139.6	3.50	5.2

In [197]:

```
data.year
```

Out[197]:

```
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 [198]:

```
data.quarter
```

Out[198]:

```
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, DataFrame için bir dizin oluşturmak üzere bunları birleştirebilirsiniz.

In [199]:

```
index = pd.PeriodIndex(year=data.year, quarter=data.quarter,
                       freq='Q-DEC')
```

In [200]:

index

Out[200]:

```
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, freq='Q-DEC')
```

In [201]:

data.index = index

In [202]:

data.infl

Out[202]:

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

11.6 Yeniden Örnekleme ve Frekans Dönüşürme

Yeniden örnekleme, bir zaman serisini bir frekanstan diğerine dönüştürme sürecini ifade eder. Daha yüksek frekanslı verilerin daha düşük frekansa toplanması alt yeniden örneklemeye, düşük frekansın daha yüksek frekansa dönüştürülmesine yukarı örneklemeye adı verilir. Tüm yeniden örneklem bu kategorilerden hiçbirine girmez; örneğin, W-WED'i (Çarşamba günleri haftalık) W-FRI'ye dönüştürmek ne yukarı örneklem ne de alt örneklemidir.

Pandas nesneleri, tüm frekans dönüşümleri için en önemli işlev olan yeniden örneklem yöntemiyle donatılmıştır. resample, groupby'ye benzer bir API'ye sahiptir; verileri grüplamak için yeniden örneklem, ardından bir toplama fonksiyonu çağırırsınız.

In [203]:

```
rng = pd.date_range('2000-01-01', periods=100, freq='D')
```

In [204]:

```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
```

In [205]:

```
ts
```

Out[205]:

```
2000-01-01    2.219496
2000-01-02    1.333854
2000-01-03   -0.910499
2000-01-04    0.656197
2000-01-05   -0.047516
...
2000-04-05    0.549141
2000-04-06    1.930335
2000-04-07   -0.517340
2000-04-08    1.738335
2000-04-09    0.412617
Freq: D, Length: 100, dtype: float64
```

In [206]:

```
ts.resample('M').mean()
```

Out[206]:

```
2000-01-31   -0.026347
2000-02-29    0.303219
2000-03-31   -0.002111
2000-04-30    0.470735
Freq: M, dtype: float64
```

In [207]:

```
ts.resample('M', kind='period').mean()
```

Out[207]:

```
2000-01   -0.026347
2000-02    0.303219
2000-03   -0.002111
2000-04    0.470735
Freq: M, dtype: float64
```

resample, çok büyük zaman serilerini işlemek için kullanılabilen esnek ve yüksek performanslı bir yöntemdir. Aşağıdaki bölümlerdeki örnekler, anlamını ve kullanımını göstermektedir. Tablo 11-5, bazı seçeneklerini özetlemektedir.

In [208]:

Image("img/picture82.png")

Out[208]:

Table 11-5. Resample method arguments

Argument	Description
<code>freq</code>	String or DateOffset indicating desired resampled frequency (e.g., 'M', '5min', or <code>Second(15)</code>)
<code>axis</code>	Axis to resample on; default axis=0
<code>fill_method</code>	How to interpolate when upsampling, as in 'ffill' or 'bfill'; by default does no interpolation
<code>closed</code>	In downsampling, which end of each interval is closed (inclusive), 'right' or 'left'
<code>label</code>	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)
<code>loffset</code>	Time adjustment to the bin labels, such as '-1s' / <code>Second(-1)</code> to shift the aggregate labels one second earlier
<code>limit</code>	When forward or backward filling, the maximum number of periods to fill
<code>kind</code>	Aggregate to periods ('period') or timestamps ('timestamp'); defaults to the type of index the time series has
<code>convention</code>	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 düzeltilemesine gerek yoktur; istenen frekans, zaman serilerini kümelenmek üzere parçalara ayırmak için kullanılan bölme kenarlarını tanımlar. Örneğin, aylık "A" veya "BM" ye dönüş türmek için, verileri bir aylık aralıklara bölməniz gerekdir. Her aralığın yarı açık olduğu söylenir; bir veri noktası yalnızca bir aralığa ait olabilir ve aralıkların birleşimi tüm zaman çerçevesini oluşturmalıdır. Verileri alt örnekleme için yeniden örnekleme kullanırken düşünülmeli gereken birkaç nokta vardır.

- Her aralığın hangi tarafı kapalıdır
- Aralığın başlangıcı veya sonu ile birleştirilmiş her bölme nasıl etiketlenir

Örnek olarak, bir dakikalık verilere bakalım.

In [209]:

rng = pd.date_range('2000-01-01', periods=12, freq='T')

In [210]:

```
ts = pd.Series(np.arange(12), index=rng)
```

In [211]:

```
ts
```

Out[211]:

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: int64

Her grubun toplamını alarak bu verileri beş dakikalık parçalar veya çubuklar halinde toplamak istediğiniz varsayıyalım.

In [212]:

```
ts.resample('5min', closed='right').sum()
```

Out[212]:

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: int64

Geçtiğiniz frekans, bölüm kenarlarını beş dakikalık artışlarla tanımlar. Var sayılan olarak, sol bölüm kenarı kapsayıcıdır, bu nedenle 00:00 değeri 00:00 - 00:05 aralığına dahil edilir. 1 Passing closed='right' sağda kapatılacak aralığı değiştirir.

In [213]:

```
ts.resample('5min', closed='right').sum()
```

Out[213]:

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: int64

Ortaya çıkan zaman serisi, her bölmenin sol tarafındaki zaman damgalarıyla etiketlenir. passing `label='right'`, bunları sağ bölme kenarıyla etiketleyebilirsiniz.

In [214]:

```
ts.resample('5min', closed='right', label='right').sum()
```

Out[214]:

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: int64

Beş dakikalık frekansa yeniden örneklenen dakika frekans verilerinin bir çizimi için Şekil 11-3'e bakın.

In [215]:

```
Image("img/picture83.png")
```

Out[215]:

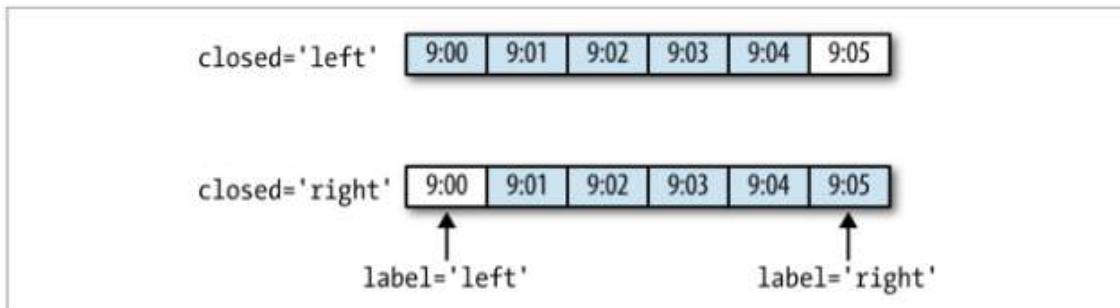


Figure 11-3. Five-minute resampling illustration of closed, label conventions

Son olarak, sonuç indeksini bir miktar kaydırmak isteyebilirsiniz, örneğin, zaman damgasının hangi aralığa atıfta bulunduğu daha net hale getirmek için sağ kenardan bir saniye çıkararak. Bunu yapmak için, `loffset`'e bir dize veya tarih ofseti yazın.

In [216]:

```
ts.resample('5min', closed='right',
            label='right', loffset='-1s').sum()
```

Out[216]:

1999-12-31 23:59:59	0
2000-01-01 00:04:59	15
2000-01-01 00:09:59	40
2000-01-01 00:14:59	11

Freq: 5T, dtype: int64

Ayrıca, `loffset`'in etkisini, `loffset` olmadan sonuç üzerinde kaydırma yöntemi ni çağırarak da elde edebilirsiniz.

Open-High-Low-Close (OHLC) yeniden örnekleme

Finansta, bir zaman serisini toplamanın popüler bir yolu, her bölüm için dört değer hesaplamaktır: first (open), last (close), maximum (high), and minimum (low) değerler. Ohlc toplama işlevini kullanarak, bu dört kümeyi içeren sütlulara sahip bir DataFrame elde edersiniz ve bunlar tek bir veri taramasında verimli bir şekilde hesaplanır.

In [217]:

```
ts.resample('5min').ohlc()
```

Out[217]:

	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

Üst Örnekleme ve Enterpolasyon

Düşük bir frekanstan daha yüksek bir frekansa dönüştürürken, hiçbir toplama gerekmez. Bazı haftalık verileri içeren bir DataFrame'i düşünelim.

In [218]:

```
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 [219]:

```
frame
```

Out[219]:

	Colorado	Texas	New York	Ohio
2000-01-05	0.436794	-1.352664	0.637924	-1.002299
2000-01-12	-0.909720	0.013857	-0.514906	-0.959425

Bu verilerle bir toplama işlevi kullandığınızda, grup başına yalnızca bir değer vardır ve eksik değerler boşluklara neden olur. Herhangi bir toplama olm

aksızın daha yüksek frekansa dönüştürmek için asfreq yöntemini kullanıyoruz.

In [220]:

```
df_daily = frame.resample('D').asfreq()
```

In [221]:

```
df_daily
```

Out[221]:

	Colorado	Texas	New York	Ohio
2000-01-05	0.436794	-1.352664	0.637924	-1.002299
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	-0.909720	0.013857	-0.514906	-0.959425

Çarşamba günleri olmayan her haftalık değeri ileriye doğru doldurmak istediğiniz varsayıyalım. Fillna ve reindex yöntemlerinde bulunan aynı doldurma veya interpolasyon yöntemleri yeniden örneklemeye için kullanılabilir.

In [222]:

```
frame.resample('D').ffill()
```

Out[222]:

	Colorado	Texas	New York	Ohio
2000-01-05	0.436794	-1.352664	0.637924	-1.002299
2000-01-06	0.436794	-1.352664	0.637924	-1.002299
2000-01-07	0.436794	-1.352664	0.637924	-1.002299
2000-01-08	0.436794	-1.352664	0.637924	-1.002299
2000-01-09	0.436794	-1.352664	0.637924	-1.002299
2000-01-10	0.436794	-1.352664	0.637924	-1.002299
2000-01-11	0.436794	-1.352664	0.637924	-1.002299
2000-01-12	-0.909720	0.013857	-0.514906	-0.959425

In [223]:

```
frame.resample('D').ffill(limit=2)
```

Out[223]:

	Colorado	Texas	New York	Ohio
2000-01-05	0.436794	-1.352664	0.637924	-1.002299
2000-01-06	0.436794	-1.352664	0.637924	-1.002299
2000-01-07	0.436794	-1.352664	0.637924	-1.002299
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	-0.909720	0.013857	-0.514906	-0.959425

Özellikle, yeni tarih endeksinin eskisiyle hiç çakışmaz.

In [224]:

```
frame.resample('W-THU').ffill()
```

Out[224]:

	Colorado	Texas	New York	Ohio
2000-01-06	0.436794	-1.352664	0.637924	-1.002299
2000-01-13	-0.909720	0.013857	-0.514906	-0.959425

Periyotlar ile Yeniden Örnekleme

Periyotlar ile dizine alınan verilerin yeniden örneklenmesi, zaman damgaları na benzer.

In [225]:

```
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 [226]:

frame[:5]

Out[226]:

	Colorado	Texas	New York	Ohio
2000-01	0.057913	-0.613083	0.753839	0.171809
2000-02	-0.464528	0.678536	1.872177	-0.783824
2000-03	-0.004426	-2.188372	-0.196629	-1.452787
2000-04	0.825686	-0.558141	0.257589	0.550500
2000-05	0.888063	0.795598	0.846690	0.855604

In [227]:

annual_frame = frame.resample('A-DEC').mean()

In [228]:

annual_frame

Out[228]:

	Colorado	Texas	New York	Ohio
2000	0.016765	-0.282660	0.029910	-0.241916
2001	0.017577	-0.014823	-0.277663	0.034438

Yeni frekansta zaman aralığının hangi ucunun değerleri yeniden örneklemeden önce yerlestireceğine karar vermeniz gerekiğinden, asfreq yönteminde olduğ gibi üst örnekleme daha ayrıntılıdır. Kural bağımsız değişkeni varsayılan olarak "start" dır, ancak "end" de olabilir.

In [229]:

```
# Q-DEC: Üç ayda bir, Aralık'ta biten yıl
annual_frame.resample('Q-DEC').ffill()
```

Out[229]:

	Colorado	Texas	New York	Ohio
2000Q1	0.016765	-0.282660	0.029910	-0.241916
2000Q2	0.016765	-0.282660	0.029910	-0.241916
2000Q3	0.016765	-0.282660	0.029910	-0.241916
2000Q4	0.016765	-0.282660	0.029910	-0.241916
2001Q1	0.017577	-0.014823	-0.277663	0.034438
2001Q2	0.017577	-0.014823	-0.277663	0.034438
2001Q3	0.017577	-0.014823	-0.277663	0.034438
2001Q4	0.017577	-0.014823	-0.277663	0.034438

In [230]:

```
annual_frame.resample('Q-DEC', convention='end').ffill()
```

Out[230]:

	Colorado	Texas	New York	Ohio
2000Q4	0.016765	-0.282660	0.029910	-0.241916
2001Q1	0.016765	-0.282660	0.029910	-0.241916
2001Q2	0.016765	-0.282660	0.029910	-0.241916
2001Q3	0.016765	-0.282660	0.029910	-0.241916
2001Q4	0.017577	-0.014823	-0.277663	0.034438

Dönemler zaman aralıklarına atıfta bulunduğuundan, yukarı örnekleme ve alt örnekleme ile ilgili kurallar daha katıdır.

- Alt örneklemede, hedef frekans, kaynak frekansının bir alt periyodu olmalıdır.
- Yukarı örneklemede, hedef frekans, kaynak frekansının bir süper periyodu olmalıdır.

Bu kurallar yerine getirilmezse, bir istisna gündeme getirilecektir. Bu, esas olarak üç aylık, yıllık ve haftalık sıkılıkları etkiler; örneğin, Q-MAR tarafından tanımlanan zaman aralıkları yalnızca A-MAR, A-JUN, A-SEP ve A-DEC ile uyumludur.

In [231]:

```
annual_frame.resample('Q-MAR').ffill()
```

Out[231]:

	Colorado	Texas	New York	Ohio
2000Q4	0.016765	-0.282660	0.029910	-0.241916
2001Q1	0.016765	-0.282660	0.029910	-0.241916
2001Q2	0.016765	-0.282660	0.029910	-0.241916
2001Q3	0.016765	-0.282660	0.029910	-0.241916
2001Q4	0.017577	-0.014823	-0.277663	0.034438
2002Q1	0.017577	-0.014823	-0.277663	0.034438
2002Q2	0.017577	-0.014823	-0.277663	0.034438
2002Q3	0.017577	-0.014823	-0.277663	0.034438

11.7 Pencere Fonksiyonlarını Taşıma

Zaman serisi işlemleri için kullanılan önemli bir dizi dönüşüm sınıfı, kayan bir pencere üzerinden veya üssel olarak azalan ağırlıklarla değerlendirilen istatistiksel ve diğer işlevlerdir. Bu, gürültülü veya boşluklu verileri düzeltmek için yararlı olabilir. Bu hareketli pencere fonksiyonlarını, üstel ağırlıklı hareketli ortalama gibi sabit uzunlukta bir pencere olmayan fonksiyonlar içermesine rağmen diyorum. Diğer istatistiksel işlevler gibi, bunlar da eksik verileri otomatik olarak hariç tutar.

Derinlemesine incelemeden önce, bazı zaman serisi verilerini yükleyebilir ve iş günü sıklığına göre yeniden örnekleyebiliriz.

In [232]:

```
close_px_all = pd.read_csv('examples/stock_px_2.csv',
                           parse_dates=True, index_col=0)
```

In [233]:

```
close_px = close_px_all[['AAPL', 'MSFT', 'XOM']]
```

In [234]:

```
close_px = close_px.resample('B').ffill()
```

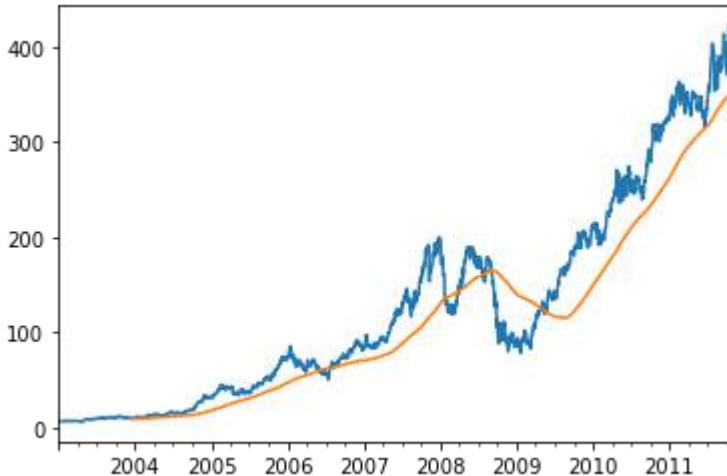
Bir Seri veya DataFrame üzerinde bir pencere ile birlikte çağrılabılır (bir dizi periyot olarak ifade edilir; oluşturulan grafik aşağıdadır).

In [235]:

```
close_px.AAPL.plot()
close_px.AAPL.rolling(250).mean().plot()
```

Out[235]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffe2dcf1880>
```



Dönen ifade (250) davranış olarak groupby'ye benzer, ancak grüplamak yerine 250 günlük bir kayan pencerede grüplamayı mümkün kılan bir nesne yaratır. İşte burada, Apple'ın hisse senedi fiyatının 250 günlük hareketli pencere ortalamasına sahibiz. Varsayılan olarak, döndürme işlevleri, penceredeki tüm değerlerin NA dışı olmasını gerektirir. Bu davranış, eksik verileri ve özellikle zaman serisinin başlangıcında pencere döneminden daha az veriye sahip olağınız gerektiğini hesaba katacak şekilde değiştirilebilir.

In [236]:

```
appl_std250 = close_px.AAPL.rolling(250, min_periods=10).std()
```

In [237]:

```
appl_std250[5:12]
```

Out[237]:

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

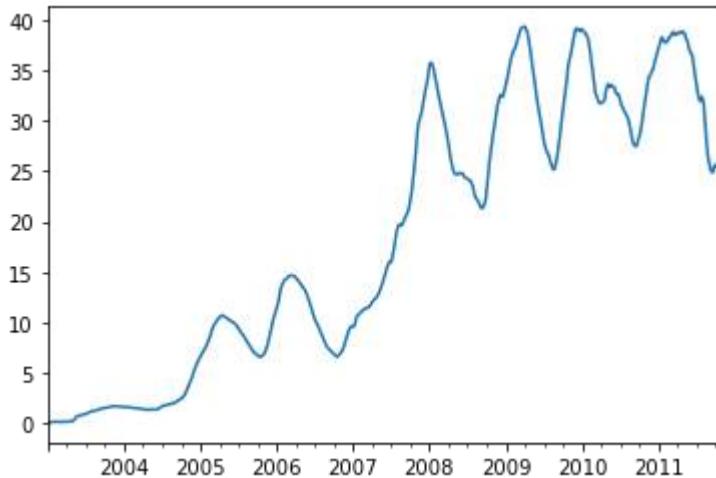
Freq: B, Name: AAPL, dtype: float64

In [238]:

```
appl_std250.plot()
```

Out[238]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffe2de466a0>
```



Genişleyen pencere ortalamasını hesaplamak için, döndürme yerine genişletme operatörünü kullanın. Genişleyen ortalama, zaman penceresini zaman serisini başlangıcından başlatır ve tüm seriyi kapsayana kadar pencerenin boyutunu artırır. `Apple_std250` zaman serisindeki genişleyen pencere anlamı şuna benzer.

In [239]:

```
expanding_mean = appl_std250.expanding().mean()
```

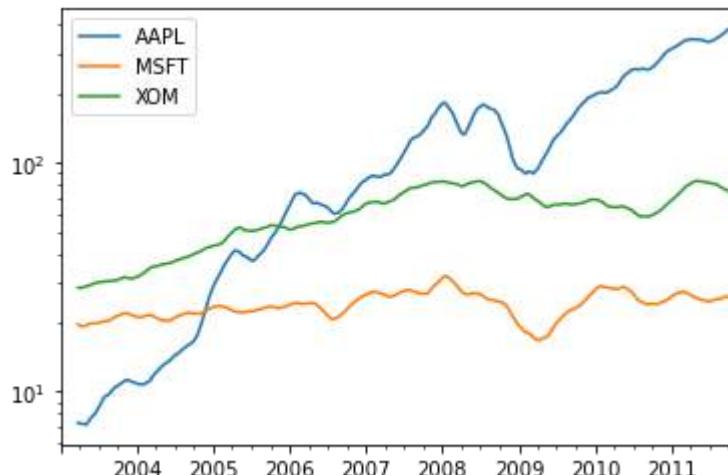
Bir DataFrame'de hareketli bir pencere fonksiyonunun çağrılması, dönüşümü her bir sütuna uygular.

In [240]:

```
close_px.rolling(60).mean().plot(logy=True)
```

Out[240]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffe2e05d490>
```



Döndürme işlevi ayrıca belirli bir dönem sayısı yerine sabit boyutlu bir zaman farkını gösteren bir dizeyi de kabul eder. Bu gösterimi kullanmak, düzeniz zaman serileri için faydalı olabilir. Bunlar, yeniden örneklemek için geçebileceğiniz aynı dizelerdir. Örneğin, 20 günlük hareketli bir ortalamayı şekilde hesaplayabiliriz:

In [241]:

```
close_px.rolling('20D').mean()
```

Out[241]:

	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 pencere boyutu kullanmanın bir alternatif'i, daha yeni gözlemlere daha fazla ağırlık vermek için sabit bir bozulma faktörü belirlemektir. Bozunma faktörünü belirlemenin birkaç yolu vardır. Popüler olanı, sonucu açılığa eşit pencere boyutuna sahip basit hareketli bir pencere işleviyle karşılaştırılabilir kılan bir aralık kullanmaktadır.

Üssel ağırlıklı bir istatistik, daha yeni gözlemlere daha fazla ağırlık verdığından, eşit ağırlıklı versiyona kıyasla değişikliklere daha hızlı "uyum sağlar".

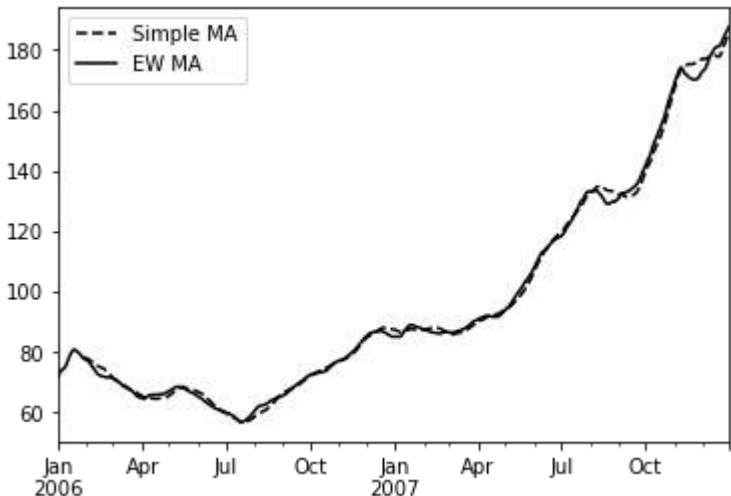
Pandas, yuvarlanma ve genişletme ile birlikte hareket edecek ewm operatörüne sahiptir. Aşağıda, Apple'ın hisse senedi fiyatının 60 günlük hareketli ortalamasını, span = 60 olan EW hareketli ortalama ile karşılaştırın bir örnek yer almaktadır.

In [242]:

```
import matplotlib.pyplot as plt
aapl_px = close_px.AAPL['2006':'2007']
ma60 = aapl_px.rolling(30, min_periods=20).mean()
ewma60 = aapl_px.ewm(span=30).mean()
ma60.plot(style='k--', label='Simple MA')
ewma60.plot(style='k-', label='EW MA')
plt.legend()
```

Out[242]:

<matplotlib.legend.Legend at 0x7ffe2e369eb0>



Binary Hareketli Pencere Fonksiyonları

Korelasyon ve kovaryans gibi bazı istatistiksel operatörlerin iki zaman seri içinde çalışması gereklidir. Örnek olarak, finansal analistler genellikle bir hissenin S&P 500 gibi bir karşılaştırma endeksi ile ilişkisiyle ilgilenirler.

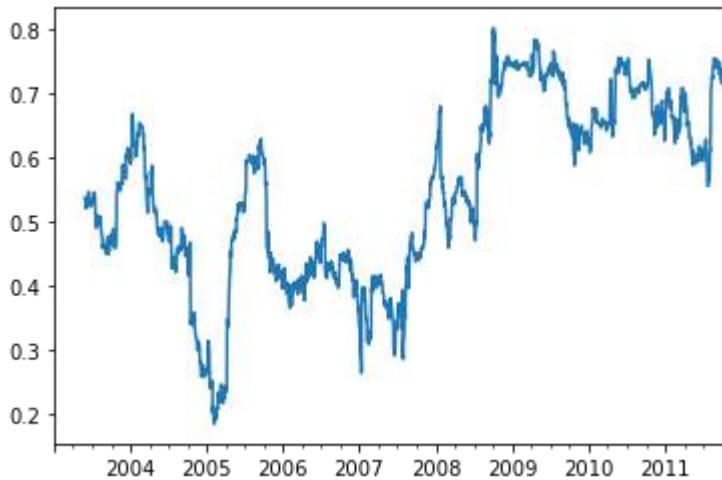
Buna bir göz atmak için, önce tüm ilgi serilerimiz için yüzde değişimini hesaplıyoruz.

In [243]:

```
spx_px = close_px_all['SPX']
spx_rets = spx_px.pct_change()
returns = close_px.pct_change()
corr = returns.AAPL.rolling(125, min_periods=100).corr(spx_rets)
corr.plot()
```

Out[243]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffe2e5baee0>
```



S&P 500 endeksinin birçok hisse senedi ile korelasyonunu aynı anda hesaplama istedığınızı varsayıyalım. Bir döngü yazmak ve yeni bir DataFrame oluşturmak kolay olabilir, ancak tekrarlanabilir, bu nedenle bir Series ve bir DataFrame geçirirseniz, rolling_corr gibi bir işlev DataFrame'deki her sütunla Series'in (bu durumda spx_rets) korelasyonunu hesaplayacaktır.

In [244]:

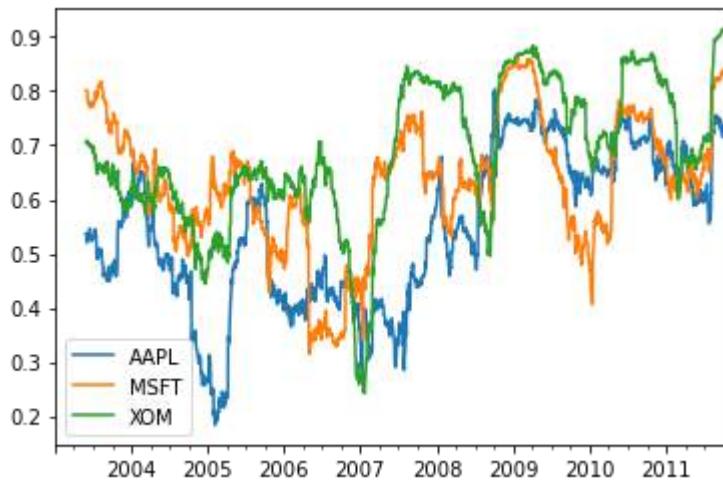
```
corr = returns.rolling(125, min_periods=100).corr(spx_rets)
```

In [245]:

corr.plot()

Out[245]:

<matplotlib.axes._subplots.AxesSubplot at 0x7ffe2e831e50>



Kullanıcı Tanımlı Hareketli Pencere Fonksiyonları

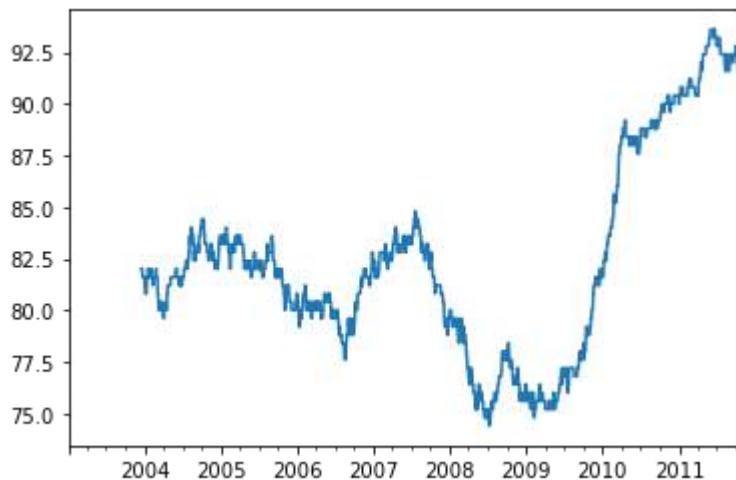
Döndürme ve ilgili yöntemler üzerinde uygulama yöntemi, hareketli bir pencere üzerinde kendi tasarladığınız bir dizi işlevini uygulamak için bir yol sağlar. Tek gereksinim, işlevin dizinin her bir parçasından tek bir değer (bir azalma) üretmesidir. Örneğin, yuvarlanan (...). Quantile (q) kullanarak örnek nicelikleri hesaplayabilirken, belirli bir değerin örneklem üzerindeki yüzdelik sıralamasıyla ilgilenebiliriz. Scipy.stats.percentileofscore işlevi tam da bunu yapar.

In [246]:

```
from scipy.stats import percentileofscore
score_at_2percent = lambda x: percentileofscore(x, 0.02)
result = returns.AAPL.rolling(250).apply(score_at_2percent)
result.plot()
```

Out[246]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffe2f509160>
```



Advanced pandas

12.1 Categorical Data

Background and Motivation

Çoğunlukla, tablodaki bir sütun, daha küçük farklı değerler kümesinin yineleen örneklerini içerebilir. Sırasıyla bir diziden farklı değerleri çıkarmamızı ve frekanslarını hesaplamamızı sağlayan unique ve value_counts gibi fonksiyonları daha önce görmüştük.

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  
dtype: int64
```

Birçok veri sistemi (veri ambarlama, istatistiksel hesaplama veya diğer kullanımlar için), verileri daha verimli depolama ve hesaplama için 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 depolayan boyut 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
```

Örijinal dizi dizisini geri yüklemek için alma 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
```

Tamsayılar olarak bu temsil, kategorik veya sözlük kodlu temsil olarak adlanır. Farklı değerler dizisi, verilerin kategorileri, sözlükleri veya düzeyleri olarak adlandırılabilir. Bu kitapta kategorik ve kategoriler terimlerini kullanacağız. Kategorilere atıfta bulunan tam sayı değerlerine kategori kodları veya basitçe kodlar denir.

Kategorik temsil, analiz yaparken önemli performans iyileştirmeleri sağlayabilir. Kodları değiştirmeden bırakarak kategoriler üzerinde de dönüşümler gerçekleştirebilirsiniz. Nispeten düşük maliyetle yapılabilecek bazı örnek döşümler şunlardır:

- Kategorileri yeniden adlandırmak
- Mevcut kategorilerin sırasını veya konumunu değiştirmeden yeni bir kategori eklemek

Pandas'ta Kategorik Tür

Pandas, tamsayı tabanlı kategorik gösterimi veya kodlamayı kullanan verileri tutmak için özel bir Kategorik tipe sahiptir. Daha önceki örnek Seriyi ele alalım:

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	8	2.587693
1	1	orange	14	2.375683
2	2	apple	14	2.173602
3	3	apple	14	3.069743
4	4	apple	3	0.401675
5	5	orange	3	2.791662
6	6	apple	3	3.935522
7	7	apple	5	2.747347

Burada, df ['meyve'] bir Python dizesi nesneleri dizisidir. Sunları arayarak onu kategoriye dönüştürebiliriz.

In [15]:

fruit_cat = df['fruit'].astype('category')

In [16]:

fruit_cat

Out[16]:

```
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 için değerler bir NumPy dizisi değil, bir pandas örneğidir.

In [17]:

```
c = fruit_cat.values
```

In [18]:

```
type(c)
```

Out[18]:

```
pandas.core.arrays.categorical.Categorical
```

Kategorik nesnenin kategorileri ve kod öznitelikleri vardır.

In [19]:

```
c.categories
```

Out[19]:

```
Index(['apple', 'orange'], dtype='object')
```

In [20]:

```
c.codes
```

Out[20]:

```
array([0, 1, 0, 0, 0, 1, 0, 0], dtype=int8)
```

Dönüştürülen sonucu atayarak DataFrame sütununu kategorik hale dönüştürebilirsiniz.

In [21]:

```
df['fruit'] = df['fruit'].astype('category')
```

In [22]:

```
df.fruit
```

Out[22]:

```
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']
```

Ayrıca doğrudan diğer Python dizileri türlerinden de pandas.Categorical ol

üştürabilirsiniz.

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 veriler elde ettiyseniz, `from_codes` yapıcısından alternatifini kullanabilirsiniz.

In [25]:

```
categories = ['foo', 'bar', 'baz']
```

In [26]:

```
codes=[0,1,2,0,0,1]
```

In [28]:

```
my_cats_2 = pd.Categorical.from_codes(codes, categories)
```

In [29]:

```
my_cats_2
```

Out[29]:

```
['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo', 'bar', 'baz']
```

Açıkça belirtilmedikçe, kategorik dönüştürmeler, kategorilerin belirli bir sıralamasının olmadığını varsayar. Dolayısıyla, kategori dizisi, giriş verilerinin sırasına bağlı olarak farklı bir sırada olabilir. `from_codes` veya diğer kuruculardan herhangi birini kullanırken, kategorilerin anlamlı bir sıralama maya sahip olduğunu belirtebilirsiniz.

In [30]:

```
ordered_cat = pd.Categorical.from_codes(codes, categories,
                                         ordered=True)
```

In [31]:

ordered_cat

Out[31]:

```
['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo' < 'bar' < 'baz']
```

[Foo <bar <baz] çıktısı, sıralamada 'foo'nun 'bar'dan önce geldiğini belirtir ve bu böyle devam eder. Sırasız kategorik bir örnek, as_ordered ile sıralanabilir.

In [32]:

my_cats_2.as_ordered()

Out[32]:

```
['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo' < 'bar' < 'baz']
```

Son bir not olarak, yalnızca dizgi örnekleri göstermemə rağmen kategorik verilerin dizge olması gerekməz. Kategorik bir dizi, herhangi bir değişmez deyər türünden oluşabilir.

Kategoriye göre hesaplamalar

Pandas Categorical'i kodlanmış sürümle karşılaştırıldığında (bir dizi dizesi gibi) kullanmak genellikle aynı şekilde davranışır. Pandaların groupby işlevi gibi bazı bölümleri, kategoriklerle çalışırken daha iyi performans gösterir. Sıralı bayrağı kullanabilen bazı işlevler de vardır. Bazı rastgele sayısal verileri ele alalım ve pandas.qcut binning fonksiyonunu kullanalım. Bu dönüs pandas'ı kategorik.

In [33]:

np.random.seed(12345)

In [35]:

draws = np.random.randn(1000)

In [36]:

draws[:5]

Out[36]:

```
array([-0.98350472,  0.93094376, -0.81167564, -1.83015626, -0.1387301
5])
```

Bu verilerin dörtte bir grüplamasını hesaplayalım ve bazı istatistikleri çıkaralım.

In [37]:

```
bins = pd.qcut(draws, 4)
```

In [38]:

```
bins
```

Out[38]:

```
[(-3.746, -0.623], (0.677, 3.26], (-3.746, -0.623], (-3.746, -0.623],
(-0.623, -0.0206], ..., (-3.746, -0.623], (-0.623, -0.0206], (-0.623,
-0.0206], (-3.746, -0.623], (0.677, 3.26]]
Length: 1000
Categories (4, interval[float64]): [(-3.746, -0.623] < (-0.623, -0.020
6] < (-0.0206, 0.677] < (0.677, 3.26]]
```

Yararlı olsa da, tam örnek çeyrekleri bir rapor oluşturmak için çeyrek adlarından daha az yararlı olabilir. Bunu qcut'ın etiketler argümanıyla başarabiliz.

In [39]:

```
bins = pd.qcut(draws, 4, labels=['Q1', 'Q2', 'Q3', 'Q4'])
```

In [40]:

```
bins
```

Out[40]:

```
['Q1', 'Q4', 'Q1', 'Q1', 'Q2', ..., 'Q1', 'Q2', 'Q2', 'Q1', 'Q4']
Length: 1000
Categories (4, object): ['Q1' < 'Q2' < 'Q3' < 'Q4']
```

In [41]:

```
bins.codes[:10]
```

Out[41]:

```
array([0, 3, 0, 0, 1, 2, 2, 1, 3, 2], dtype=int8)
```

Kategorik etiketli bölmeler, verilerdeki bölme kenarları hakkında bilgi içermez, bu nedenle bazı özet istatistikleri çıkarmak için groupby'yi kullanabiliriz.

In [42]:

```
bins = pd.Series(bins, name='quartile')
```

In [43]:

```
results = (pd.Series(draws)
            .groupby(bins)
            .agg(['count', 'min', 'max'])
            .reset_index())
```

In [44]:

```
results
```

Out[44]:

	quartile	count	min	max
0	Q1	250	-3.745356	-0.627426
1	Q2	250	-0.621249	-0.021299
2	Q3	250	-0.019941	0.677246
3	Q4	250	0.677263	3.260383

Sonuçtaki 'çeyrek' sütunu, kutulardaki sıralama dahil olmak üzere orijinal kategorik bilgileri saklar.

In [45]:

```
results['quartile']
```

Out[45]:

```
0    Q1
1    Q2
2    Q3
3    Q4
Name: quartile, dtype: category
Categories (4, object): ['Q1' < 'Q2' < 'Q3' < 'Q4']
```

Kategoriklerle daha iyi performans

Belirli bir veri kümesi üzerinde çok fazla analiz yaparsanız, kategoriye dönüştürmek önemli genel performans kazanımları sağlayabilir. Bir DataFrame sütununun kategorik bir sürümü de genellikle ölçüde daha az bellek kullanır. 10 milyon unsur ve az sayıda farklı kategori içeren bazı Serileri ele alalım:

In [46]:

```
N = 10000000
```

In [47]:

```
draws = pd.Series(np.random.randn(N))
```

In [48]:

```
labels = pd.Series(['foo', 'bar', 'baz', 'qux'] * (N // 4))
```

Şimdi etiketleri kategorik hale getiriyoruz:

In [49]:

```
categories = labels.astype('category')
```

Şimdi etiketlerin kategorilerden önemli ölçüde daha fazla bellek kullandığını `not` ediyoruz:

In [50]:

```
labels.memory_usage()
```

Out[50]:

80000128

In [51]:

```
categories.memory_usage()
```

Out[51]:

10000320

Kategoriye dönüştürme elbette ücretsiz değildir, ancak tek seferlik bir maliyettir.

In [52]:

```
%time _ = labels.astype('category')
```

```
CPU times: user 381 ms, sys: 25.1 ms, total: 406 ms
Wall time: 407 ms
```

GroupBy işlemleri kategoriklerle önemli ölçüde daha hızlı olabilir, çünkü temel algoritmalar bir dizi dizisi yerine tamsayı tabanlı kod dizisini kullanır.

Kategorik Yöntemler

Kategorik veriler içeren diziler, `Series.str` özel dizge yöntemlerine benzer birkaç özel yönteme sahiptir. Bu aynı zamanda kategori ve kodlara kolay erişim sağlar. Seriyi düşünün:

In [53]:

```
s = pd.Series(['a', 'b', 'c', 'd'] * 2)
```

In [54]:

```
cat_s = s.astype('category')
```

In [55]:

```
cat_s
```

Out[55]:

```
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 [56]:

```
cat_s.cat.codes
```

Out[56]:

```
0    0
1    1
2    2
3    3
4    0
5    1
6    2
7    3
dtype: int8
```

In [57]:

```
cat_s.cat.categories
```

Out[57]:

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

Bu veriler için gerçek kategori kümesinin verilerde gözlemlenen dört değerin ötesine geçtiğini bildiğimizi varsayıyalım. Bunları değiştirmek için set_categories yöntemini kullanabiliriz.

In [58]:

```
actual_categories = ['a', 'b', 'c', 'd', 'e']
```

In [59]:

```
cat_s2 = cat_s.cat.set_categories(actual_categories)
```

In [60]:

```
cat_s2
```

Out[60]:

```
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 onları kullanan işlemle re yansıtılacaktır. Örneğin, değer_sayıısı, varsa kategorilere saygı duyar.

In [61]:

```
cat_s.value_counts()
```

Out[61]:

```
d    2
c    2
b    2
a    2
dtype: int64
```

In [62]:

```
cat_s2.value_counts()
```

Out[62]:

```
d    2
c    2
b    2
a    2
e    0
dtype: int64
```

Büyük veri kümelerinde, kategorik bilgiler genellikle bellek tasarrufu ve da ha iyi performans için uygun bir araç olarak kullanılır. Büyük bir DataFrame veya Seriyi filtreledikten sonra, kategorilerin çoğu verilerde görünmeyebilir.

r. Buna yardımcı olmak için, gözlenmeyen kategorileri kırpmak için `remove_unused_categories` yöntemini kullanabiliriz.

In [63]:

```
cat_s3 = cat_s[cat_s.isin(['a', 'b'])]
```

In [64]:

```
cat_s3
```

Out[64]:

```
0    a
1    b
4    a
5    b
dtype: category
Categories (4, object): ['a', 'b', 'c', 'd']
```

In [65]:

```
cat_s3.cat.remove_unused_categories()
```

Out[65]:

```
0    a
1    b
4    a
5    b
dtype: category
Categories (2, object): ['a', 'b']
```

In [66]:

```
from IPython.display import Image
Image("img/picture84.png")
```

Out[66]:

Table 12-1. Categorical methods for Series in pandas

Method	Description
<code>add_categories</code>	Append new (unused) categories at end of existing categories
<code>as_ordered</code>	Make categories ordered
<code>as_unordered</code>	Make categories unordered
<code>remove_categories</code>	Remove categories, setting any removed values to null
<code>remove_unused_categories</code>	Remove any category values which do not appear in the data
<code>rename_categories</code>	Replace categories with indicated set of new category names; cannot change the number of categories
<code>reorder_categories</code>	Behaves like <code>rename_categories</code> , but can also change the result to have ordered categories
<code>set_categories</code>	Replace the categories with the indicated set of new categories; can add or remove categories

Modelleme için 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 sütun içeren bir DataFrame oluşturmayı içerir; bu sütunlar, belirli bir kategorinin oluşumları için 1'leri ve aksi takdirde 0'ı içerir.

Önceki örneği ele alalım:

In [67]:

```
cat_s = pd.Series(['a', 'b', 'c', 'd'] * 2, dtype='category')
```

Bölüm 7'de daha önce bahsedildiği gibi, pandas.get_dummies işlevi bu tek boyutlu kategorik verileri kukla değişkeni içeren bir DataFrame'e dönüştürür.

In [68]:

```
pd.get_dummies(cat_s)
```

Out[68]:

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

12.2 Gelişmiş GrupBy Kullanımı

Bölüm 10'da Series ve DataFrame için groupby yöntemini derinlemesine ele almış olsak da, kullanabileceğiniz bazı ek teknikler vardır.

Grup Dönüşümleri ve "Sarılmamış" GroupBys

Bölüm 10'da, dönüşümleri gerçekleştirmek için gruplanmış işlemlerde uygulama yöntemine baktık. Dönüşüm adında, uygulamaya benzer ancak kullanabileceğiniz işlev türüne daha fazla kısıtlama getiren başka bir yerleşik yöntem vardır:

- Grubun şekline yayınlanacak skaler bir değer üretebilir
- Giriş grubu ile aynı şeke sahip bir nesne üretebilir
- Girdisini değiştirmemelidir

Örnek olarak basit bir örnek ele alalım:

In [69]:

```
df = pd.DataFrame({'key': ['a', 'b', 'c'] * 4,  
                   'value': np.arange(12.)})
```

In [70]:

```
df
```

Out[70]:

	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 [72]:

```
g = df.groupby('key').value
```

In [73]:

```
g.mean()
```

Out[73]:

```
key
a    4.5
b    5.5
c    6.5
Name: value, dtype: float64
```

Bunun yerine, df ['value'] ile aynı şekle sahip, ancak 'anahtar' ile gruplanmış ortalama değerlerle değiştirilen bir Seri üretmek istediğimizi varsayılmam. Lambda x: x.mean () fonksiyonunu dönüşüm geçirebiliriz.

In [75]:

```
g.transform(lambda x: x.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
```

Yerleşik toplama işlevleri için, GroupBy agg yönteminde olduğu gibi bir dize takma adı iletебiliriz.

In [76]:

```
g.transform('mean')
```

Out[76]:

```
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 Series döndüren işlevlerle çalışır, ancak sonuç gir diyle aynı boyutta olmalıdır. Örneğin, bir lambda işlevi kullanarak her grub u 2 ile çarpabiliriz.

In [77]:

```
g.transform(lambda x: x * 2)
```

Out[77]:

```
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
```

Daha karmaşık bir örnek olarak, sıralamaları her grup için azalan sırada hes aplayabiliriz.

In [79]:

```
g.transform(lambda x: x.rank(ascending=False))
```

Out[79]:

```
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
```

Basit toplamalardan oluşan bir grup dönüştürme işlevi düşünün:

In [80]:

```
def normalize(x):
    return (x - x.mean()) / x.std()
```

Bu durumda, dönüşüm kullanarak veya uygulayarak eşdeğer sonuçlar elde edebiliriz:

In [81]:

```
g.transform(normalize)
```

Out[81]:

```
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
```

In [82]:

```
g.apply(normalize)
```

Out[82]:

```
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
```

"Ortalama" veya "toplam" gibi yerleşik toplama işlevleri genellikle genel uygulama işlevinden çok daha hızlıdır. Dönüşümle birlikte kullanıldığında bunları da "hızlı geçmişi" vardır. Bu, paketlenmemiş grup işlemi yapmamıza izin verir.

In [83]:

```
g.transform('mean')
```

Out[83]:

```
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 [85]:

```
normalized = (df['value'] - g.transform('mean')) / g.transform('std')
```

In [86]:

```
normalized
```

Out[86]:

```
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
```

Sarılmamış bir grup işlemi birden fazla grup toplamasını içerebilirken, vektörleştirilmiş işlemlerin genel faydası genellikle bundan daha ağır basar.

Gruplanmış Zaman Resampling

Zaman serisi verileri için, yeniden örnekleme yöntemi anlamsal olarak bir zaman aralığına dayalı bir grup işlemidir. İşte küçük bir örnek tablo:

In [88]:

```
N=15
```

In [89]:

```
times = pd.date_range('2017-05-20 00:00', freq='1min', periods=N)
```

In [90]:

```
df = pd.DataFrame({'time': times,
                   'value': np.arange(N)})
```

In [91]:

df

Out[91]:

	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

Burada, 'zaman'a göre indeksleyebilir ve sonra yeniden örnekleyebiliriz.

In [92]:

df.set_index('time').resample('5min').count()

Out[92]:

	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 anahtarı sütunu ile işaretlenmiş birden çok zaman serisi içerdigini varsayıyalım:

In [93]:

```
df2 = pd.DataFrame({'time': times.repeat(3),
                    'key': np.tile(['a', 'b', 'c'], N),
                    'value': np.arange(N * 3.)})
```

In [94]:

```
df2[:7]
```

Out[94]:

	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

Her bir "anahtar" değeri için aynı yeniden örneklemeyi yapmak için, pandas tanıtıyoruz. Zaman Gruplayıcı nesnesi:

```
time_key = pd.TimeGrouper('5min')
```

Daha sonra zaman indeksini ayarlayabilir, 'anahtar' ve zaman_anahtarına göre gruplayabilir ve bir araya getirebiliriz.

```
resampled = (df2.set_index('time')
              .groupby(['key', time_key])
              .sum())
resampled
resampled.reset_index()
```

TimeGrouper kullanımıyla ilgili bir kısıtlama, zamanın Serilerin veya Data Frame'in indeksi olması gerekligidir.

12.3 Yöntem Zincirleme Teknikleri

Bir veri kümesine bir dizi dönüşüm uygularken, kendinizi analizinizde asla kullanılmayan çok sayıda geçici değişken yaratırken bulabilirsiniz. Örneğin şu örneği ele alalım:

```
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 kullanmasak da, bu örnek bazı yeni yöntemler i vurgulamaktadır. İlk olarak, DataFrame.assign yöntemi, df [k] = v biçimind eki sütun atamalarına işlevsel bir alternatifdir. Nesneyi yerinde değiştirme k yerine, belirtilen değişiklikleri içeren yeni bir DataFrame döndürür. Yani bu ifadeler eşdeğerdir.

```
# Usual non-functional way
df2 = df.copy()
df2['k'] = v
# Functional assign way
df2 = df.assign(k=v)
```

Yerinde atama, atamayı kullanmaktan daha hızlı yürütülebilir, ancak atama da ha kolay yöntem zincirleme sağlar.

```
result = (df2.assign(col1_demeaned=df2.col1 - df2.col2.mean())
          .groupby('key')
          .col1_demeaned.std())
```

Yöntem zincirleme yaparken akılda tutulması gereken bir şey, geçici nesneler e başvurmanız gerekebileceğidir. Önceki örnekte, df geçici değişkenine atana na kadar load_data sonucuna başvuramayız. Buna yardımcı olmak için, atama ve diğer pek çok pandanın işlevi çağrılabılır olarak da bilinen işlev benzer i argümanları kabul eder.

Callable'ları iş başında göstermek için, önceki örneğin bir parçasını düşünün:

```
df = load_data()
df2 = df[df['col2'] < 0]
```

Bu şu şekilde yeniden yazılabılır:

```
df = (load_data()
      [lambda x: x['col2'] < 0])
```

Burada, yük_verisinin sonucu bir değişkene atanmaz, dolayısıyla [] 'ye i letilen işlev daha sonra yöntem zincirinin o aşamasında nesneye bağlanır.

Ardından devam edebilir ve tüm diziyi tek zincirli bir ifade olarak yaza biliyoruz:

```
result = (load_data()
          [lambda x: x.col2 < 0]
          .assign(col1_demeaned=lambda x: x.col1 - x.col1.mean()) .groupby('key')
          .col1_demeaned.std())
```

Bu tarzda kod yazmayı tercih edip etmemeniz bir zevk meselesidir ve ifade yi birden çok adıma bölmek kodunuzu daha okunaklı hale getirebilir.

Boru Yöntemi

Yerleşik pandas işlevleriyle ve az önce incelediğimiz çağrılabılır yöntemlerle yöntem zincirleme yaklaşımıyla çok şey başarabilirsiniz. Ancak, bazen üçüncü taraf kitaplıklarından kendi işlevlerinizi veya işlevlerinizi kullanmanız gereklidir. Boru yönteminin devreye girdiği yer burasıdır.

Bir dizi işlev çağrısını düşünün.

```
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 boru çağrılarını kullanarak yeniden yazabilirsiniz.

```
result = (df.pipe(f, arg1=v1)
          .pipe(g, v2, arg3=v3)
          .pipe(h, arg4=v4))
```

F (df) ve df.pipe (f) ifadesi eşdeğerdir, ancak boru zincirleme çağrımayı kolaylaştırır.

Boru için potansiyel olarak yararlı bir model, işlem dizilerini yeniden kullanılabilir işlevlere genelleştirmektir. Örnek olarak, bir sütundan grup ortalamalarını çıkarmayı düşünelim:

```
g = df.groupby(['key1', 'key2'])
df['col1'] = df['col1'] - g.transform('mean')
```

Birden fazla sütunu küçümsemek ve grup anahtarlarını kolayca değiştirmek istediğinizi varsayıyalım. Ek olarak, bu dönüşümü bir yöntem zincirinde gerçekleştirmek isteyebilirsiniz. İşte örnek bir uygulama:

```
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

    result = (df[df.col1 < 0]
              .pipe(group_demean, ['key1', 'key2'], ['col1']))
```


Python'da Modelleme Kitaplıklarına Giriş

13.1 Pandas ve Model Kodu Arasında Arayüz

Model geliştirme için ortak bir iş akışı, modelin kendisini oluşturmak için bir modelleme kitaplığına geçmeden önce veri yükleme ve temizleme için pandasın kullanılmasıdır. Model geliştirme sürecinin önemli bir parçası, makine öğreniminde özellik mühendisliği olarak adlandırılır. Bu, bir modelleme bağlamında yararlı olabilecek ham veri kümesinden bilgi çıkarılan herhangi bir veri dönüşümünü veya analitiği açıklayabilir. Bu kitapta araştırdığımız veri toplama ve GroupBy araçları, genellikle bir yapı mühendisliği bağlamında kullanılmaktadır. "iyi" özellik mühendisliğinin detayları bu kitap için kapsam dışı olsa da, pandasla veri manipülasyonu ve modellenmesi arasında mümkün olduğunca zahmetsiz geçiş yapmak için bazı yöntemler göstereceğim. Pandas ve diğer analiz kitaplıkları arasındaki temas noktası genellikle NumPy dizileridir. Bir DataFrame'i NumPy dizisine dönüştürmek için .values özelliğini kullanın.

In [1]:

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

In [2]:

```
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 [3]:

```
data
```

Out[3]:

	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 [4]:

data.columns

Out[4]:

Index(['x0', 'x1', 'y'], dtype='object')

In [5]:

data.values

Out[5]:

```
array([[ 1. ,  0.01, -1.5 ],
       [ 2. , -0.01,  0. ],
       [ 3. ,  0.25,  3.6 ],
       [ 4. , -4.1 ,  1.3 ],
       [ 5. ,  0. , -2. ]])
```

DataFrame'e geri dönüştürmek için, önceki bölümlerden hatırlayabileceğiniz gibi, isteğe bağlı sütun adlarııyla iki boyutlu bir ndarray geçirebilirsiniz.

In [6]:

df2 = pd.DataFrame(data.values, columns=['one', 'two', 'three'])

In [7]:

df2

Out[7]:

	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

.values özniteliğinin, verileriniz homojen olduğunda (örneğin, tüm sayısal türler) kullanılması amaçlanmıştır. Heterojen verileriniz varsa, sonuç Python nesnelerinin bir ndarray'ı olacaktır.

In [8]:

df3 = data.copy()

In [9]:

df3['strings'] = ['a', 'b', 'c', 'd', 'e']

In [10]:

df3

Out[10]:

	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 [11]:

df3.values

Out[11]:

```
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. Değerlerle loc indekslemeyi kullanmanızı tavsiye edilir.

In [12]:

model_cols = ['x0', 'x1']

In [13]:

data.loc[:, model_cols].values

Out[13]:

```
array([[ 1.,  0.01],
       [ 2., -0.01],
       [ 3.,  0.25],
       [ 4., -4.1 ],
       [ 5.,  0. ]])
```

Bazı kitaplıkların pandalar için yerel desteği vardır ve bu işlerin bir kısmını sizin için otomatik olarak yapar: DataFrame'den NumPy'ye dönüştürme ve model parametre adlarını çıktı tablolarının veya Serilerin sütunlarına eklemeye. Diğer durumlarda, bu "meta veri yönetimini" manuel olarak yapmanız gerekecektir. Bölüm 12'de pandas Kategorik türüne ve pandas.get_dummies işlevine bakık. Örnek veri kümemizde sayısal olmayan bir sütunumuz olduğunu varsayıyalıyız:

In [14]:

```
data['category'] = pd.Categorical(['a', 'b', 'a', 'a', 'b'],
                                 categories=['a', 'b'])
```

In [15]:

```
data
```

Out[15]:

	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 oluşturur, 'kategori' sütununu kaldırır ve ardından sonucu birleştiririz.

In [16]:

```
dummies = pd.get_dummies(data.category, prefix='category')
```

In [17]:

```
data_with_dummies = data.drop('category', axis=1).join(dummies)
```

In [18]:

```
data_with_dummies
```

Out[18]:

	x0	x1	y	category_a	category_b
0	1	0.01	-1.5	1	0
1	2	-0.01	0.0	0	1
2	3	0.25	3.6	1	0
3	4	-4.10	1.3	1	0
4	5	0.00	-2.0	0	1

Bazı istatistiksel modelleri kukla değişkenlerle uydurmak için bazı nüanslar vardır. Basit sayısal sütunlardan daha fazlasına sahip olduğunuzda Patsy'yi (sonraki bölümün konusu) kullanmak daha basit ve daha az hata yapmaya meyilli olabilir.

13.2 Patsy ile Model Açıklamaları Oluşturma

Patsy, R ve S istatistiksel programlama dilleri tarafından kullanılan formül sözdiziminden esinlenen (ancak tam olarak aynı olmayan) küçük dizge tabanlı "formül sözdizimi" ile istatistiksel modelleri (özellikle doğrusal modelleme) açıklayan bir Python kitaplığıdır. Patsy, istatistik modellerinde doğrusal modelleri belirtmek için iyi bir şekilde desteklenmektedir, Patsy'nin formülleri şuna benzeyen özel bir dizgi sözdizimidir:

```
y ~ x0 + x1
```

A + b sözdizimi, a'yı b'ye eklemek anlamına gelmez, bunun yerine bunların model için oluşturulan tasarım matrisindeki terimler olduğu anlamına gelir. Patsy.dmatrices işlevi, bir veri kümesiyle birlikte (bir DataFrame veya bir dizidiktesi olabilir) bir formül dizesi alır ve doğrusal bir model için tasarım matrisleri üretir.

In [19]:

```
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 [20]:

```
data
```

Out[20]:

	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 [21]:

```
pip install patsy
```

```
Requirement already satisfied: patsy in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (0.5.1)
Requirement already satisfied: six in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from patsy) (1.15.0)
Requirement already satisfied: numpy>=1.4 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from patsy) (1.19.4)
Note: you may need to restart the kernel to use updated packages.
```

In [22]:

```
import patsy
```

In [23]:

```
y, X = patsy.dmatrices('y ~ x0 + x1', data)
```

In [24]:

```
y
```

Out[24]:

```
DesignMatrix with shape (5, 1)
    y
-1.5
 0.0
 3.6
 1.3
-2.0
Terms:
'y' (column 0)
```

In [25]:

```
X
```

Out[25]:

```
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 ndarraylerdir.

In [26]:

```
np.asarray(y)
```

Out[26]:

```
array([[-1.5],
       [ 0. ],
       [ 3.6],
       [ 1.3],
       [-2. ]])
```

In [27]:

```
np.asarray(X)
```

Out[27]:

```
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. Model $e + 0$ terimini ekleyerek kesmeyi bastırabilirsiniz.

In [28]:

```
patsy.dmatrices('y ~ x0 + x1 + 0', data)[1]
```

Out[28]:

```
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 [29]:

```
coef, resid, _, _ = np.linalg.lstsq(X, y)
```

```
/Users/veyseldogan/opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning: `rcond` parameter will change to the default of machine precision times ``max(M, N)`` where M and N are the input matrix dimensions.
```

```
To use the future default and silence this warning we advise to pass `rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
```

```
"""Entry point for launching an IPython kernel.
```

Model meta verileri design_info özniteliğinde tutulur, böylece bir Seri elde etmek için uydurulmuş katsayılarla model sütun adları, örneğin:

In [30]:

coef

Out[30]:

```
array([[ 0.31290976],
       [-0.07910564],
       [-0.26546384]])
```

In [31]:

```
coef = pd.Series(coef.squeeze(), index=X.design_info.column_names)
```

In [32]:

coef

Out[32]:

```
Intercept      0.312910
x0            -0.079106
x1            -0.265464
dtype: float64
```

Patsy Formüllerinde Veri Dönüşümleri

Python kodunu Patsy formüllerinize karıştırabilirsiniz; Formülü değerlendirdirken kütüphane, kapsama alanında kullandığınız fonksiyonları bulmaya çalışacaktır.

In [33]:

```
y, X = patsy.dmatrices('y ~ x0 + np.log(np.abs(x1) + 1)', data)
```

In [34]:

X

Out[34]:

```
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)
```

Yayın olarak kullanılan bazı değişken dönüşümler arasında standartlaştırma (0 ve varyans 1 anlamına gelir) ve merkezleme (ortalamanın çıkarılması) bulunur. Patsy'nin bu amaç için yerleşik FONKSİYONLARI vardır:

In [35]:

```
y, X = patsy.dmatrices('y ~ standardize(x0) + center(x1)', data)
```

In [36]:

```
X
```

Out[36]:

```
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 siğdırabilir, ardından modeli diğerine göre değerlendirebilirsiniz. Bu, daha sonra gözlemlenecek bir uzatma kısmı veya yeni veriler olabilir. Merkez ve standartlaştırma gibi dönüşümleri uygularken, modeli yeni verilere dayalı tahminle r oluşturmak için kullanırken dikkatli olmalısınız. Yeni bir veri kümesini d önüşürürken orijinal veri kümesinin ortalaması veya standart sapması gibi istatistikleri kullanmanız gerektiğinden bunlara durum bilgisi olan dönüştürmel er denir. Patsy.build_design_matrices işlevi, orijinal örnek içi veri kümesi nden kaydedilen bilgileri kullanarak yeni örnek dışı verilere dönüşümler uygulayabilir.

In [37]:

```
new_data = pd.DataFrame({
    'x0': [6, 7, 8, 9],
    'x1': [3.1, -0.5, 0, 2.3],
    'y': [1, 2, 3, 4]})
```

In [38]:

```
new_X = patsy.build_design_matrices([X.design_info], new_data)
```

In [39]:

new_X

Out[39]:

```
[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 formüllerinin bağlamındaki artı simgesi (+) toplama anlamına gelmediği
inden, bir veri kümelerinden ada göre sütun eklemek istediğinizde, bunları özel
I işlevine sarmalısınız.

In [40]:

y, X = patsy.dmatrices('y ~ I(x0 + x1)', data)

In [41]:

X

Out[41]:

```
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. Bu konunun tam olarak ele alınması bu kitabın kapsamı dış
ındadır ve en iyi istatistik dersi ile birlikte incelenebilir. Bir Patsy for
mülünde sayısal olmayan terimler kullandığınızda, bunlar varsayılan olarak k
ukla değişkenlere dönüştürülür. Bir kesişme varsa, eşdoğrusallığı önlemek iç
in seviyelerden biri dışında bırakılacaktır.

In [42]:

```
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 [43]:

```
y, X = patsy.dmatrices('v2 ~ key1', data)
```

In [44]:

```
X
```

Out[44]:

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 kesmeyi çıkarırsanız, her bir kategori değeri için sütunlar model tasarım matrisine dahil edilecektir.

In [45]:

```
y, X = patsy.dmatrices('v2 ~ key1 + 0', data)
```

In [46]:

X

Out[46]:

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 işlevi ile kategorik olarak yorumlanabilir.

In [47]:

y, X = patsy.dmatrices('v2 ~ C(key2)', data)

In [48]:

X

Out[48]:

DesignMatrix with shape (8, 2)

Intercept C(key2)[T.1]

1	0
1	1
1	0
1	1
1	0
1	1
1	0
1	0

Terms:

'Intercept' (column 0)

'C(key2)' (column 1)

Bir modelde birden fazla kategorik terim kullandığınızda, örneğin varyans analizi (ANOVA) modellerinde kullanılabilen key1: key2 formunun etkileşim terimlerini dahil edebileceğiniz için işler daha karmaşık hale gelebilir.

In [49]:

data['key2'] = data['key2'].map({0: 'zero', 1: 'one'})

In [50]:

data

Out[50]:

	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 [51]:

y, X = patsy.dmatrices('v2 ~ key1 + key2', data)

In [52]:

X

Out[52]:

DesignMatrix with shape (8, 3)

	Intercept	key1[T.b]	key2[T.zero]
0	1	0	1
1	1	0	0
2	1	1	1
3	1	1	0
4	1	0	1
5	1	1	0
6	1	0	1
7	1	1	1

Terms:

- 'Intercept' (column 0)
- 'key1' (column 1)
- 'key2' (column 2)

In [53]:

y, X = patsy.dmatrices('v2 ~ key1 + key2 + key1:key2', data)

In [54]:

X

Out[54]:

```
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)
```

13.3 İstatistik modellerine giriş

statsmodels, birçok türden istatistiksel modeli uydurmak, istatistiksel testleri gerçekleştirmek ve veri keşfi ve görselleştirmek için bir Python kitaplığıdır. Statsmodels daha "klasik" sıkılıkçı istatistiksel yöntemler içerirken, Bayes yöntemleri ve makine öğrenme modelleri diğer kütüphanelerde bulunur.

İstatistik modellerinde bulunan bazı model türleri şunları içerir:

- Doğrusal modeller, genelleştirilmiş doğrusal modeller ve sağlam doğrusal modeller
- Doğrusal karışık efekt modelleri
- Varyans analizi (ANOVA) yöntemleri
- Zaman serisi süreçleri ve durum uzayı modelleri
- Genelleştirilmiş anlar yöntemi

Doğrusal Modellerin Tahmin Edilmesi

İstatistik modellerinde, daha basitten (örneğin, sıradan en küçük kareler) daha karmaşığa (örneğin, yinelemeli olarak yeniden ağırlıklendirilmiş en küçük kareler) çeşitli doğrusal regresyon modeli vardır. İstatistik modellerinde doğrusal modeller iki farklı ana arayüze sahiptir: dizi tabanlı ve formül tabanlı. Bunlara, şu API modülü içe aktarmalarıyla erişilir:

In [55]:

```
pip install statsmodels
```

```
Requirement already satisfied: statsmodels in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (0.12.1)
Requirement already satisfied: scipy>=1.1 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from statsmodels) (1.5.2)
Requirement already satisfied: numpy>=1.15 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from statsmodels) (1.19.4)
Requirement already satisfied: patsy>=0.5 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from statsmodels) (0.5.1)
Requirement already satisfied: pandas>=0.21 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from statsmodels) (1.1.3)
Requirement already satisfied: six in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from patsy>=0.5->statsmodels) (1.15.0)
Requirement already satisfied: pytz>=2017.2 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from pandas>=0.21->statsmodels) (2020.1)
Requirement already satisfied: python-dateutil>=2.7.3 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from pandas>=0.21->statsmodels) (2.8.1)
Note: you may need to restart the kernel to use updated packages.
```

In [57]:

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

Bunların nasıl kullanılacağını göstermek için bazı rastgele verilerden doğrudan bir model oluşturuyoruz.

In [58]:

```
def dnorm(mean, variance, size=1):
    if isinstance(size, int):
        size = size,
    return mean + np.sqrt(variance) * np.random.randn(*size)
```

In [59]:

```
np.random.seed(12345)
```

In [62]:

```
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, beta parametreleri bilinen "gerçek" modeli yazdık. Bu durumda, dnorm, belirli bir ortalama ve varyans ile normal olarak dağıtılmış veriler oluş

turmak için yardımcı bir işlevdir. Şimdi elimizde:

In [63]:

```
X[:5]
```

Out[63]:

```
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 [64]:

```
Y[:5]
```

Out[64]:

```
array([ 0.42786349, -0.67348041, -0.09087764, -0.48949442, -0.1289410  
9])
```

Doğrusal bir model genellikle daha önce Patsy ile gördüğümüz gibi bir kesişme terimi ile donatılmıştır. Sm.add_constant işlevi, mevcut bir matrise bir kesme sütunu ekleyebilir.

In [65]:

```
X_model = sm.add_constant(X)
```

In [66]:

```
X_model[:5]
```

Out[66]:

```
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 regresyona uyabilir.

In [67]:

```
model = sm.OLS(y, X)
```

Modelin uyum yöntemi, tahmini model parametrelerini ve diğer təshisleri içeren bir regresyon sonuçları nesnesi döndürür.

In [68]:

```
results = model.fit()
```

In [69]:

```
results.params
```

Out[69]:

```
array([0.17826108, 0.22303962, 0.50095093])
```

Sonuçların özet yöntemi, sonuçların tanılama çıktısını detaylandıran bir modeli yazdırabilir.

model:

In [70]:

```
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: Sun, 27 Dec 2020 Prob (F-statistic): 7.44e-12
Time: 01:27:52 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
0.975]					
x1	0.1783	0.053	3.364	0.001	0.073
0.283					
x2	0.2230	0.046	4.818	0.000	0.131
0.315					
x3	0.5010	0.080	6.237	0.000	0.342
0.660					

=====

```
Omnibus: 4.662 Durbin-Watson: 2.201
Prob(Omnibus): 0.097 Jarque-Bera (JB): 4.098
Skew: 0.481 Prob(JB): 0.129
Kurtosis: 3.243 Cond. No. 1.74
=====
```

Notes:

[1] R^2 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 adalarına x_1 , x_2 vb. Genel adlar verilmiştir. Bunun yerine, tüm model parametrelerinin bir DataFrame'de olduğunu varsayıyalım.

In [71]:

```
data = pd.DataFrame(X, columns=['col0', 'col1', 'col2'])
```

In [72]:

```
data['y'] = y
```

In [73]:

```
data[:5]
```

Out[73]:

	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

Şimdi istatistik modelleri formül API'sini ve Patsy formül dizelerini kullanabiliriz.

In [74]:

```
results = smf.ols('y ~ col0 + col1 + col2', data=data).fit()
```

In [75]:

```
results.params
```

Out[75]:

```
Intercept      0.033559
col0          0.176149
col1          0.224826
col2          0.514808
dtype: float64
```

In [76]:

```
results.tvalues
```

Out[76]:

```
Intercept      0.952188
col0          3.319754
col1          4.850730
col2          6.303971
dtype: float64
```

olarak nasıl döndürdüğünü gözlemleyin. Ayrıca formülleri ve pandas nesnelerini kullanırken add_constant kullanmamıza gerek yoktur.

Yeni örneklem dışı veriler göz önüne alındığında, tahmini model parametreleri verildiğinde tahmin edilen değerleri hesaplayabilirsiniz.

In [77]:

```
results.predict(data[:5])
```

Out[77]:

```
0    -0.002327
1    -0.141904
2     0.041226
3    -0.323070
4    -0.100535
dtype: float64
```

Doğrusal model sonuçlarının analizi, teşhisisi ve keşfedebileceğiniz istatistik modellerinde görselleştirilmesi için birçok ek araç vardır. Sıradan en küçük karelerin ötesinde başka türde doğrusal modeller de vardır.

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 vardır. Otoregresif bir yapı ve gürültü ile bazı zaman serisi verilerini simüle edelim.

In [78]:

```
init_x = 4
```

In [79]:

```
import random
values = [init_x, init_x]
N=1000
```

In [80]:

```
b0=0.8
b1 = -0.4
noise = rnorm(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 parametreli bir AR (2) yapısına (iki gecikme) sahiptir.

Bir AR modeline uyduğunuzda, eklenecek gecikmeli terimlerin sayısını bilmeyebilirsiniz, bu nedenle modeli daha fazla sayıda gecikmeyle sığdırabilirsınız.

In [81]:

```
MAXLAGS = 5
```

In [82]:

```
model = sm.tsa.AR(values)
```

```
/Users/veyseldogan/opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages/statsmodels/tsa/ar_model.py:791: FutureWarning:  
statsmodels.tsa.AR has been deprecated in favor of statsmodels.tsa.AutoReg and  
statsmodels.tsa.SARIMAX.
```

AutoReg adds the ability to specify exogenous variables, include time trends, and add seasonal dummies. The AutoReg API differs from AR since the model is treated as immutable, and so the entire specification including the lag length must be specified when creating the model. This change is too substantial to incorporate into the existing AR api. The function ar_select_order performs lag length selection for AutoReg models.

AutoReg only estimates parameters using conditional MLE (OLS). Use SARIMAX to estimate ARX and related models using full MLE via the Kalman Filter.

To silence this warning and continue using AR until it is removed, use:

```
import warnings  
warnings.filterwarnings('ignore', 'statsmodels.tsa.ar_model.AR', FutureWarning)  
  
warnings.warn(AR_DEPRECATED_WARN, FutureWarning)
```

In [83]:

```
results = model.fit(MAXLAGS)
```

Sonuçlarda tahmin edilen parametrelerde önce kesişme, ardından ilk iki gecikme için tahminler bulunur.

In [84]:

```
results.params
```

Out[84]:

```
array([-0.00616093,  0.78446347, -0.40847891, -0.01364148,  0.0149687  
2,  
     0.01429462])
```

13.4 Scikit-learn'e giriş

scikit-learn, en yaygın kullanılan ve güvenilir genel amaçlı Python makine öğrenimi araç setlerinden biridir. Model seçimi ve değerlendirmesi, veri dönüştümü, veri yükleme ve model kalıcılığı için araçlar içeren çok çeşitli standart denetimli ve denetimsiz makine öğrenimi yöntemleri içerir. Bu modeller sınıflandırma, kümeleme, tahmin ve diğer ortak görevler için kullanılabilir.

Bu bölümde scikit-learn API stilinin kısa bir çeşidini vereceğiz.

Örnek olarak, 1912'de batan Titanik'teki yolcu hayatı kalma oranları hakkında bir Kaggle yarışmasından klasik bir veri kümesini kullanıyorum. Test ve eğitim veri kümesini pandaları kullanarak yükliyoruz.

In [85]:

```
train = pd.read_csv('datasets/titanic/train.csv')
```

In [86]:

```
test = pd.read_csv('datasets/titanic/test.csv')
```

In [87]:

train[:4]

Out[87]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	I
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2, 3101282	7.9250	I
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C

İstatistik modelleri ve scikit-learn gibi kitaplıklar genellikle eksik veriyle beslenemez, bu nedenle eksik verileri içeren herhangi bir veri olup olmadığını görmek için sütunlara bakarız.

In [88]:

train.isnull().sum()

Out[88]:

PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2
dtype: int64	

In [89]:

```
test.isnull().sum()
```

Out[89]:

PassengerId	0
Pclass	0
Name	0
Sex	0
Age	86
SibSp	0
Parch	0
Ticket	0
Fare	1
Cabin	327
Embarked	0
dtype:	int64

Bunun gibi istatistiklerde ve makine öğrenimi örneklerinde, tipik bir görev, verilerdeki özelliklere dayalı olarak bir yolcunun hayatı kalmayıcağıını tahmin etmektir. Bir model, bir eğitim veri kümesine yerleştirilir ve ardından örneklem dışı bir test veri kümesinde değerlendirilir.

Yaşı bir tahmin aracı olarak kullanmak istiyoruz, ancak eksik verileri var.

Eksik veri atamasını yapmanın birkaç yolu vardır, ancak basit bir tane yapacağımız ve her iki tablodaki boş değerleri doldurmak için eğitim veri kümesini n medyanını kullanacağımız.

In [90]:

```
impute_value = train['Age'].median()
```

In [91]:

```
train['Age'] = train['Age'].fillna(impute_value)
```

In [92]:

```
test['Age'] = test['Age'].fillna(impute_value)
```

Şimdi modellerimizi belirlememiz gerekiyor. 'Sex' sütununun kodlanmış versiyonu olarak bir IsFemale sütunu ekliyoruz.

In [93]:

```
train['IsFemale'] = (train['Sex'] == 'female').astype(int)
```

In [94]:

```
test['IsFemale'] = (test['Sex'] == 'female').astype(int)
```

Sonra bazı model değişkenlerine karar veriyoruz ve NumPy dizileri oluşturuyoruz.

ruz.

In [95]:

```
predictors = ['Pclass', 'IsFemale', 'Age']
```

In [96]:

```
X_train = train[predictors].values
```

In [97]:

```
X_test = test[predictors].values
```

In [99]:

```
y_train = train['Survived'].values
```

In [100]:

```
X_train[:5]
```

Out[100]:

```
array([[ 3.,  0., 22.],
       [ 1.,  1., 38.],
       [ 3.,  1., 26.],
       [ 1.,  1., 35.],
       [ 3.,  0., 35.]])
```

In [101]:

```
y_train[:5]
```

Out[101]:

```
array([0, 1, 1, 1, 0])
```

Scikit-learn'ün LogisticRegression modelini kullanıyoruz ve bir model örneği oluşturuyoruz.

In [102]:

```
from sklearn.linear_model import LogisticRegression
```

In [103]:

```
model = LogisticRegression()
```

İstatistik modellerine benzer şekilde, bu modeli modelin uyum yöntemini kullanarak eğitim verilerine uydurabiliriz.

In [104]:

```
model.fit(X_train, y_train)
```

Out[104]:

```
LogisticRegression()
```

Şimdi, `model.predict`'i kullanarak test veri kümesi için tahminler oluşturabiliriz.

In [106]:

```
y_predict = model.predict(X_test)
```

In [107]:

```
y_predict[:10]
```

Out[107]:

```
array([0, 0, 0, 0, 1, 0, 1, 0, 1, 0])
```

Test veri kümesi için doğru değerlere sahipseniz, bir doğruluk yüzdesi veya başka bir hata ölçüsü hesaplayabilirsiniz.

```
(y_true == y_predict).mean()
```

Uygulamada, model eğitiminde genellikle birçok ek karmaşıklık katmanı vardır. Birçok modelin ayarlanabilen parametreleri vardır ve eğitim verilerine aşırı uymayı önlemek için parametre ayarlamada kullanılabilecek çapraz doğrulama gibi teknikler vardır. Bu, genellikle yeni verilerde daha iyi tahmin performansı veya sağlamlık sağlayabilir. Çapraz doğrulama, örneklem dışı öngörü simüle etmek için eğitim verilerini bölerek çalışır. Ortalama hata karesi gibi bir model doğruluk puanına dayalı olarak, model parametreleri üzerinde bir ızgara araması gerçekleştirilebilir. Lojistik regresyon gibi bazı modeller, yerleşik çapraz doğrulamaya sahip tahminci sınıflarına sahiptir. Örneğin, `LogisticRegressionCV` sınıfı, C model düzenleme parametresinde ne kadar ayrıntılı bir ızgara aramasının yapılacağını gösteren bir parametre ile kullanılabilir.

In [109]:

```
from sklearn.linear_model import LogisticRegressionCV
```

In [111]:

```
model_cv = LogisticRegressionCV(10)
```

In [112]:

```
model_cv.fit(X_train, y_train)
```

Out[112]:

```
LogisticRegressionCV()
```

Elle çapraz doğrulama yapmak için, veri bölme sürecini işleyen cross_val_score yardımcı işlevini kullanabilirsiniz. Örneğin, modelimizi eğitim verilerinin örtüşmeyen dört bölümyle çapraz doğrulamak için şunları yapabiliriz:

In [114]:

```
from sklearn.model_selection import cross_val_score
```

In [115]:

```
model = LogisticRegression(C=10)
```

In [116]:

```
scores = cross_val_score(model, X_train, y_train, cv=4)
```

In [117]:

```
scores
```

Out[117]:

```
array([0.77578475, 0.79820628, 0.77578475, 0.78828829])
```

Varsayılan puanlama ölçüsü modele bağlıdır, ancak açık bir puanlama işlevi sağlamak mümkündür. Çapraz doğrulanmış modellerin eğitilmesi daha uzun sürer, ancak genellikle daha iyi model performansı sağlayabilir.

13.5 Eğitiminize Devam Etme

Python'da veya bir Python kullanıcı arayüzüyle uygulanan çeşitli istatistik ve makine öğrenimi türleri için gittikçe daha fazla çerçeve var. Modelleme ve veri bilimi araçlarına adanmış başka birçok kitap da vardır. Bazı mükemmel olanlar:

- Andreas Mueller ve Sarah Guido (O'Reilly) tarafından Python ile Makine Öğrenime Giriş
- Python Veri Bilimi El Kitabı, Jake VanderPlas (O'Reilly)
- Sıfırdan Veri Bilimi: Python ile İlkeler, Joel Grus (O'Reilly)
- Sebastian Raschka (Packt Publishing) tarafından Python Makine Öğrenimi
- Aurélien Géron (O'Reilly) tarafından Scikit-Learn ve TensorFlow ile Uygulamalı Makine Öğrenimi

Veri Analizi Örnekleri

14.1 1.USA.gov Verileri Bitly'den

2011'de, URL kısaltma hizmeti Bitly, .gov veya .mil ile biten bağlantıları kısaltan kullanıcılarından toplanan anonim verilerin beslemesini sağlamak için ABD hükümeti web sitesi USA.gov ile ortaklık kurdu. 2011 yılında, canlı bir beslemenin yanı sıra saatlik anlık görüntüler indirilebilir metin dosyaları olarak mevcuttu. Bu hizmet, bu yazının yazıldığı sırada (2017) kapatılmıştı. Saatlik anlık görüntülerde, her dosyadaki her satır, JavaScript Nesne Gösterimi anlamına gelen JSON olarak bilinen ortak bir web verileri biçimini içerir. Örneğin, bir dosyanın sadece ilk satırını okursak şuna benzer bir şey görebiliriz:

In [8]:

```
path = 'datasets/bitly_usagov/example.txt'
```

In [9]:

```
open(path).readline()
```

Out[9]:

```
{ "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", "n": "k": 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\\1\\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'
```

In [10]:

```
import json
path = 'datasets/bitly_usagov/example.txt'
records = [json.loads(line) for line in open(path)]
```

In [11]:

records[0]

Out[11]:

```
{'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]}
```

Python'da Saat Dilimlerini Sayma

Veri kümesinde (tz alanı) en sık meydana gelen saat dilimlerini bulmakla ilgiliendiğimizi varsayıyalım. Bunu yapmanın birçok yolu var. Öncelikle, bir liste anlayışı kullanarak tekrar saat dilimlerinin bir listesini çıkaralım:

In [12]:

time_zones = [rec['tz'] for rec in records]

```
-----
KeyError                               Traceback (most recent call
last)
<ipython-input-12-f3fbcc37f129> in <module>
----> 1 time_zones = [rec['tz'] for rec in records]

<ipython-input-12-f3fbcc37f129> in <listcomp>(.0)
----> 1 time_zones = [rec['tz'] for rec in records]

KeyError: 'tz'
```

Tüm kayıtların bir saat dilimi alanına sahip olmadığı ortaya çıktı. Listenin anlaşılmasıının sonuna "tz" kaydı ekleyebileceğimiz için, bunu idare etmek kolaydır.

In [13]:

time_zones = [rec['tz'] for rec in records if 'tz' in rec]

In [14]:

```
time_zones[:10]
```

Out[14]:

```
['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, bazlarının bilinmediğini görüyoruz (boş dize). Şimdi, saat dilimine göre sayımlar üretmek için iki yaklaşım varıdır: daha zor yol (sadece Python standart kitaplığını kullanarak) ve daha kolay yol (pandas kullanarak).

In [15]:

```
def get_counts(sequence):
    counts = {}
    for x in sequence:
        if x in counts:
            counts[x] += 1
        else:
            counts[x] = 1
    return counts
```

Python standart kitaplığındaki daha gelişmiş araçları kullanarak aynı şeyi daha kısaca yazabilirsiniz.

In [16]:

```
from collections import defaultdict
```

In [17]:

```
def get_counts2(sequence):
    counts = defaultdict(int) # değerler 0'a başlayacak
    for x in sequence:
        counts[x] += 1
    return counts
```

Bu mantığı, daha yeniden kullanılabilir hale getirmek için bir işlev'e koyduk. Bunu saat dilimlerinde kullanmak için zaman dilimleri listesini geçirmeniz yeterlidir.

In [18]:

```
counts = get_counts(time_zones)
```

In [19]:

```
counts['America/New_York']
```

Out[19]:

```
1
```

In [20]:

```
len(time_zones)
```

Out[20]:

```
3440
```

En iyi 10 saat dilimini ve sayılarını istersek, biraz sözlük akrobasi yapabiliyoruz.

In [21]:

```
def top_counts(count_dict, n=10):
    value_key_pairs = [(count, tz) for tz, count in count_dict.items()]
    value_key_pairs.sort()
    return value_key_pairs[-n:]
```

In [22]:

```
top_counts(counts)
```

Out[22]:

```
[(1, 'America/New_York')]
```

Python standart kitaplığında arama yaparsanız, `collections.Counter` sınıfını bulabilirsiniz, bu da bu görevi çok daha kolay hale getirir.

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)]
```

Pandas Saat Dilimlerini Sayma

Orjinal kayıt kümesinden bir DataFrame oluşturmak, kayıtların listesini pandas'a geçirmek 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
```

Çerçeve için gösterilen çıktı, büyük DataFrame nesneleri için gösterilen özet görünümüdür. Ardından Series için value_counts yöntemini kullanabiliriz.

In [30]:

```
tz_counts = frame['tz'].value_counts()
```

In [31]:

tz_counts[:10]

Out[31]:

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: tz, dtype: int64

Bu verileri matplotlib kullanarak görselleştirebiliriz. Kayıtlardaki bilinmemiş ve eksik saat dilimi verileri için bir yedek değer doldurmak için biraz parçalama yapabilirsiniz. Eksik değerleri fillna yöntemiyle değiştiriyoruz ve boş dizeler için boolean dizi indekslemesini 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 [35]:

tz_counts[:10]

Out[35]:

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: tz, dtype: int64

Bu noktada, seaborn paketini yatay bir çubuk grafiği yapmak için kullanabiliyoruz (sonuç görselleştirme için Şekil 14-1).

In [36]:

```
pip install seaborn
```

```
Requirement already satisfied: seaborn in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (0.11.1)
Requirement already satisfied: scipy>=1.0 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from seaborn) (1.5.2)
Requirement already satisfied: pandas>=0.23 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from seaborn) (1.1.3)
Requirement already satisfied: matplotlib>=2.2 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from seaborn) (3.3.1)
Requirement already satisfied: numpy>=1.15 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from seaborn) (1.19.4)
Requirement already satisfied: pytz>=2017.2 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from pandas>=0.23->seaborn) (2020.1)
Requirement already satisfied: python-dateutil>=2.7.3 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from pandas>=0.23->seaborn) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!>2.1.2,!>2.1.6,>=2.0.3 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from matplotlib>=2.2->seaborn) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from matplotlib>=2.2->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied: certifi>=2020.06.20 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from matplotlib>=2.2->seaborn) (2020.12.5)
Requirement already satisfied: pillow>=6.2.0 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from matplotlib>=2.2->seaborn) (8.0.0)
Requirement already satisfied: six>=1.5 in ./opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from python-dateutil>=2.7.3->pandas>=0.23->seaborn) (1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

In [37]:

```
import seaborn as sns
```

In [38]:

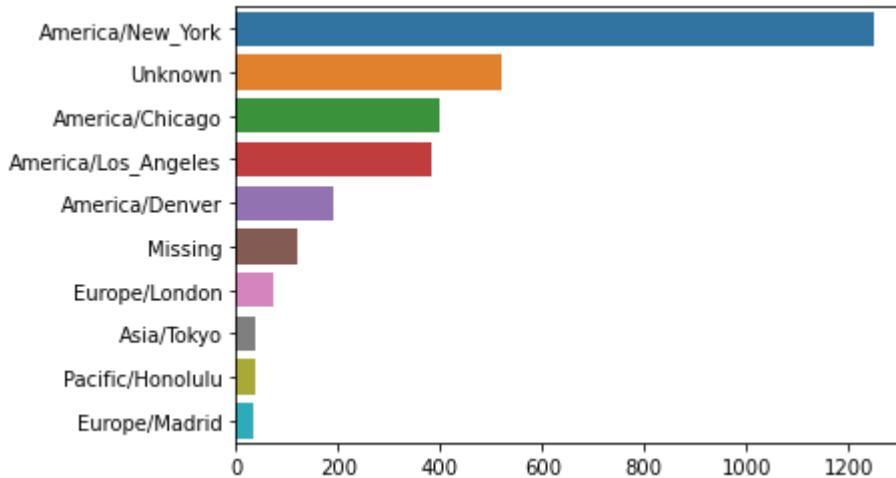
```
subset = tz_counts[:10]
```

In [39]:

```
sns.barplot(y=subset.index, x=subset.values)
```

Out[39]:

<AxesSubplot:>



a, alan, URL kısaltmasını 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'
```

Bu "aracı" dizelerindeki tüm ilginç bilgileri ayırtmak göz korkutucu bir görev gibi görünebilir. Olası bir strateji, dizedeki ilk belirteci ayırmak (kabaca tarayıcı kapasitesine karşılık gelir) ve kullanıcı davranışının baş ka bir özetini yapmaktadır.

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

dtype: int64

Şimdi, en iyi saat dilimlerini Windows ve Windows olmayan kullanıcılara ayırmak istedığınızı varsayıyalım. Basitleştirme olarak, ajan dizesinde 'Windows' dizesi varsa kullanıcının Windows'ta olduğunu varsayıyalım. Bazı araçlar eksik olduğundan, bunları verilerden çıkaracağız.

In [46]:

```
cframe = frame[frame.a.notnull()]
```

Daha sonra her satırın Windows olup olmadığı için bir değer hesaplamak istiyorum.

In [47]:

```
import numpy as np
cframe['os'] = np.where(cframe['a'].str.contains('Windows'),
                        'Windows', 'Not Windows')
```

```
/Users/veyseldogan/opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

This is separate from the ipykernel package so we can avoid doing imports until

In [48]:

```
cframe['os'][:5]
```

Out[48]:

```
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 gruplayabilirsiniz.

In [49]:

```
by_tz_os = cframe.groupby(['tz', 'os'])
```

Değer_sayıısı işlevine benzer şekilde grup sayıları boyut ile hesaplanabilir. Bu sonuç daha sonra unstack ile bir tabloya dönüştürülür.

In [50]:

```
agg_counts = by_tz_os.size().unstack().fillna(0)
```

In [51]:

agg_counts[:10]

Out[51]:

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, genel olarak en iyi saat dilimlerini seçelim. Bunu yapmak için, agg_counts içindeki satır sayımlarından dolaylı bir dizin dizisi oluşturuyoruz.

In [52]:

indexer = agg_counts.sum(1).argsort()

In [53]:

indexer[:10]

Out[53]:

tz	
Africa/Cairo	24
Africa/Casablanca	20
Africa/Ceuta	21
Africa/Johannesburg	92
Africa/Lusaka	87
America/Anchorage	53
America/Argentina/Buenos_Aires	53
America/Argentina/Cordoba	54
America/Argentina/Mendoza	26
	55
	dtype: int64

Sıraları bu sırayla seçmek için `take` komutunu kullanıyorum, ardından son 10 satırı dilimlere ayırıyorum (en büyük değerler).

In [54]:

```
count_subset = agg_counts.take(indexer[-10:])
```

In [55]:

```
count_subset
```

Out[55]:

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

Pandas, aynı şeyi yapan `nlargest` adlı bir kolaylık yöntemine sahiptir.

In [56]:

```
agg_counts.sum(1).nlargest(10)
```

Out[56]:

```
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 grafikte çizilebilir; Seaborn'un `barplot` işlevine ek bir argüman ileterek bunu yığılmış

bir çubuk grafiği yapacağız (Şekil 14-2).

In [57]:

```
count_subset = count_subset.stack()
```

In [58]:

```
count_subset.name = 'total'
```

In [59]:

```
count_subset = count_subset.reset_index()
```

In [60]:

```
count_subset[:10]
```

Out[60]:

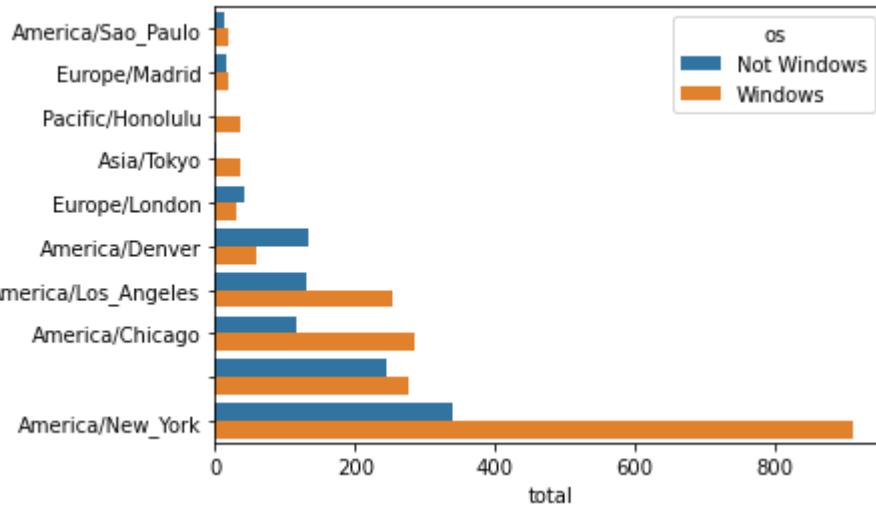
	tz	os	total
0	America/Sao_Paulo	Not Windows	13.0
1	America/Sao_Paulo	Windows	20.0
2	Europe/Madrid	Not Windows	16.0
3	Europe/Madrid	Windows	19.0
4	Pacific/Honolulu	Not Windows	0.0
5	Pacific/Honolulu	Windows	36.0
6	Asia/Tokyo	Not Windows	2.0
7	Asia/Tokyo	Windows	35.0
8	Europe/London	Not Windows	43.0
9	Europe/London	Windows	31.0

In [61]:

```
sns.barplot(x='total', y='tz', hue='os', data=count_subset)
```

Out[61]:

```
<AxesSubplot:xlabel='total', ylabel='tz'>
```



Grafik, daha küçük gruppardaki Windows kullanıcınınin göreli yüzdesini görmeyi kolaylaştırmaz, bu yüzden grup yüzdelelerini 1'e toplamı normalleştirelim.

In [62]:

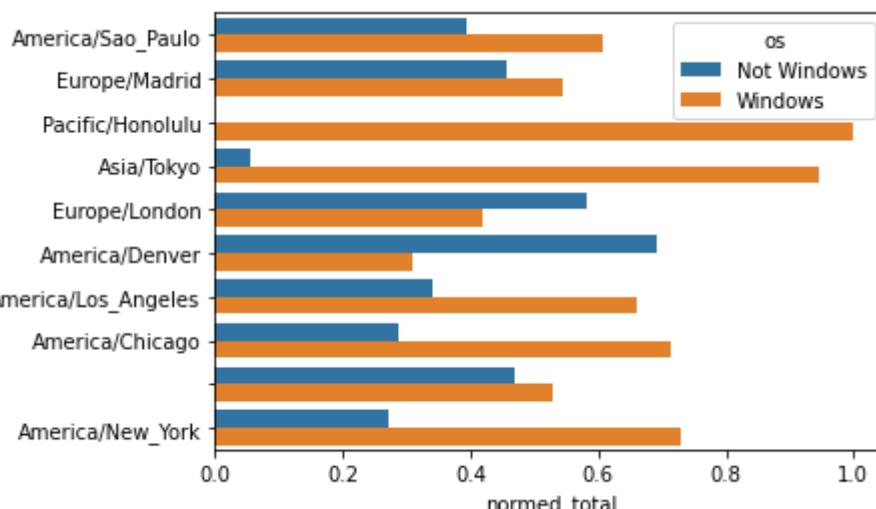
```
def norm_total(group):
    group['normed_total'] = group.total / group.total.sum()
    return group
results = count_subset.groupby('tz').apply(norm_total)
```

In [63]:

```
sns.barplot(x='normed_total', y='tz', hue='os', data=results)
```

Out[63]:

```
<AxesSubplot:xlabel='normed_total', ylabel='tz'>
```



Groupby ile transform yöntemi kullanarak normalleştirilmiş toplamı daha verimli bir şekilde hesaplayabiliyoruz.

In [64]:

```
g = count_subset.groupby('tz')
```

In [65]:

```
results2 = count_subset.total / g.total.transform('sum')
```

14.2 MovieLens 1M Veri Kümesi

GroupLens Research, 1990'ların sonunda ve 2000'lerin başında MovieLens kullanıcılarından toplanan bir dizi film derecelendirme verisi sunar. Veriler, kullanıcılar hakkında film derecelendirmeleri, film meta verileri (türler ve yıl) ve demografik veriler (yaş, posta kodu, cinsiyet kimliği ve meslek) sağlar. Bu tür veriler, genellikle makine öğrenimi algoritmalarına dayalı öneriler sistemlerinin geliştirilmesiyle ilgilidir. Bu kitapta makine öğrenimi tekniklerini ayrıntılı olarak incelemesek de, size bunun gibi veri kümelerini tam olarak ihtiyacınız olan biçimde nasıl böleceğinizi göreceğiz. MovieLens 1M veri kümesi, 4.000 filmdeki 6.000 kullanıcıdan toplanan 1 milyon derecelendirmeye içerir. Üç tabloya yayılmıştır: derecelendirmeler, kullanıcı bilgileri ve film bilgileri. Verileri ZIP dosyasından çıkardıktan sonra, her tabloyu pandas.read_table kullanarak bir pandas DataFrame nesnesine yükleyebiliriz.

In [66]:

```
import pandas as pd
```

In [67]:

```
pd.options.display.max_rows = 10
```

In [68]:

```
unames = ['user_id', 'gender', 'age', 'occupation', 'zip']
users = pd.read_table('datasets/movielens/users.dat', sep='::',
                      header=None, names=unames)
```

```
/Users/veyseldogan/opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages/pandas/io/parsers.py:765: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex separators (separators > 1 char and different from '\s+' are interpreted as regex); you can avoid this warning by specifying engine='python'.
    return read_csv(**locals())
```

In [69]:

```
rnames = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_table('datasets/movielens/ratings.dat', sep='::',
                        header=None, names=rnames)
```

In [70]:

```
mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('datasets/movielens/movies.dat', sep='::',
                      header=None, names=mnames)
```

Python'un dilim sözdizimi ile her bir DataFrame'in ilk birkaç satırına bakarak her şeyin başarılı olduğunu doğrulayabilirsiniz.

In [71]:

```
users[:5]
```

Out[71]:

	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 [72]:

```
ratings[:5]
```

Out[72]:

	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

In [73]:

movies[:5]

Out[73]:

	movie_id	title	genres
0	1	Toy Story (1995)	Animation Children's Comedy
1	2	Jumanji (1995)	Adventure Children's Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama
4	5	Father of the Bride Part II (1995)	Comedy

In [74]:

ratings

Out[74]:

	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

Yaşların ve mesleklerin, veri kümesinin README dosyasında açıklanan grupları gösteren tamsayılar olarak kodlandığını unutmayın. Üç tabloya dağılmış verileri analiz etmek basit bir iş değildir; örneğin, belirli bir film için cinsiyet ve yaşı göre ortalama derecelendirmeleri hesaplamak istediğiniz varsayılim. Göreceğiniz gibi, tek bir tabloda birleştirilen tüm verilerle bunu yapmak çok daha kolaydır. Pandas birleştirme işlevini kullanarak, önce kullanıcı oylarını kullanıcılarla birleştiriyoruz ve ardından bu sonucu film verileriyle birleştiriyoruz. Pandas, örtüsen adlara göre hangi sütunların birleştirme (veya birleştirme) anahtarları olarak kullanılacağını belirler.

In [75]:

```
data = pd.merge(pd.merge(ratings, users), movies)
```

In [76]:

```
data
```

Out[76]:

	user_id	movie_id	rating	timestamp	gender	age	occupation	zip	title
0	1	1193	5	978300760	F	1		10	48067 One Flew Over the Cuckoo's Nest (1975)
1	2	1193	5	978298413	M	56		16	70072 One Flew Over the Cuckoo's Nest (1975)
2	12	1193	4	978220179	M	25		12	32793 One Flew Over the Cuckoo's Nest (1975)
3	15	1193	4	978199279	M	25		7	22903 One Flew Over the Cuckoo's Nest (1975)
4	17	1193	5	978158471	M	50		1	95350 One Flew Over the Cuckoo's Nest (1975)
...
1000204	5949	2198	5	958846401	M	18		17	47901 Modulations (1998)
1000205	5675	2703	3	976029116	M	35		14	30030 Broken Vessels (1998)
1000206	5780	2845	1	958153068	M	18		17	92886 White Boys (1999)
1000207	5851	3607	5	957756608	F	18		20	55410 One Little Indian (1973) Co
1000208	5938	2909	4	957273353	M	25		1	35401 Five Wives, Three Secretaries and Me (1998)

1000209 rows × 10 columns

In [77]:

data.iloc[0]

Out[77]:

```

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 derecelendirmeleri almak için pivot_table yöntemini kullanabiliriz.

In [78]:

```
mean_ratings = data.pivot_table('rating', index='title',
                                 columns='gender', aggfunc='mean')
```

In [79]:

mean_ratings[:5]

Out[79]:

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
'burbs, The (1989)	2.793478	2.962085
...And Justice for All (1979)	3.828571	3.689024

Bu, satır etiketleri ("dizin") olarak film başlıklarını ve sütun etiketleri olarak cinsiyet içeren ortalama derecelendirmeleri içeren başka bir DataFrame üretti. Önce en az 250 derecelendirme (tamamen rastgele bir sayı) alan filmleri filtreliyorum; Bunu yapmak için, verileri başlığa göre grupperiyorum ve her başlık için bir dizi grup boyutu elde etmek için boyut () kullanıyorum:

In [80]:

```
ratings_by_title = data.groupby('title').size()
```

In [81]:

```
ratings_by_title[:10]
```

Out[81]:

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 [82]:

```
active_titles = ratings_by_title.index[ratings_by_title >= 250]
```

In [83]:

```
active_titles
```

Out[83]:

```
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 mean_ratings'den satır seçmek için kullanılabilir.

In [84]:

```
mean_ratings = mean_ratings.loc[active_titles]
```

In [85]:

mean_ratings

Out[85]:

gender	F	M
	title	
'burbs, The (1989)	2.793478	2.962085
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 izlenen filmleri görmek için F sütununa göre azalan sıradan sıralayabiliriz.

In [86]:

top_female_ratings = mean_ratings.sort_values(by='F', ascending=False)

In [87]:

```
top_female_ratings[:10]
```

Out[87]:

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 izleyicileri en çok bölen filmleri bulmak istedığınızı varsayıyorum. Bunun bir yolu, ortalamalardaki farklılığı içeren ortalama puanlara bir sütun eklemek ve ardından buna göre sıralamaktır.

In [88]:

```
mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']
```

Kadınlar tarafından hangilerinin tercih edildiğini görebilmemiz için 'farklı'ya göre sıralama, reyting farkı en yüksek olan filmleri verir.

In [89]:

```
sorted_by_diff = mean_ratings.sort_values(by='diff')
```

In [90]:

sorted_by_diff[:10]

Out[90]:

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 çevirip yine ilk 10 sırayı bölerek, erkeklerin tercih ettiği ve kadınların pek de beğenmediği filmleri elde ederiz.

In [91]:

```
sorted_by_diff[::-1][:10]
```

Out[91]:

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 anlaşmazlığı ortaya çıkan filmleri istedığınızı varsayalım. Uyuşmazlıklar, derecelendirmelerin varyansı veya standart sapması ile ölçülebilir.

In [92]:

```
rating_std_by_title = data.groupby('title')['rating'].std()# Başlığa göre gruplandı
```

In [93]:

```
rating_std_by_title = rating_std_by_title.loc[active_titles]# Active_titles'a göre i
```

In [94]:

```
rating_std_by_title.sort_values(ascending=False)[:10]# Azalan sırayla değere göre Se
```

Out[94]:

title	rating
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	

Film türlerinin dikey çizgi ile ayrılmış (|) dizge olarak verildiğini fark etmiş olabilirsiniz. Türe göre bazı analizler yapmak isterseniz, tür bilgisini daha kullanışlı bir forma dönüştürmek için daha fazla çalışma yapmanız gerekecektir.

14.3 ABD Bebek İsimleri 1880–2010

Amerika Birleşik Devletleri Sosyal Güvenlik İdaresi (SSA), 1880'den günümüze kadar bebek isimlerinin sıklığı ile ilgili verileri kullanıma sundu. Birkaç popüler R paketinin yazarı olan Hadley Wickham, R'deki veri manipülasyonunu göstermek için bu veri setini sık sık kullanmıştır. Bu veri setini yüklemek için bazı veri tartışmaları yapmamız gereklidir, ancak bunu yaptığımızda aşağıdaki daki gibi görünen bir DataFrame'e sahip olacağız. bu:

```
names.head(10)
      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 var:

- Belirli bir ad (kendi adınız veya başka bir ad) verilen bebeklerin oranını zaman içinde görselleştirin
- Bir adın göreceli sırasını belirleyin
- Her yıl en popüler isimleri veya popülerliği en çok artan veya azalan isimleri belirleyin
- İsimlerdeki eğilimleri analiz edin: ünlüler, ünsüzler, uzunluk, genel çeşitlilik, yazım değişiklikleri, ilk ve son harfler
- Dış eğilim kaynaklarını analiz edin: İncil isimleri, ünlüler, demografik değişimler

In [95]:

```
!head -n 10 datasets/babynames/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 zaten virgülle ayrılmış güzel bir biçimde olduğundan, pandas.read_csv ile Veri Çerçevesine yüklenebilir.

In [96]:

```
import pandas as pd
```

In [97]:

```
names1880 = pd.read_csv('datasets/babynames/yob1880.txt',
                        names=['name', 'sex', 'births'])
```

In [98]:

```
names1880
```

Out[98]:

	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 sadece her yıl en az beş kez meydana gelen isimleri içerir, bu yüzden basitlik uğruna, o yıldaki toplam doğum sayısı olarak cinsiyete göre doğumların toplamını kullanabiliriz.

In [99]:

```
names1880.groupby('sex').births.sum()
```

Out[99]:

sex	
F	90993
M	110493
Name:	births, dtype: int64

Veri kümesi yıllara göre dosyalara bölündüğünden, yapılacak ilk şeylerden biri, tüm verileri tek bir DataFrame'de toplamak ve ayrıca bir yıl alanı eklemektir. Bunu pandas.concat kullanarak yapabilirsiniz.

In [100]:

```
years = range(1880, 2011)

pieces = []
columns = ['name', 'sex', 'births']

for year in years:
    path = 'datasets/babynames/yob%d.txt' % year
    frame = pd.read_csv(path, names=columns)

    frame['year'] = year
    pieces.append(frame)

# Her şeyi tek bir DataFrame'de birleştirin

names = pd.concat(pieces, ignore_index=True)
```

Burada dikkat edilmesi gereken birkaç nokta var. İlk olarak, concat'in Veri Çerçevesi nesnelerini varsayılan olarak satır şeklinde birbirine yapıştırıldığıını unutmayın. İkinci olarak, ignore_index = True olarak geçmelisiniz çünkü read_csv'den döndürülen orijinal satır numaralarını korumakla ilgilenmiyoruz. Şimdi tüm isim verilerini içeren çok büyük bir DataFrame'imiz var.

In [101]:

names

Out[101]:

	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
...
1690779	Zymaire	M	5	2010
1690780	Zyonne	M	5	2010
1690781	Zyquarius	M	5	2010
1690782	Zyran	M	5	2010
1690783	Zzyzx	M	5	2010

1690784 rows × 4 columns

Elimizdeki bu verilerle, groupby veya pivot_table kullanarak verileri yıl ve cinsiyet düzeyinde toplamaya başlayabiliriz (Şekil 14-4).

In [102]:

```
total_births = names.pivot_table('births', index='year',
                                 columns='sex', aggfunc=sum)
```

In [103]:

total_births.tail()

Out[103]:

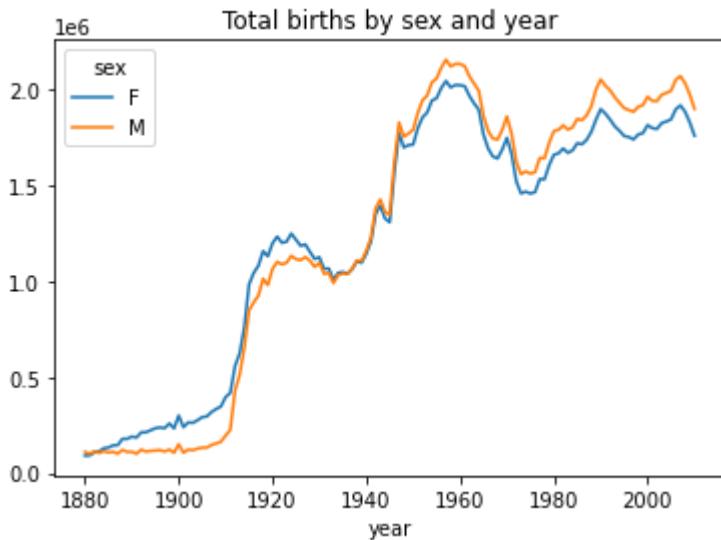
sex	F	M
year		
2006	1896468	2050234
2007	1916888	2069242
2008	1883645	2032310
2009	1827643	1973359
2010	1759010	1898382

In [104]:

```
total_births.plot(title='Total births by sex and year')
```

Out[104]:

```
<AxesSubplot:title={'center':'Total births by sex and year'}, xlabel='year'>
```



Daha sonra, toplam doğum sayısına göre her adın verildiği bebeklerin oranını içeren bir sütun desteği ekleyelim. 0,02'lik bir prop değeri, her 100 bebekten 2'sine belirli bir ad verildiğini gösterir. Bu nedenle, verileri yıla ve cinsiyete göre grupperlendiriyoruz ve ardından her gruba yeni sütunu ekliyoruz.

In [105]:

```
def add_prop(group):
    group['prop'] = group.births / group.births.sum()
    return group
names = names.groupby(['year', 'sex']).apply(add_prop)
```

Elde edilen eksiksiz veri kümesi artık aşağıdaki sütunlara sahiptir:

In [106]:

names

Out[106]:

	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
...
1690779	Zymaire	M	5	2010	0.000003
1690780	Zyonne	M	5	2010	0.000003
1690781	Zyquarius	M	5	2010	0.000003
1690782	Zyran	M	5	2010	0.000003
1690783	Zzyzx	M	5	2010	0.000003

1690784 rows × 5 columns

Bunun gibi bir grup işlemi gerçekleştirirken, tüm gruplar içinde destek sütu nunun toplamının 1 olduğunu doğrulamak gibi bir kontrol yapmak genellikle de gerlidir.

In [107]:

names.groupby(['year', 'sex']).prop.sum()

Out[107]:

year	sex	
1880	F	1.0
	M	1.0
1881	F	1.0
	M	1.0
1882	F	1.0
	...	
2008	M	1.0
2009	F	1.0
	M	1.0
2010	F	1.0
	M	1.0

Name: prop, Length: 262, dtype: float64

Şimdi bu yapıldığına göre, daha fazla analizi kolaylaştırmak için verilerin bir alt kümesini çıkaracağız, her cinsiyet / yıl kombinasyonu için ilk 1.00 0 isim. Bu yine başka bir grup işlemidir.

In [108]:

```
def get_top1000(group):
    return group.sort_values(by='births', ascending=False)[:1000]
grouped = names.groupby(['year', 'sex'])
top1000 = grouped.apply(get_top1000)
# Grup dizinini bırakın, gerekli değildir
top1000.reset_index(inplace=True, drop=True)
```

Bir kendin yap yaklaşımını tercih ediyorsanız, bunun yerine şunu deneyin:

In [109]:

```
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)
```

Ortaya çıkan veri kümesi artık biraz daha küçük.

In [110]:

top1000

Out[110]:

	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 × 5 columns

Bu ilk 1.000 veri kümesini, verilerle ilgili aşağıdaki araştırmalarda kullanacağız.

Adlandırma Trendlerini Analiz Etme

Elimizdeki tam veri kümesi ve ilk 1.000 veri kümesiyle, çeşitli adlandırma eğilimlerini analiz etmeye başlayabiliriz. İlk 1.000 ismi erkek ve kız böülümlerine ayırmak ilk önce kolaydır.

In [111]:

```
boys = top1000[top1000.sex == 'M']
```

In [112]:

```
girls = top1000[top1000.sex == 'F']
```

Her yıl için Johns veya Mary sayısı gibi basit zaman serileri çizilebilir, ancak daha kullanışlı olması için biraz parçalama gerektirir. Yıla ve isme göre toplam doğum sayısının bir pivot tablosunu oluşturalım.

In [113]:

```
total_births = top1000.pivot_table('births', index='year',
                                    columns='name',
                                    aggfunc=sum)
```

Şimdi, bu, DataFrame'in çizim yöntemiyle bir avuç ad için çizilebilir (Şekil 14-5 sonucu gösterir).

In [114]:

```
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 [115]:

```
subset = total_births[['John', 'Harry', 'Mary', 'Marilyn']]
```

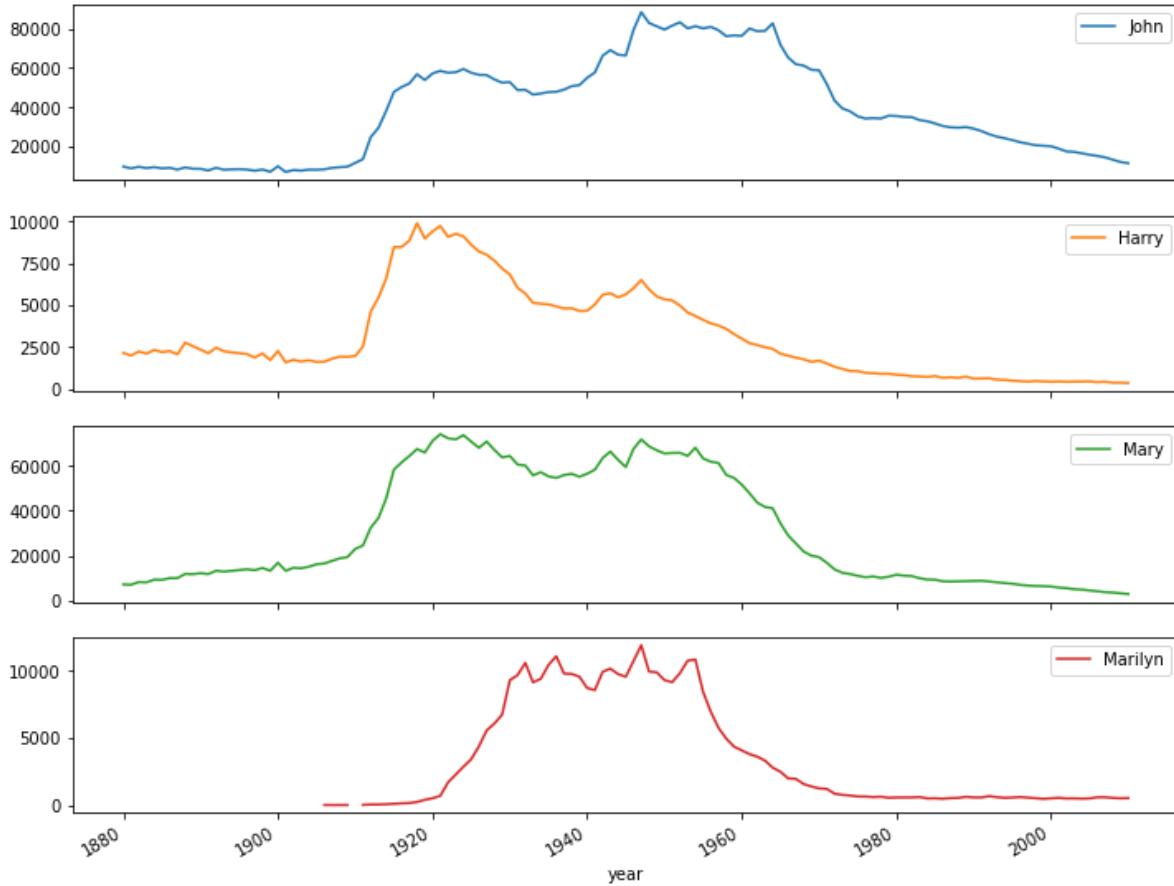
In [116]:

```
subset.plot(subplots=True, figsize=(12, 10), grid=False,
            title="Number of births per year")
```

Out[116]:

```
array([<AxesSubplot:xlabel='year'>, <AxesSubplot:xlabel='year'>,
       <AxesSubplot:xlabel='year'>, <AxesSubplot:xlabel='year'>],
      dtype=object)
```

Number of births per year



Buna baktığınızda, bu isimlerin Amerikan halkın gözünden düşüğü sonucuna varabilirsiniz. Ancak hikaye, bir sonraki bölümde inceleneceği gibi aslında bundan daha karmaşık.

Adlandırma çeşitliliğindeki artışı ölçmek

Konuların azalmasının bir açıklaması, daha az sayıda ebeveynin çocukları içi n ortak isimler seçmesidir. Bu hipotez, verilerde araştırılabılır ve doğrulanabilir. Bir ölçü, yıllara ve cinsiyete göre bir araya getirip grafiğini çizdiğim ilk 1.000 en popüler ismin temsil ettiği doğumların oranıdır (Şekil 14-6 sonuctaki durumu göstermektedir).

In [120]:

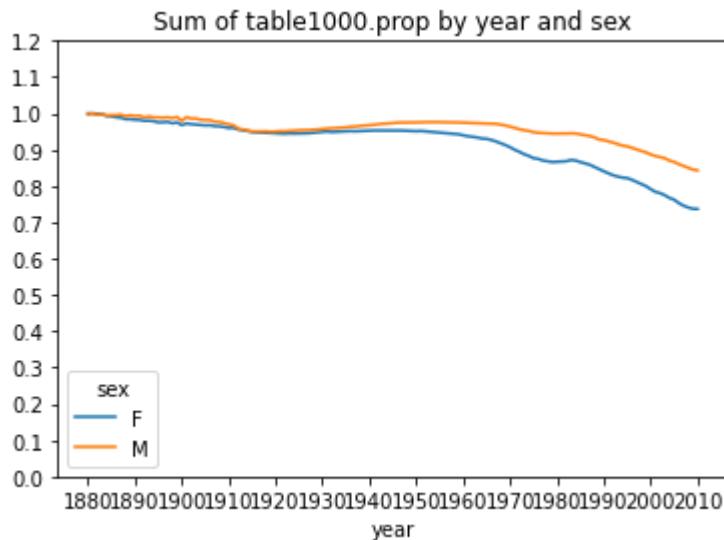
```
table = top1000.pivot_table('prop', index='year',
                           columns='sex', aggfunc=sum)
```

In [121]:

```
table.plot(title='Sum of table1000.prop by year and sex',
           yticks=np.linspace(0, 1.2, 13), xticks=range(1880, 2020, 10))
```

Out[121]:

```
<AxesSubplot:title={'center':'Sum of table1000.prop by year and sex'},
 xlabel='year'>
```



Aslında, isim çeşitliliğinin arttığını görebilirsiniz (ilk 1.000'deki toplam oran azalıyor). Bir başka ilginç ölçü de doğumların ilk% 50'sinde popülerlik sırasına göre en yüksekten en düşüğe doğru alınan farklı adların sayısıdır.

Bu sayıyı hesaplamak biraz daha zor. 2010'daki erkek isimlerini ele alalım:

In [122]:

```
df = boys[boys.year == 2010]
```

In [123]:

df

Out[123]:

		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 × 5 columns

Pervaneyi azalan sıraladıktan sonra, % 50'ye ulaşmak için en popüler isimlerden kaç tanesinin gerektiğini bilmek istiyoruz. Bunu yapmak için bir for döngüsü yazabilirsiniz, ancak vektörleştirilmiş bir NumPy yöntemi biraz da zekice. Pervanenin kümülatif toplamını, toplamını almak ve ardından aranın yöntemi çağırırmak, sıralı düzende tutmak için 0,5'in eklenmesi gereken kümülatif toplamdaki konumu döndürür.

In [125]:

prop_cumsum = df.sort_values(by='prop', ascending=False).prop.cumsum()

In [127]:

```
prop_cumsum[:10]
```

Out[127]:

```
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 [128]:

```
prop_cumsum.values.searchsorted(0.5)
```

Out[128]:

```
116
```

Diziler sıfır dizinli olduğundan, bu sonuca 1 eklemek size 117 sonucunu verir. Buna karşılık, 1900'de bu sayı çok daha küçüktü.

In [129]:

```
df = boys[boys.year == 1900]
```

In [130]:

```
in1900 = df.sort_values(by='prop', ascending=False).prop.cumsum()
```

In [131]:

```
in1900.values.searchsorted(0.5) + 1
```

Out[131]:

```
25
```

Artık bu işlemi her yıl / cinsiyet kombinasyonuna uygulayabilir, bu alanlara göre gruplandırabilir ve her grup için sayımı döndüren bir işlev uygulayabilirsiniz.

In [137]:

```
def get_quantile_count(group, q=0.5):
    group = group.sort_values(by='prop', ascending=False)
    return group.prop.cumsum().values.searchsorted(q) + 1

diversity = top1000.groupby(['year', 'sex']).apply(get_quantile_count)
diversity = diversity.unstack('sex')
```

Sonuçta ortaya çıkan bu DataFrame çeşitliliği artık her cinsiyet için bir tane olmak üzere yıllara göre indekslenmiş iki zaman serisine sahiptir. Bu, Python'da incelenebilir ve daha önce olduğu gibi çizilebilir (Şekil 14-7).

In [138]:

```
diversity.head()
```

Out[138]:

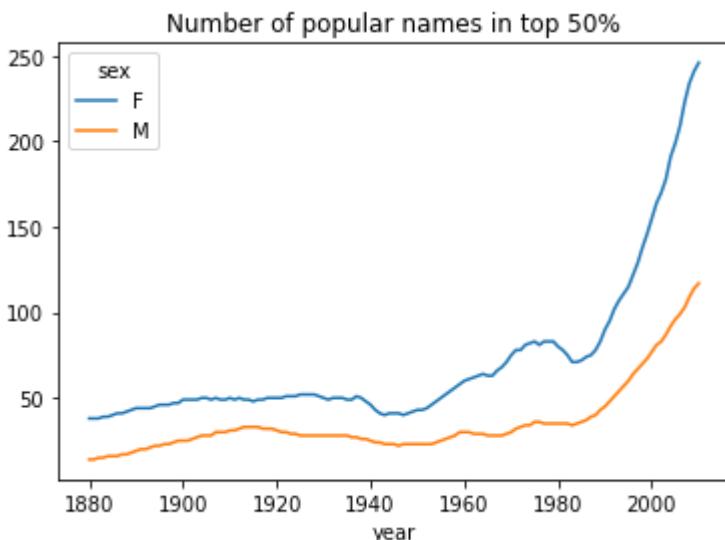
sex	F	M
year		
1880	38	14
1881	38	14
1882	38	15
1883	39	15
1884	39	16

In [139]:

```
diversity.plot(title="Number of popular names in top 50%)
```

Out[139]:

```
<AxesSubplot:title={'center':'Number of popular names in top 50%'}, xlabel='year'>
```



Gördüğünüz gibi, kız isimleri her zaman erkek isimlerinden daha çeşitli olmustur ve ancak zamanla daha da artmıştır. Alternatif yazımlarının artması gibi, çeşitliliği tam olarak neyin yönlendirdiğine dair daha fazla analiz okuyucuya bırakılmıştır.

"last letter" devrimi

2007'de bebek ismi araştırmacısı Laura Wattenberg, internet sitesinde erkek isimlerinin son harflere göre dağılımının son 100 yılda önemli ölçüde değiştiğine dikkat çekti. Bunu görmek için, ilk olarak tam veri kümelerindeki tüm doğumlara yıla, cinsiyete ve son mektuba göre topluyoruz.

In [142]:

```
# isim sütunundaki son harfi çıkar
get_last_letter = lambda x: x[-1]
last_letters = names.name.map(get_last_letter)
last_letters.name = 'last_letter'

table = names.pivot_table('births', index=last_letters,
                           columns=['sex', 'year'], aggfunc=sum)
```

Ardından, geçmişi kapsayan üç temsili yılı seçip ilk birkaç satırı yazdırıyoruz.

In [143]:

```
subtable = table.reindex(columns=[1910, 1960, 2010], level='year')
```

In [144]:

```
subtable.head()
```

Out[144]:

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 doğum oranını içeren yeni bir tablo hesaplamak için tabloyu toplam doğumlara göre normalleştiririn.

In [145]:

subtable.sum()

Out[145]:

```
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 [146]:

letter_prop = subtable / subtable.sum()

In [147]:

letter_prop

Out[147]:

sex	F			M		
year	1910	1960	2010	1910	1960	2010
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 × 6 columns

Artık elimizdeki harf oranlarıyla, her cinsiyet için yıllara göre ayrılmış çubuk grafikleri yapabiliriz (Şekil 14-8).

In [148]:

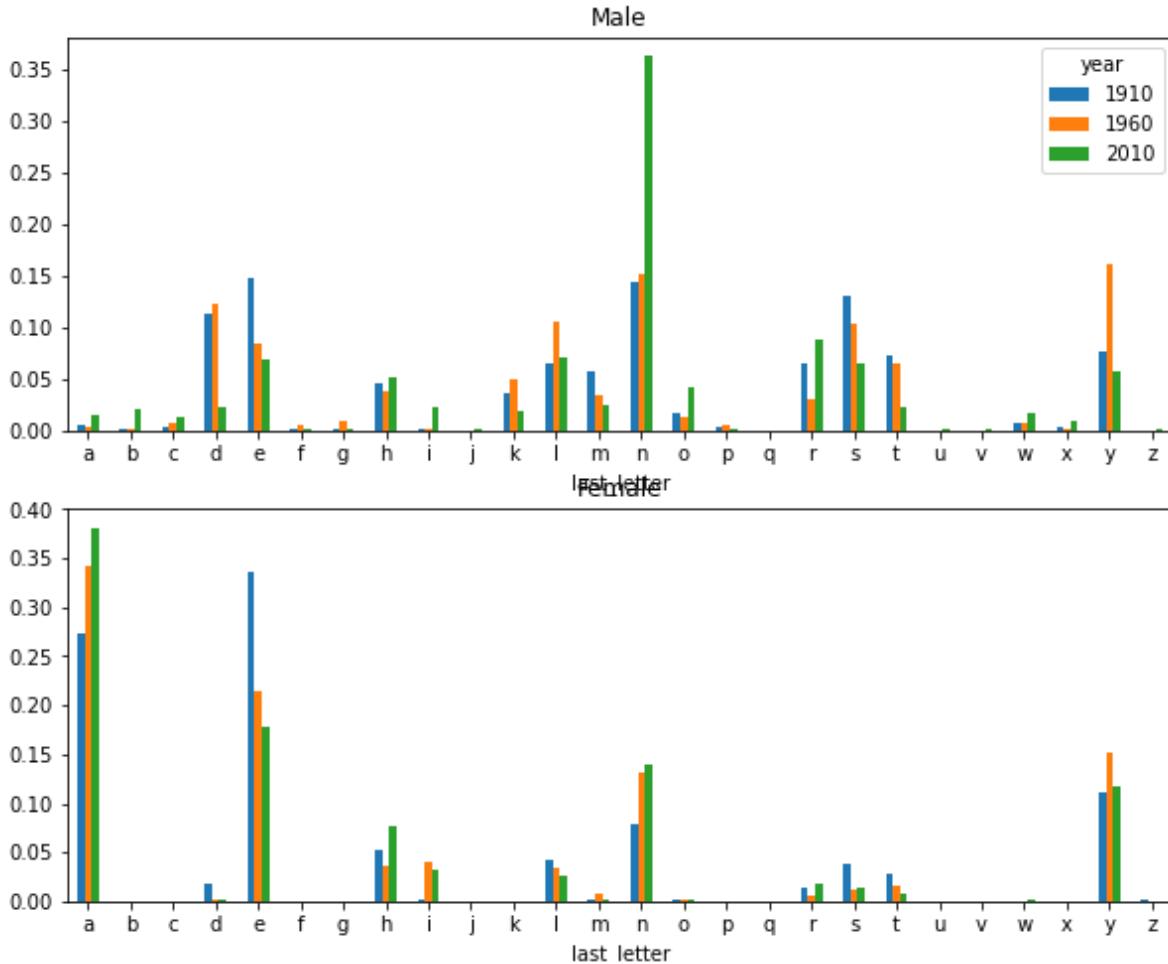
import matplotlib.pyplot as plt

In [151]:

```
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)
```

Out[151]:

```
<AxesSubplot:title={'center':'Female'}, xlabel='last_letter'>
```



Gördüğünüz gibi, n ile biten erkek isimleri 1960'lardan beri önemli bir büyümeye yaşamıştır. Daha önce oluşturulan tam tabloya geri dönersek, yine yillara ve cinsiyete göre normalleştiriyoruz ve erkek isimleri için bir harf alt kümlesi seçiyoruz, sonunda her bir sütunu bir zaman serisi yapmak için yer değiştiriyoruz.

In [152]:

```
letter_prop = table / table.sum()
```

In [153]:

```
dny_ts = letter_prop.loc[['d', 'n', 'y'], 'M'].T
```

In [154]:

dny_ts.head()

Out[154]:

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

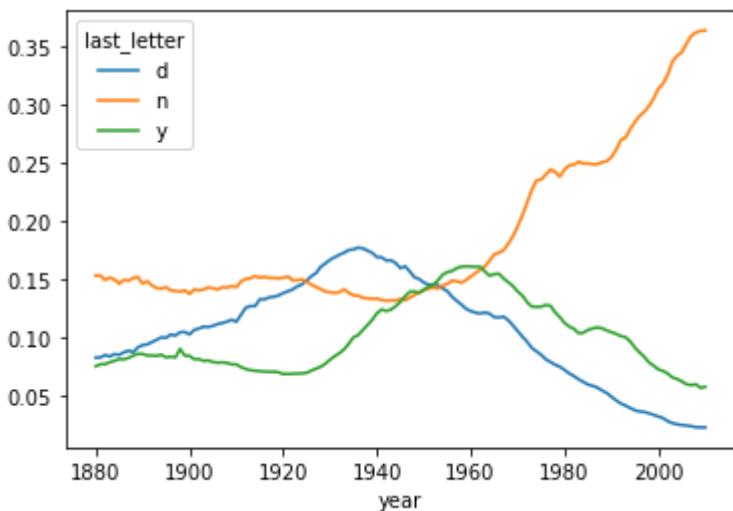
Elimizdeki bu DataFrame zaman serisiyle, çizim yöntemiyle tekrar zaman içind eki trendlerin bir grafiğini oluşturabiliriz (Şekil 14-9).

In [155]:

dny_ts.plot()

Out[155]:

<AxesSubplot:xlabel='year'>



Kız isimleri haline gelen erkek isimleri (ve tersi)

Bir başka eğlenceli trend ise, örnekteki bir cinsiyetten daha popüler olan, ancak günümüzde "cinsiyet değiştiren" erkek isimlerine bakmaktadır. Bir örnek Lesley veya Leslie adıdır. İlk 1000 DataFrame'e geri dönersek, "lesl" ile başlayan veri kümesinde meydana gelen adların bir listesini hesaplayalım.

In [156]:

```
all_names = pd.Series(top1000.name.unique())
```

In [157]:

```
lesley_like = all_names[all_names.str.lower().str.contains('lesl')]
```

In [158]:

```
lesley_like
```

Out[158]:

```
632      Leslie
2294     Lesley
4262     Leslee
4728     Lesli
6103     Lesly
dtype: object
```

Oradan, göreceli siklikları görmek için isme göre gruplandırılan isimleri ve toplam doğumları filtreleyebiliriz.

In [159]:

```
filtered = top1000[top1000.name.isin(lesley_like)]
```

In [160]:

```
filtered.groupby('name').births.sum()
```

Out[160]:

```
name
Leslee      1082
Lesley     35022
Lesli       929
Leslie    370429
Lesly      10067
Name: births, dtype: int64
```

Sonra, cinsiyete ve yıla göre toplayalım ve yıl içinde normale dönelim.

In [161]:

```
table = filtered.pivot_table('births', index='year',
                             columns='sex', aggfunc='sum')
```

In [162]:

```
table = table.div(table.sum(1), axis=0)
```

In [163]:

table.tail()

Out[163]:

	sex	F	M
year			
2006	1.0	Nan	
2007	1.0	Nan	
2008	1.0	Nan	
2009	1.0	Nan	
2010	1.0	Nan	

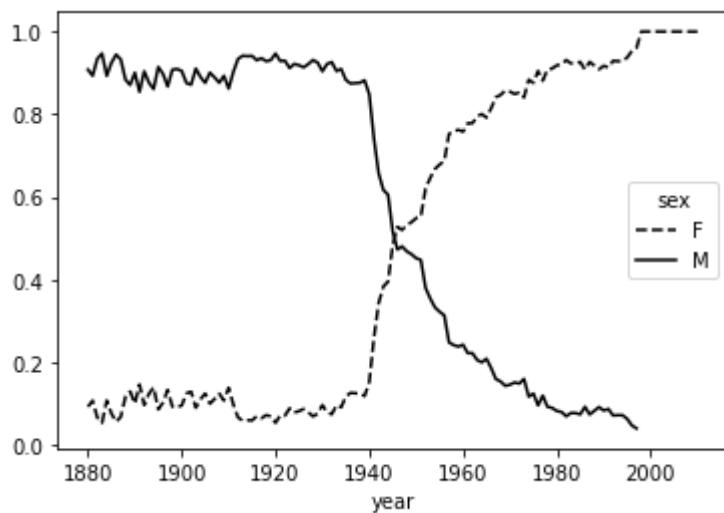
Son olarak, zaman içinde cinsiyete göre dağılımın bir grafiğini çıkarmak artık mümkün.

In [164]:

table.plot(style={'M': 'k-', 'F': 'k--'})

Out[164]:

<AxesSubplot:xlabel='year'>



14.4 USDA Gıda Veritabanı

ABD Tarım Bakanlığı, gıda besin maddesi bilgilerinin bir veri tabanını kullanıma sunuyor. Programcı Ashley Williams, bu veritabanının JSON formatında bir versiyonunu kullanıma sundu. Kayıtlar şuna benzer:

```
{
  "id": 21441,
  "description": "KENTUCKY FRIED CHICKEN, Fried Chicken, EXTRA CRISPY, Wing, meat and skin with breading",
  "tags": ["KFC"],
  "manufacturer": "Kentucky Fried Chicken",
  "group": "Fast Foods",
  "portions": [
    {
      "amount": 1,
      "unit": "wing, with skin",
      "grams": 68.0
    },
    ...
  ],
  "nutrients": [
    {
      "value": 20.8,
      "units": "g",
      "description": "Protein",
      "group": "Composition"
    },
    ...
  ]
}
```

Her yiyeceğin, iki besin listesi ve porsiyon boyutları ile birlikte bir dizi tanımlayıcı özelliği vardır. Bu formdaki veriler özellikle analiz için uygun değildir, bu nedenle verileri daha iyi bir forma dönüştürmek için biraz çalışmak gereklidir.

Verileri bağlantıdan indirip çıkardıktan sonra, seçtiğiniz herhangi bir JSON kitaplığıyla Python'a yükleyebilirsiniz. Yerleşik Python json modülünü kullanacağız.

In [192]:

```
import json
```

In [193]:

```
db = json.load(open('datasets/usda_food/database.json'))
```

In [194]:

```
len(db)
```

Out[194]:

6636

Db'deki her giriş, tek bir gıda için tüm verileri içeren bir diktördür. 'Besi

nler' alanı, her besin için bir tane olacak şekilde bir zorunluluk listesidir.

In [195]:

```
db[0].keys()
```

Out[195]:

```
dict_keys(['id', 'description', 'tags', 'manufacturer', 'group', 'portions', 'nutrients'])
```

In [196]:

```
db[0]['nutrients'][0]
```

Out[196]:

```
{'value': 25.18,
 'units': 'g',
 'description': 'Protein',
 'group': 'Composition'}
```

In [197]:

```
nutrients = pd.DataFrame(db[0]['nutrients'])
```

In [198]:

```
nutrients[:7]
```

Out[198]:

	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's listesini DataFrame'e dönüştürürken, ayıklanacak alanların bir listesini belirleyebiliriz. Yiyecek adlarını, grubunu, kimliğini ve üreticisini alacağız.

In [199]:

```
info_keys = ['description', 'group', 'id', 'manufacturer']
```

In [200]:

```
info = pd.DataFrame(db, columns=info_keys)
```

In [201]:

```
info[:5]
```

Out[201]:

	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 [202]:

```
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
```

Value_counts ile gıda gruplarının dağılımını görebilirsiniz.

In [203]:

```
pd.value_counts(info.group)[:10]
```

Out[203]:

Vegetables and Vegetable Products	812
Beef Products	618
Baked Products	496
Breakfast Cereals	403
Fast Foods	365
Legumes and Legume Products	365
Lamb, Veal, and Game Products	345
Sweets	341
Pork Products	328
Fruits and Fruit Juices	328
Name: group, dtype: int64	

Şimdi, tüm besin verileri üzerinde bazı analizler yapmak için en kolayı, her bir yiyeceğin besinlerini tek bir büyük masada toplamaktır. Bunu yapmak için birkaç adım atmamız gerekiyor. İlk olarak, her besin besinleri listesini bir DataFrame'e dönüştüreceğim, yiyecek kimliği için bir sütun ekleyeceğim ve DataFrame'i bir listeye ekleyeceğim. Ardından, bunlar concat ile birlikte birleştirilebilir.

Her şey yolunda giderse, besinler şöyle görünebilir:

In [204]:

```
nutrients
```

Out[204]:

	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 yinelenen öğeler var, bu yüzden onları siliyoruz.

In [205]:

```
nutrients.duplicated().sum() # kopya sayısı
```

Out[205]:

108

In [206]:

```
nutrients = nutrients.drop_duplicates()
```

Her iki DataFrame nesnesinde de "grup" ve "açıklama" olduğundan, netlik sağlamak için yeniden adlandırabiliriz.

In [207]:

```
col_mapping = {'description' : 'food',
               'group' : 'fgroup'}
```

In [208]:

```
info = info.rename(columns=col_mapping, copy=False)
```

In [209]:

```
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 [210]:

```
col_mapping = {'description' : 'nutrient',
               'group' : 'nutgroup'}
```

In [211]:

```
nutrients = nutrients.rename(columns=col_mapping, copy=False)
```

In [212]:

nutrients

Out[212]:

	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
...
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

54 rows × 4 columns

Tüm bunları yaptıktan sonra, bilgileri besinlerle birleştirmeye hazırız.

ndata = pd.merge(nutrients, info, on='id', how='outer')

ndata.info()

ndata.iloc[30000]

Artık gıda grubu ve besin türüne göre medyan değerlerin bir grafiğini oluşturabiliriz.

result = ndata.groupby(['nutrient', 'fgroup'])['value'].quantile(0.5)

result['Zinc, Zn'].sort_values().plot(kind='barh')

In [227]:

```
from IPython.display import Image
Image(filename='img/Picture85.png')
```

Out[227]:

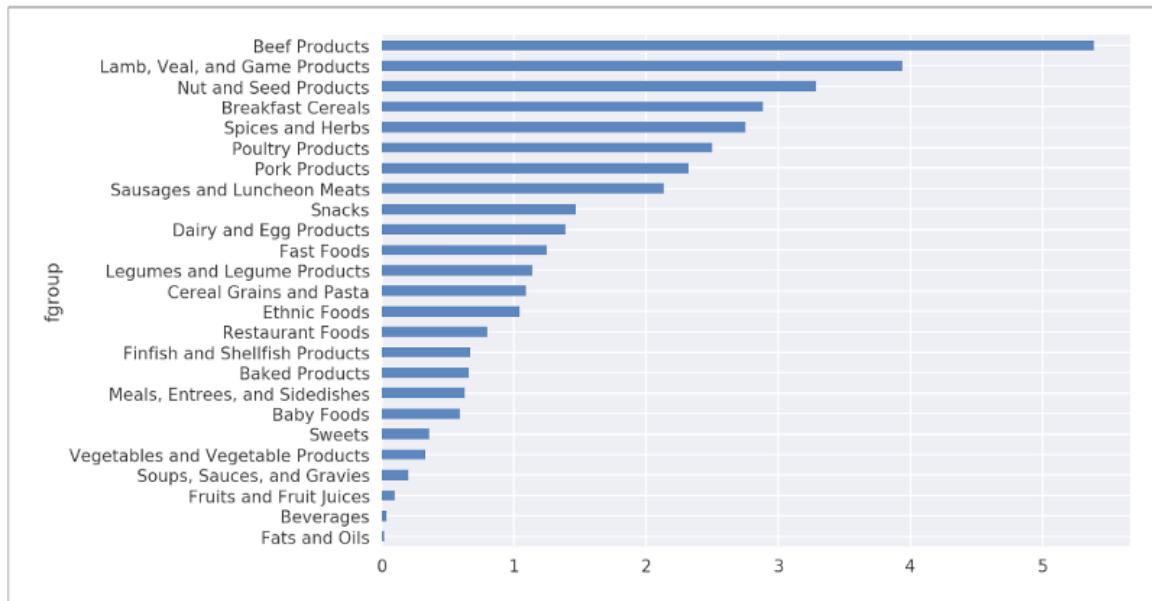


Figure 14-11. Median zinc values by nutrient group

Her besinde hangi yiyeceğin en yoğun olduğunu bulabilirsiniz.

```
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 küçült
max_foods.food = max_foods.food.str[:50]
```

Sonuçta elde edilen DataFrame, kitapta görüntülenemeyecek kadar büyük; işte sadece 'Amino Asitler' besin grubu.

```
max_foods.loc['Amino Acids']['food']
```

14.5 2012 Federal Seçim Komisyonu Veritabanı

ABD Federal Seçim Komisyonu, siyasi kampanyalara katkılarla ilgili verileri yayınlar. Buna katkıda bulunanların isimleri, mesleği ve işveren, adres ve katkı miktarı dahildir. İlginç bir veri seti, 2012 ABD başkanlık seçimlerinden. Haziran 2012'de indirdiğim veri kümесinin bir sürümü, pan das.read_csv ile yüklenebilen 150 megabaytlık bir CSV dosyası P00000001-ALL.csv'dir (kitabın veri havuzuna bakın).

In [4]:

```
import pandas as pd
fec = pd.read_csv('datasets/fec/P00000001-ALL.csv')
```

```
/Users/veyseldogan/opt/anaconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3071: DtypeWarning: Columns (6) have mixed types. Specify dtype option on import or set low_memory=False.
    has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

In [5]:

```
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          1001731 non-null   object 
 2   cand_nm          1001731 non-null   object 
 3   contbr_nm        1001731 non-null   object 
 4   contbr_city      1001712 non-null   object 
 5   contbr_st         1001727 non-null   object 
 6   contbr_zip        1001620 non-null   object 
 7   contbr_employer   988002 non-null   object 
 8   contbr_occupation 993301 non-null   object 
 9   contb_receipt_amt 1001731 non-null   float64
 10  contb_receipt_dt  1001731 non-null   object 
 11  receipt_desc      14166 non-null    object 
 12  memo_cd           92482 non-null   object 
 13  memo_text          97770 non-null   object 
 14  form_tp           1001731 non-null   object 
 15  file_num          1001731 non-null   int64  
dtypes: float64(1), int64(1), object(14)
memory usage: 122.3+ MB
```

DataFrame'deki örnek bir kayıt şuna benzer:

In [6]:

fec.iloc[123456]

Out[6]:

cmte_id	C00431445
cand_id	P80003338
cand_nm	Obama, Barack
contbr_nm	ELLMAN, IRA
contbr_city	TEMPE
contbr_st	AZ
contbr_zip	852816719
contbr_employer	ARIZONA STATE UNIVERSITY
contbr_occupation	PROFESSOR
contb_receipt_amt	50
contb_receipt_dt	01-DEC-11
receipt_desc	NaN
memo_cd	NaN
memo_text	NaN
form_tp	SA17A
file_num	772372

Name: 123456, dtype: object

Verilerde herhangi bir siyasi parti bağlantısı olmadığını görebilirsiniz, bu nedenle bunu eklemek faydalı olacaktır. Benzersiz olanı kullanarak tüm benzersiz siyasi adayların bir listesini alabilirsiniz.

In [8]:

unique_cands = fec.cand_nm.unique()

In [9]:

unique_cands

Out[9]:

```
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 [10]:

unique_cands[2]

Out[10]:

'Obama, Barack'

In [11]:

```
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'}
```

Şimdi, bu eşlemeyi ve Seri nesnelerdeki harita yöntemini kullanarak, aday isimlerinden bir dizi siyasi parti hesaplayabilirsiniz.

In [12]:

```
fec.cand_nm[123456:123461]
```

Out[12]:

```
123456    Obama, Barack
123457    Obama, Barack
123458    Obama, Barack
123459    Obama, Barack
123460    Obama, Barack
Name: cand_nm, dtype: object
```

In [13]:

```
fec.cand_nm[123456:123461].map(parties)
```

Out[13]:

```
123456    Democrat
123457    Democrat
123458    Democrat
123459    Democrat
123460    Democrat
Name: cand_nm, dtype: object
```

In [14]:

```
# Sütun olarak ekleyin
fec['party'] = fec.cand_nm.map(parties)
```

In [15]:

```
fec['party'] = fec.cand_nm.map(parties)
```

In [16]:

```
fec['party'].value_counts()
```

Out[16]:

```
Democrat      593746  
Republican    407985  
Name: party, dtype: int64
```

Birkaç veri hazırlama noktası. İlk olarak, bu veriler hem katkıları hem de gери ödemeleri içerir (negatif katkı tutarı).

In [17]:

```
(fec.contb_receipt_amt > 0).value_counts()
```

Out[17]:

```
True       991475  
False      10256  
Name: contb_receipt_amt, dtype: int64
```

In [18]:

```
fec = fec[fec.contb_receipt_amt > 0]
```

Barack Obama ve Mitt Romney ana iki aday olduğundan, kampanyalarına sadece katkıları olan bir alt küme de hazırlayalım.

In [19]:

```
fec_mrbo = fec[fec.cand_nm.isin(['Obama', 'Barack', 'Romney', 'Mitt'])]
```

Meslek ve İşverene Göre Bağış İstatistikleri

Mesleğe göre bağışlar, sıkça araştırılan bir başka istatistiktitir. Örneğin, avukatlar (avukatlar) Demokratlara daha fazla bağış yapma eğilimindeyken, şirket yöneticileri Cumhuriyetçilere daha fazla bağış yapma eğilimindedir. Birinci sınıfı, mesleğe göre toplam bağış sayısı kolaydır.

In [20]:

```
fec.contbr_occupation.value_counts()[:10]
```

Out[20]:

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

Birçoğunun aynı temel iş türüne atıfta bulunduğu mesleklerde veya aynı şeyin birkaç çeşidi olduğuna bakarak fark edeceksiniz. Aşağıdaki kod parçası, bir meslekten diğerine eşleyerek birkaç tanesini temizlemek için bir teknik göstermektedir; dict.get kullanmanın "hilesine", eşleştirme olmadan mesleklerde "geçmesine" izin vermeyi unutmayın.

In [24]:

```
occ_mapping = {
    'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
    'INFORMATION REQUESTED' : 'NOT PROVIDED',
    'INFORMATION REQUESTED (BEST EFFORTS)' : 'NOT PROVIDED',
    'C.E.O.' : 'CEO'
}

# Eşleşme sağlanmadıysa, x
f = lambda x: occ_mapping.get(x, x)
fec.contbr_occupation = fec.contbr_occupation.map(f)
```

In [25]:

```
emp_mapping = {
    'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
    'INFORMATION REQUESTED' : 'NOT PROVIDED',
    'SELF' : 'SELF-EMPLOYED',
    'SELF EMPLOYED' : 'SELF-EMPLOYED',
}

# Eşleşme sağlanmadıysa, x
f = lambda x: emp_mapping.get(x, x)
fec.contbr_employer = fec.contbr_employer.map(f)
```

Şimdi, verileri partiye ve mesleğe göre toplamak için pivot_table'ı kullanabilir, ardından toplamda en az 2 milyon ABD doları bağış yapan alt kümeye göre filtre uygulayabilirsiniz.

In [26]:

```
by_occupation = fec.pivot_table('contb_receipt_amt',
                                 index='contbr_occupation',
                                 columns='party', aggfunc='sum')
```

In [27]:

```
over_2mm = by_occupation[by_occupation.sum(1) > 2000000]
```

In [28]:

```
over_2mm
```

Out[28]:

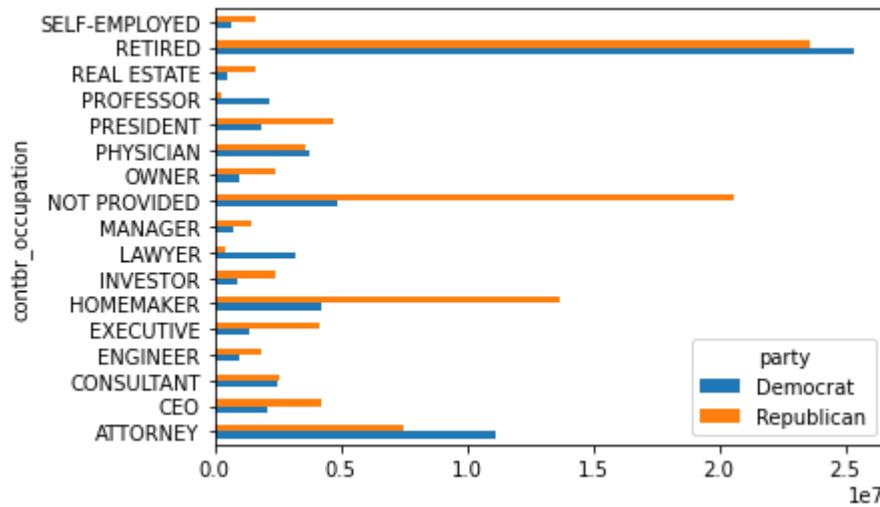
party	Democrat	Republican
contbr_occupation		
ATTORNEY	11141982.97	7.477194e+06
CEO	2074974.79	4.211041e+06
CONSULTANT	2459912.71	2.544725e+06
ENGINEER	951525.55	1.818374e+06
EXECUTIVE	1355161.05	4.138850e+06
HOMEMAKER	4248875.80	1.363428e+07
INVESTOR	884133.00	2.431769e+06
LAWYER	3160478.87	3.912243e+05
MANAGER	762883.22	1.444532e+06
NOT PROVIDED	4866973.96	2.056547e+07
OWNER	1001567.36	2.408287e+06
PHYSICIAN	3735124.94	3.594320e+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

In [29]:

```
over_2mm.plot(kind='barh')
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f984e13c100>
```



In [30]:

```
def get_top_amounts(group, key, n=5):
    totals = group.groupby(key)[ 'contb_receipt_amt' ].sum()
    return totals.nlargest(n)
```

Ardından mesleğe ve işverene göre toplayın.

In [31]:

```
grouped = fec_mrbo.groupby( 'cand_nm' )
```

In [32]:

```
grouped.apply(get_top_amounts, 'contbr_occupation', n=7)
```

Out[32]:

cand_nm	contbr_occupation	
Obama, Barack	RETIRED	25305116.38
	ATTORNEY	11141982.97
	INFORMATION REQUESTED	4866973.96
	HOMEMAKER	4248875.80
	PHYSICIAN	3735124.94
	LAWYER	3160478.87
	CONSULTANT	2459912.71
Romney, Mitt	RETIRED	11508473.59
	INFORMATION REQUESTED PER BEST EFFORTS	11396894.84
	HOMEMAKER	8147446.22
	ATTORNEY	5364718.82
	PRESIDENT	2491244.89
	EXECUTIVE	2300947.03
	C.E.O.	1968386.11

Name: contb_receipt_amt, dtype: float64

In [33]:

```
grouped.apply(get_top_amounts, 'contbr_employer', n=10)
```

Out[33]:

cand_nm	contbr_employer	
Obama, Barack	RETIRED	22694358.85
	SELF-EMPLOYED	17080985.96
	NOT EMPLOYED	8586308.70
	INFORMATION REQUESTED	5053480.37
	HOMEMAKER	2605408.54
	SELF	1076531.20
	SELF EMPLOYED	469290.00
	STUDENT	318831.45
	VOLUNTEER	257104.00
	MICROSOFT	215585.36
Romney, Mitt	INFORMATION REQUESTED PER BEST EFFORTS	12059527.24
	RETIRED	11506225.71
	HOMEMAKER	8147196.22
	SELF-EMPLOYED	7409860.98
	STUDENT	496490.94
	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, dtype: float64

Paketleme Bağış Miktarları

Bu verileri analiz etmenin kullanışlı bir yolu, katkı paylarını katkı boyutuna göre böümlere ayırmak için kesme işlevini kullanmaktır.

In [35]:

```
import numpy as np
bins = np.array([0, 1, 10, 100, 1000, 10000,
                 100000, 1000000, 10000000])
```

In [36]:

```
labels = pd.cut(fec_mrbo.contb_receipt_amt, bins)
```

In [37]:

```
labels
```

Out[37]:

```
411      (10, 100]
412      (100, 1000]
413      (100, 1000]
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] < (1000, 10000] < (10000, 100000] < (100000, 1000000] < (100000
0, 10000000]]
```

Bağış boyutuna göre bir histogram elde etmek için Obama ve Romney verilerini ada ve çöp kutusu etiketine göre gruplayabiliriz.

In [39]:

```
grouped = fec_mrbo.groupby(['cand_nm', labels])
```

In [40]:

```
grouped.size().unstack(0)
```

Out[40]:

cand_nm	Obama, Barack	Romney, Mitt
contb_receipt_amt		
(0, 1]	493	77
(1, 10]	40070	3681
(10, 100]	372280	31853
(100, 1000]	153991	43357
(1000, 10000]	22284	26186
(10000, 100000]	2	1
(100000, 1000000]	3	0
(1000000, 10000000]	4	0

Bu veriler, Obama'nın Romney'den çok daha fazla sayıda küçük bağış aldığıını gösteriyor. Ayrıca, adaya göre her boyuttaki toplam bağış yüzdesini görsell estirmek için katkı miktarlarını toplayabilir ve kovalar içinde normalleştir ebilirsiniz.

In [41]:

```
bucket_sums = grouped.contb_receipt_amt.sum().unstack(0)
```

In [42]:

```
normed_sums = bucket_sums.div(bucket_sums.sum(axis=1), axis=0)
```

In [43]:

normed_sums

Out[43]:

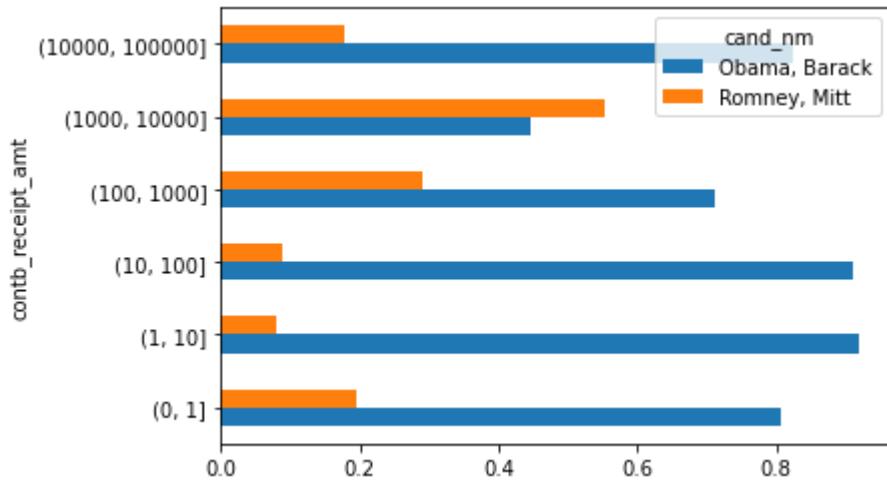
cand_nm	Obama, Barack	Romney, Mitt
contb_receipt_amt		
(0, 1]	0.805182	0.194818
(1, 10]	0.918767	0.081233
(10, 100]	0.910769	0.089231
(100, 1000]	0.710176	0.289824
(1000, 10000]	0.447326	0.552674
(10000, 100000]	0.823120	0.176880
(100000, 1000000]	1.000000	NaN
(1000000, 10000000]	1.000000	NaN

In [44]:

normed_sums[:2].plot(kind='barh')

Out[44]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f984e2b3310>



Bu analiz birçok yönden iyileştirilebilir ve geliştirilebilir. Örneğin, bir veya daha fazla büyük bağış yerine çok sayıda küçük miktarlar veren bağışçıları ayarlamak için bağışçı adı ve posta koduna göre bağışları toplayabilirsınız.

Devlete Göre Bağış İstatistikleri

Verilerin adaya ve devlete göre toplanması rutin bir meseledir.

In [45]:

```
grouped = fec_mrbo.groupby(['cand_nm', 'contbr_st'])
```

In [46]:

```
totals = grouped.contb_receipt_amt.sum().unstack(0).fillna(0)
```

In [48]:

```
totals = totals[totals.sum(1) > 100000]
```

In [49]:

```
totals[:10]
```

Out[49]:

cand_nm	Obama, Barack	Romney, Mitt
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

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 miktarına bölerseniz, her aday için eyalete göre toplam bağışların görelî yüzdesini elde edersiniz.

In [50]:

```
percent = totals.div(totals.sum(1), axis=0)
```

In [51]:

```
percent[:10]
```

Out[51]:

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

Gelişmiş NumPy

A.1 ndarray Object Internals

NumPy ndarray, homojen bir veri bloğunu (bitişik veya çok yönlü) çok boyutlu bir dizi nesnesi olarak yorumlamak için bir yol sağlar. Veri türü veya dtype, verilerin kayan nokta, tamsayı, boole veya bakmakta olduğumuz diğer türlerden herhangi biri olarak nasıl yorumlanacağını belirler.

Ndarray'i esnek kılan şeylelerden biri, her dizi nesnesinin bir veri bloğu üzerinde aşamalı bir görünüm olmasıdır. Örneğin, dizi görünümünün `[:: 2, :: -1]` herhangi bir veriyi nasıl kopyalamadığını merak edebilirsiniz. Bunun nedeni, ndarray'in bir bellek parçası ve bir dtype'dan daha fazlası olmasıdır; ayrıca dizinin farklı adım boyutlarıyla bellekte hareket etmesini sağlayan "adım adım" bilgisine sahiptir. Daha doğrusu, ndarray dahili olarak aşağıda kilerden oluşur.

- Veriye işaretçi - yani RAM'deki veya bellek eşlemeli bir dosyadaki bir veri bloğu
- Dizideki sabit boyutlu değerli hücreleri açıklayan veri türü veya dtype
- Dizinin şeklini gösteren bir demet
- Bir boyut boyunca bir öğeyi ilerletmek için "adım" atılacak bayt sayısını belirten tam sayılarından oluşan bir adım dizisi

Örneğin, bir 10×5 dizisinin şekli `(10, 5)` olacaktır.

In [2]:

```
import numpy as np
np.ones((10, 5)).shape
```

Out[2]:

```
(10, 5)
```

Tipik bir (C sırası) $3 \times 4 \times 5$ float64 (8 bayt) değer dizisi adımlara sahiptir $(160, 40, 8)$ (adımların bilinmesi yararlı olabilir çünkü genel olarak belirli bir eksendeki adımlar ne kadar büyükse, bu eksen boyunca hesaplama yapmak daha maliyetli olur).

In [3]:

```
np.ones((3, 4, 5), dtype=np.float64).strides
```

Out[3]:

```
(160, 40, 8)
```

unlar "sıfır kopya" dizi görünümlerinin oluşturulmasında kritik bileşendir.

Adımlar negatif bile olabilir, bu da bir dizinin bellekte "geri" hareket etmesini sağlar (örneğin, `obj [:: - 1]` veya `obj [:, :: - 1]` gibi bir dilimde böyle bir durum söz konusudur).

In [8]:

```
from IPython.display import Image
Image(filename='img/Picture86.png')
```

Out[8]:

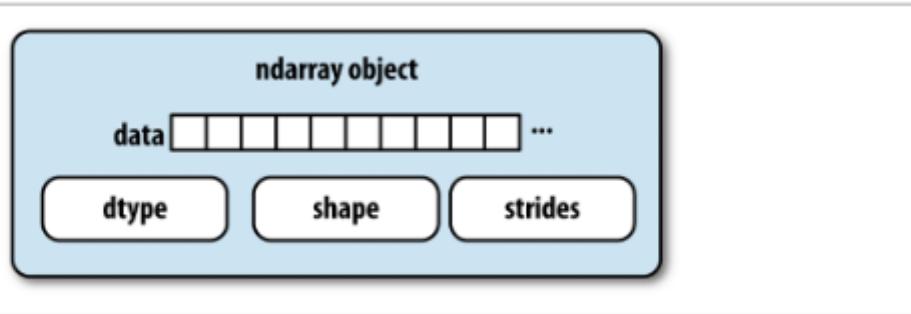


Figure A-1. The NumPy ndarray object

NumPy dtype Hiyerarşisi

Ara sıra, bir dizinin tamsayılar, kayan nokta sayıları, dizeler veya Python nesneleri içerip içermediğini kontrol etmesi gereken koda sahip olabilirsiniz. Birden fazla kayan nokta sayısı türü olduğundan (`float16`'dan `float128`'e kadar), `dtype`'ın bir tür listesi arasında olup olmadığını kontrol etmek çok ayrıntılı olacaktır. Neyse ki, `dtype`'lerde `np.integer` ve `np.floating` gibi super sınıflar vardır ve bunlar `np.issubdtype` işlevi ile birlikte kullanılabilir.

In [9]:

```
ints = np.ones(10, dtype=np.uint16)
```

In [11]:

```
floats = np.ones(10, dtype=np.float32)
```

In [12]:

```
np.issubdtype(ints.dtype, np.integer)
```

Out[12]:

True

In [13]:

```
np.issubdtype(floats.dtype, np.floating)
```

Out[13]:

True

type's mro yöntemini çağırarak belirli bir dtype'ın tüm üst sınıflarını göre bilirsiniz.

In [14]:

```
np.float64.mro()
```

Out[14]:

```
[numpy.float64,
 numpy.floating,
 numpy.inexact,
 numpy.number,
 numpy.generic,
 float,
 object]
```

In [15]:

```
np.issubdtype(ints.dtype, np.number)
```

Out[15]:

True

Dtype hiyerarşisi ve üst-alt sınıf ilişkilerinin bir grafiği için Şekil A-2

In [20]:

```
Image(filename='img/Picture87.png')
```

Out[20]:

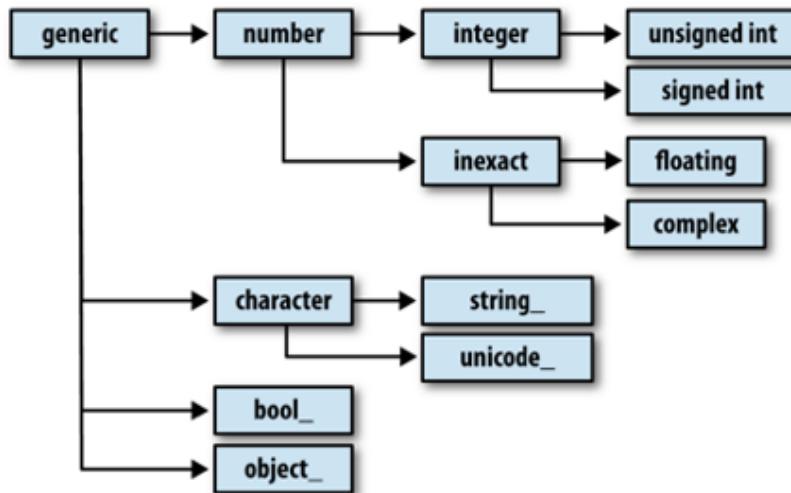


Figure A-2. The NumPy dtype class hierarchy

A.2 Gelişmiş Dizi Manipülasyonu

Dizileri Yeniden Şekillendirme(Reshaping Arrays)

Çoğu durumda, herhangi bir veri kopyalamadan bir diziyi bir şeilden diğerine dönüştürebilirsiniz. Bunu yapmak için, diziyi yeniden şekillendirme örneği ne yeni şekli gösteren bir demet iletin. Örneğin, bir matris halinde yeniden düzenlemek istediğimiz tek boyutlu bir değerler dizimiz olduğunu varsayıyalım (sonuç Şekil A-3'te gösterilmektedir).

In [17]:

```
arr = np.arange(8)
```

In [18]:

```
arr
```

Out[18]:

```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

In [19]:

arr.reshape((4, 2))

Out[19]:

array([[0, 1],
 [2, 3],
 [4, 5],
 [6, 7]])

In [21]:

Image(filename='img/Picture88.png')

Out[21]:

0	1	2	3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	---	---	---	----	----

arr.reshape((4, 3), order=?)

C order (row major)

0	1	2
3	4	5
6	7	8
9	10	11

order='C'

Fortran order (column major)

0	4	8
1	5	9
2	6	10
3	7	11

order='F'

Figure A-3. Reshaping in C (row major) or Fortran (column major) order

Çok boyutlu bir dizi de yeniden şekillendirilebilir.

In [22]:

arr.reshape((4, 2)).reshape((2, 4))

Out[22]:

array([[0, 1, 2, 3],
 [4, 5, 6, 7]])

Aktarılan şekil boyutlarından biri -1 olabilir; bu durumda, o boyut için kullanılan değer verilerden çıkarılır.

In [23]:

arr = np.arange(15)

In [24]:

```
arr.reshape((5, -1))
```

Out[24]:

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

Bir dizinin şekil özniteliği bir demet olduğundan, yeniden şekillendirmek işin de iletilebilir.

In [25]:

```
other_arr = np.ones((3, 5))
```

In [26]:

```
other_arr.shape
```

Out[26]:

```
(3, 5)
```

In [27]:

```
arr.reshape(other_arr.shape)
```

Out[27]:

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

Tek boyutlu dan daha yüksek bir boyuta yeniden şekillendirmenin zıt işlemi tipik olarak düzleştirme veya açılma olarak bilinir.

In [28]:

```
arr = np.arange(15).reshape((5, 3))
```

In [29]:

```
arr.ravel()
```

Out[29]:

```
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. Yassılaştırma yöntemi, her zaman verilerin bir kopyasını dönürtmesi dışında ravel gibi davranışır.

In [30]:

```
arr.flatten()
```

Out[30]:

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

Veriler, farklı sıralarda yeniden şekillendirilebilir veya yeniden şekillendirilebilir.

C Fortran Order Kıyaslaması

NumPy, bellekteki verilerinizin düzeni üzerinde size kontrol ve esneklik sağlar. Varsayılan olarak, NumPy dizileri satır ana sırasına göre oluşturulur.

Uzamsal olarak bu, iki boyutlu bir veri diziniz varsa, dizinin her satırında aki öğelerin bitişik bellek konumlarında depolandığı anlamına gelir. Satır büyük sıralamasına alternatif, sütun büyük sıralamadır; bu, her veri sütunundaki değerlerin bitişik bellek konumlarında depolandığı anlamına gelir.

Tarihsel nedenlerden dolayı, satır ve sütun ana sırası sırasıyla C ve Fortran sırası olarak da bilinir. FORTRAN 77 dilinde, matrislerin tümü sütun temelidir.

Reshape ve ravel gibi işlevler, dizideki verileri kullanma sırasını belirten bir sıra bağımsız değişkenini kabul eder. Bu, çoğu durumda genellikle 'C' ve ya 'F' olarak ayarlanır.

In [33]:

```
arr = np.arange(12).reshape((3, 4))
```

In [34]:

```
arr
```

Out[34]:

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

In [35]:

```
arr.ravel()
```

Out[35]:

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

In [36]:

arr.ravel('F')

Out[36]:

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 düzeni arasındaki temel fark, boyutların yüründüğü yoldur.

C / satır ana sırası

Önce daha yüksek boyutları geçin (örneğin, eksen 0'a ilerlemeden önce eksen 1).

Fortran / sütun ana sırası

En son daha yüksek boyutları geçin (örneğin, eksen 1'e ilerlemeden önce eksen 0).

Dizileri Birleştirme ve Bölme

`numpy.concatenate`, dizi dizilerini (tuple, list, vb.) alır ve bunları giriş ekseni boyunca sırayla birleştirir.

In [37]:

arr1 = np.array([[1, 2, 3], [4, 5, 6]])

In [38]:

arr2 = np.array([[7, 8, 9], [10, 11, 12]])

In [39]:

np.concatenate([arr1, arr2], axis=0)

Out[39]:

array([[1, 2, 3],
 [4, 5, 6],
 [7, 8, 9],
 [10, 11, 12]])

In [40]:

np.concatenate([arr1, arr2], axis=1)

Out[40]:

array([[1, 2, 3, 7, 8, 9],
 [4, 5, 6, 10, 11, 12]])

Yaygın birleştirme türleri için `vstack` ve `hstack` gibi bazı kullanışlı işlevler vardır. Önceki işlemler şu şekilde ifade edilebilirdi:

In [41]:

```
np.vstack((arr1, arr2))
```

Out[41]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [42]:

```
np.hstack((arr1, arr2))
```

Out[42]:

```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

`split`, bir diziyi bir eksen boyunca birden çok diziye böler.

In [43]:

```
arr = np.random.randn(5, 2)
```

In [44]:

```
arr
```

Out[44]:

```
array([[ 1.26295214,  0.87440954],
       [-0.38438596, -0.90336778],
       [ 0.62775591, -1.15367278],
       [ 0.44059987, -0.24521833],
       [ 0.006977,   0.32898347]])
```

In [45]:

```
first, second, third = np.split(arr, [1, 3])
```

In [46]:

```
first
```

Out[46]:

```
array([[1.26295214, 0.87440954]])
```

In [47]:

second

Out[47]:

```
array([[-0.38438596, -0.90336778],
       [ 0.62775591, -1.15367278]])
```

In [48]:

third

Out[48]:

```
array([[ 0.44059987, -0.24521833],
       [ 0.006977 ,  0.32898347]])
```

Np.split'e aktarılan [1, 3] değeri, dizinin parçalara bölüneceği endeksleri belirtir.

Tüm ilgili birleştirme ve bölme işlevlerinin bir listesi için Tablo A-1'e bakın, bunlardan bazıları yalnızca çok genel amaçlı birleştirme kolaylığı olarak sunulmuştur.

In [49]:

```
Image(filename='img/Picture89.png')
```

Out[49]:

Table A-1. Array concatenation functions

Function	Description
concatenate	Most general function, concatenates collection of arrays along one axis
vstack, ravel_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

İstiflme yardımcıları: r_ ve c_

NumPy ad alanında, yığınlama dizilerini daha kısa hale getiren iki özel nesne vardır: r_ ve c_.

In [50]:

```
arr = np.arange(6)
```

In [51]:

```
arr1 = arr.reshape((3, 2))
```

In [52]:

```
arr2 = np.random.randn(3, 2)
```

In [53]:

```
np.r_[arr1, arr2]
```

Out[53]:

```
array([[ 0.          ,  1.          ],
       [ 2.          ,  3.          ],
       [ 4.          ,  5.          ],
       [ 0.79683449, -0.73581472],
       [ 1.61452779, -0.23174841],
       [ 0.08537535,  0.64332375]])
```

In [54]:

```
np.c_[np.r_[arr1, arr2], arr]
```

Out[54]:

```
array([[ 0.          ,  1.          ,  0.          ],
       [ 2.          ,  3.          ,  1.          ],
       [ 4.          ,  5.          ,  2.          ],
       [ 0.79683449, -0.73581472,  3.          ],
       [ 1.61452779, -0.23174841,  4.          ],
       [ 0.08537535,  0.64332375,  5.          ]])
```

Bunlar ayrıca dilimleri dizilere çevirebilir.

In [55]:

```
np.c_[1:6, -10:-5]
```

Out[55]:

```
array([[ 1, -10],
       [ 2, -9],
       [ 3, -8],
       [ 4, -7],
       [ 5, -6]])
```

Yinelenen Öğeler: döşeyin ve tekrarlayın

Daha büyük diziler oluşturmak için dizileri yinelemek veya çoğaltmak için ik

i kullanışlı araç, yineleme ve döşeme işlevleridir. tekrar, bir dizideki her bir öğeyi birkaç kez çoğaltır ve daha büyük bir dizi oluşturur.

In [56]:

```
arr = np.arange(3)
```

In [57]:

```
arr
```

Out[57]:

```
array([0, 1, 2])
```

In [58]:

```
arr.repeat(3)
```

Out[58]:

```
array([0, 0, 0, 1, 1, 1, 2, 2, 2])
```

Varsayılan olarak, bir tamsayı geçirirseniz, her öğe bu sayıda tekrarlanacaktır. Bir tamsayı dizisi geçirirseniz, her öğe farklı sayıda tekrarlanabilir.

In [59]:

```
arr.repeat([2, 3, 4])
```

Out[59]:

```
array([0, 0, 1, 1, 1, 2, 2, 2, 2])
```

Çok boyutlu dizilerin elemanları belirli bir eksen boyunca tekrarlanabilir.

In [60]:

```
arr = np.random.randn(2, 2)
```

In [61]:

```
arr
```

Out[61]:

```
array([[ 0.3189704 , -0.66895566],
       [ 0.89103869,  0.71390849]])
```

In [62]:

```
arr.repeat(2, axis=0)
```

Out[62]:

```
array([[ 0.3189704 , -0.66895566],
       [ 0.3189704 , -0.66895566],
       [ 0.89103869,  0.71390849],
       [ 0.89103869,  0.71390849]])
```

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 dizisi geçirebilirsiniz.

In [63]:

```
arr.repeat([2, 3], axis=0)
```

Out[63]:

```
array([[ 0.3189704 , -0.66895566],
       [ 0.3189704 , -0.66895566],
       [ 0.89103869,  0.71390849],
       [ 0.89103869,  0.71390849],
       [ 0.89103869,  0.71390849]])
```

In [64]:

```
arr.repeat([2, 3], axis=1)
```

Out[64]:

```
array([[ 0.3189704 ,  0.3189704 , -0.66895566, -0.66895566, -0.66895566],
       [ 0.89103869,  0.89103869,  0.71390849,  0.71390849,  0.71390849]])
```

tile ise, bir dizinin kopyalarını bir eksen boyunca istiflemek için bir kısa yoldur. Görsel olarak bunu "fayans döşemek" gibi düşünübilirsiniz.

In [65]:

```
arr
```

Out[65]:

```
array([[ 0.3189704 , -0.66895566],
       [ 0.89103869,  0.71390849]])
```

In [66]:

np.tile(arr, 2)

Out[66]:

```
array([[ 0.3189704 , -0.66895566,  0.3189704 , -0.66895566],
       [ 0.89103869,  0.71390849,  0.89103869,  0.71390849]])
```

İkinci argüman tiles sayısıdır; skaler ile döşeme, sütun sütun yerine satır satır yapılır. tile a yönelik ikinci argüman, "tile in" düzenini gösteren bir demet olabilir.

In [67]:

arr

Out[67]:

```
array([[ 0.3189704 , -0.66895566],
       [ 0.89103869,  0.71390849]])
```

In [68]:

np.tile(arr, (2, 1))

Out[68]:

```
array([[ 0.3189704 , -0.66895566],
       [ 0.89103869,  0.71390849],
       [ 0.3189704 , -0.66895566],
       [ 0.89103869,  0.71390849]])
```

In [69]:

np.tile(arr, (3, 2))

Out[69]:

```
array([[ 0.3189704 , -0.66895566,  0.3189704 , -0.66895566],
       [ 0.89103869,  0.71390849,  0.89103869,  0.71390849],
       [ 0.3189704 , -0.66895566,  0.3189704 , -0.66895566],
       [ 0.89103869,  0.71390849,  0.89103869,  0.71390849],
       [ 0.3189704 , -0.66895566,  0.3189704 , -0.66895566],
       [ 0.89103869,  0.71390849,  0.89103869,  0.71390849]])
```

Fancy Indexing Equivalents: al ve koy

In [70]:

arr = np.arange(10) * 100

In [71]:

inds=[7,1,2,6]

In [72]:

```
arr[inds]
```

Out[72]:

```
array([700, 100, 200, 600])
```

Yalnızca tek bir eksende seçim yapmanın özel durumunda faydalı olan alternatif ndarray yöntemleri vardır.

In [73]:

```
arr.take(inds)
```

Out[73]:

```
array([700, 100, 200, 600])
```

In [74]:

```
arr.put(inds, 42)
```

In [75]:

```
arr
```

Out[75]:

```
array([ 0, 42, 42, 300, 400, 500, 42, 42, 800, 900])
```

In [76]:

```
arr.put(inds, [40, 41, 42, 43])
```

In [77]:

```
arr
```

Out[77]:

```
array([ 0, 41, 42, 300, 400, 500, 43, 40, 800, 900])
```

In [78]:

```
inds=[2,0,2,1]
```

In [79]:

```
arr = np.random.randn(2, 4)
```

In [80]:

```
arr
```

Out[80]:

```
array([[ 0.75440666,  0.76811305,  1.58588341,  1.09235958],
       [-0.63699664,  0.66305023,  0.73496467, -2.00202678]])
```

In [81]:

```
arr.take(inds, axis=1)
```

Out[81]:

```
array([[ 1.58588341,  0.75440666,  1.58588341,  0.76811305],
       [ 0.73496467, -0.63699664,  0.73496467,  0.66305023]])
```

put bir eksen bağımsız değişkenini kabul etmez, bunun yerine dizinin düzleştirilmiş (tek boyutlu, C sırası) versiyonunu indeksler. Bu nedenle, diğer eksenlerde bir dizin dizisi kullanarak öğeleri ayarlamaman gerekiyor. Genellikle en kolay ve güvenilir indekslemeyi kullanmak.

A.3 Broadcasting

Yayın, farklı şekillerdeki diziler arasında aritmetiğin nasıl çalıştığını açıklıyor. Güçlü bir özellik olabilir, ancak deneyimli kullanıcılar için bile karma karışıklığına neden olabilir. En basit yayın örneği, bir skaler değeri birleştirirken bir dizi ile ortaya çıkar.

In [2]:

```
import numpy as np
arr = np.arange(5)
```

```
arr
```

In [3]:

```
arr * 4
```

Out[3]:

```
array([ 0,  4,  8, 12, 16])
```

Burada 4 skaler değerinin çarpma işlemindeki diğer tüm öğelere yayındır. 1 söylüyoruz.

Örneğin, bir dizinin her bir sütununu sütun ortalamalarını çıkararak küçültübiliriz.

In [5]:

```
arr = np.random.randn(4, 3)
```

In [6]:

```
arr.mean(0)
```

Out[6]:

```
array([-0.10262695,  0.66132823, -0.59804343])
```

In [7]:

```
demeaned = arr - arr.mean(0)
```

In [8]:

```
demeaned
```

Out[8]:

```
array([[-1.01876925,  0.62888733, -1.07511455],  
      [-0.10639473,  0.30507536, -0.19811288],  
      [ 1.55251509, -0.37169759,  0.59968121],  
      [-0.42735111, -0.56226511,  0.67354621]])
```

In [9]:

```
demeaned.mean(0)
```

Out[9]:

```
array([ 2.77555756e-17, -2.77555756e-17,  2.77555756e-17])
```

Bu işlemin bir açıklaması için Şekil A-4'e bakın. Satırları bir yayın işlemi olarak küçültmek biraz daha özen gerektirir. Neyse ki, bir dizinin herhangi bir boyutunda potansiyel olarak daha düşük boyutlu değerleri yayınlamak (iki boyutlu bir dizinin her bir sütunundan satır ortalamasını çıkarmak gibi).

Broadcasting Kuralı

Her bir son boyut için (yani, sondan başlayarak) eksen uzunlukları eşleşiyorsa veya uzunluklardan biri 1 ise, yayın için iki dizi uyumludur. Eksik veya uzunluk 1 boyutları üzerinden yayın gerçekleştirilir.

In [10]:

```
from IPython.display import Image
Image(filename='img/Picture90.png')
```

Out[10]:

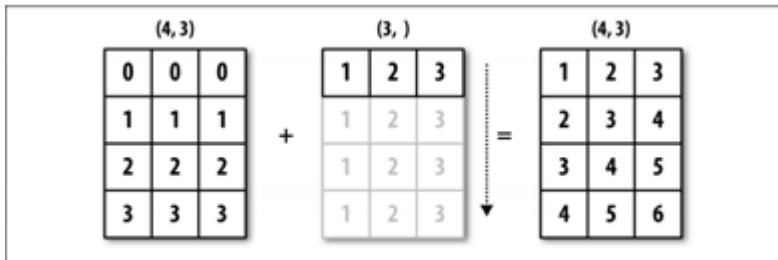


Figure A-4. Broadcasting over axis 0 with a 1D array

Deneyimli bir NumPy kullanıcısı olarak bile, yayın kuralı hakkında düşünürken sık sık kendimi duraklatmak ve bir diyagram çizmek zorunda buluyorum. Son örneği düşünün ve bunun yerine her satırdan ortalama değeri çıkarmak istediyimizi varsayıyalım. `Arr.mean(0)` 3 uzunluğa sahip olduğundan, eksen 0 boyunca yayın için uyumludur çünkü son dizi boyut 3'tür ve bu nedenle eşleşir. Kural lara göre, eksen 1'den çıkarmak için (yani, her satırdan satır ortalamasını çıkarmak), daha küçük dizinin şekli olmalıdır.

In [11]:

arr

Out[11]:

```
array([[-1.12139619e+00,  1.29021556e+00, -1.67315797e+00],
       [-2.09021671e-01,  9.66403586e-01, -7.96156307e-01],
       [ 1.44988814e+00,  2.89630643e-01,  1.63778813e-03],
       [-5.29978059e-01,  9.90631235e-02,  7.55027864e-02]])
```

In [12]:

row_means = arr.mean(1)

In [14]:

row_means.shape

Out[14]:

(4,)

In [15]:

row_means.reshape((4, 1))

Out[15]:

```
array([[-0.5014462 ],
       [-0.0129248 ],
       [ 0.58038552],
       [-0.11847072]])
```

In [16]:

```
demeaned = arr - row_means.reshape((4, 1))
```

In [17]:

```
demeaned.mean(1)
```

Out[17]:

```
array([-1.48029737e-16,  0.00000000e+00,  0.00000000e+00, -9.25185854e-18])
```

In [18]:

```
Image(filename='img/Picture91.png')
```

Out[18]:

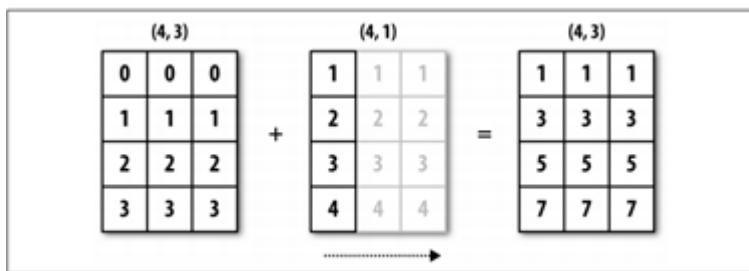


Figure A-5. Broadcasting over axis 1 of a 2D array

Başka bir örnek için Şekil A-6'ya bakın, bu kez eksen 0 boyunca üç boyutlu bir diziye iki boyutlu bir dizi ekliyoruz.

In [19]:

```
Image(filename='img/Picture92.png')
```

Out[19]:

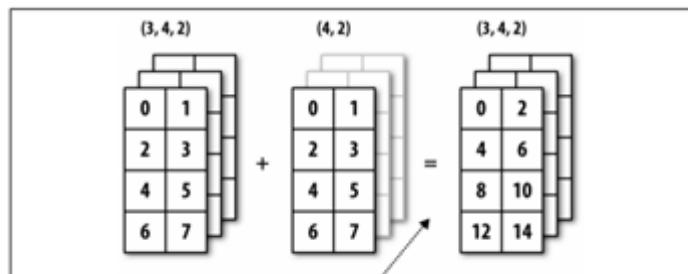


Figure A-6. Broadcasting over axis 0 of a 3D array

Diğer Eksenler Üzerinden Broadcasting

Daha yüksek boyutlu dizilerle yayın yapmak daha da kafa karıştırıcı görünebilir, ancak bu gerçekten kurallara uymaktır. Aksi takdirde, şuna benzer bir hata alırsınız.

In [20]:

```
arr = arr.mean(1)
```

```
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-20-8b8ada26fac0> in <module>
----> 1 arr = arr.mean(1)
```

ValueError: operands could not be broadcast together with shapes (4,3) (4,)

Eksen 0 dışındaki eksenler boyunca daha düşük boyutlu bir dizi ile aritmetik bir işlem gerçekleştirmek oldukça yaygındır. Yayın kuralına göre, "yayın boy utları" küçük dizide 1 olmalıdır. Burada gösterilen satır küçültme örneğinde bu, sırayı yeniden şekillendirmek, şekil olmak anlamına gelir (4, 1) (4,) ye rine.

In [21]:

```
arr = arr.mean(1).reshape(4, 1)
```

Out[21]:

```
array([[-0.61994999,  1.79166176, -1.17171177],
       [-0.19609687,  0.97932838, -0.78323151],
       [ 0.86950262, -0.29075488, -0.57874774],
       [-0.41150734,  0.21753384,  0.1939735 ]])
```

Üç boyutlu durumda, üç boyuttan herhangi biri üzerinden yayın yapmak yalnızca veriyi şekil uyumlu olacak şekilde yeniden şekillendirme meselesidir. Şekil 1 A-7, üç boyutlu bir dizinin her bir eksenin üzerinde yaynlamak için gerekken şekilleri güzel bir şekilde görselleştirir.

Bu nedenle yaygın bir sorun, özellikle yaynlama amaçları için uzunluk 1 olan yeni bir eksen eklemeye ihtiyaç duyulmasıdır. Yeniden şekillendirmeyi kullanmak bir seçenekdir, ancak bir eksen eklemek, yeni şekli gösteren bir demet oluşturmayı gerektirir. Bu genellikle sıkıcı bir egzersiz olabilir. Bu nedenle NumPy dizileri, indeksleme yoluyla yeni eksenler eklemek için özel bir sözdizimi sunar. Yeni eksenin eklemek için özel np.newaxis niteliğini "tam" dillerimlerle birlikte kullanırız.

In [22]:

```
arr = np.zeros((4, 4))
```

In [23]:

```
arr_3d = arr[:, np.newaxis, :]
```

In [24]:

```
arr_3d.shape
```

Out[24]:

(4, 1, 4)

In [25]:

```
arr_1d = np.random.normal(size=3)
```

In [26]:

```
arr_1d[:, np.newaxis]
```

Out[26]:

```
array([[1.02756759],
       [0.77335906],
       [0.46256821]])
```

In [27]:

```
arr_1d[np.newaxis, :]
```

Out[27]:

```
array([[1.02756759, 0.77335906, 0.46256821]])
```

In [28]:

```
Image(filename='img/Picture93.png')
```

Out[28]:

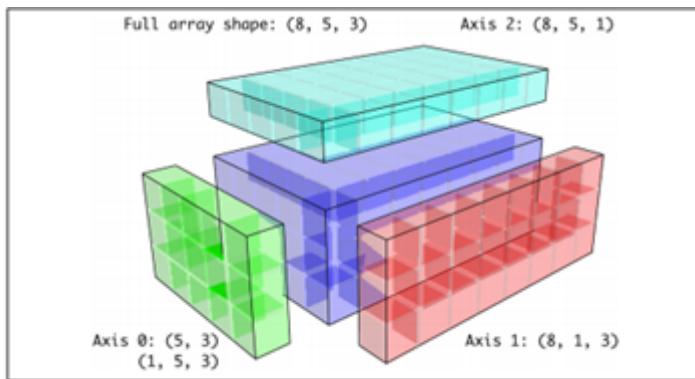


Figure A-7. Compatible 2D array shapes for broadcasting over a 3D array

Bu nedenle, üç boyutlu bir dizimiz olsaydı ve 2. ekseni küçültmek isteseydi k, şunu yazmamız gereklirdi:

In [29]:

```
arr = np.random.randn(3, 4, 5)
```

In [30]:

```
depth_means = arr.mean(2)
```

In [31]:

```
depth_means
```

Out[31]:

```
array([[-0.80026618, -0.24680539, -0.34028323, -1.2243451 ],
       [-0.17673106, -0.23526429, -0.13232257,  0.67924317],
       [-0.27754829, -0.41712387,  0.04067293, -0.11665971]])
```

In [32]:

```
depth_means.shape
```

Out[32]:

(3, 4)

In [33]:

```
demeaned = arr - depth_means[:, :, np.newaxis]
```

In [34]:

```
demeaned.mean(2)
```

Out[34]:

```
array([[ -4.44089210e-17,   2.22044605e-17,   0.00000000e+00,
         2.22044605e-17],
       [-2.22044605e-17,  -8.88178420e-17,   0.00000000e+00,
      -4.44089210e-17],
       [ 0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
      -1.11022302e-17]])
```

Performanstan ödün vermeden bir ekseni küçültmenin genellemenin bir yolu vardır.

In [36]:

```
def demean_axis(arr, axis=0):
    means = arr.mean(axis)
    # Bu, [:::, np.newaxis] gibi seyleri N boyuta geneller
    indexer = [slice(None)] * arr.ndim
    indexer[axis] = np.newaxis
    return arr - means[indexer]
```

Dizi Değerlerini Broadcasting e Göre Ayarlama

Aritmetik işlemleri yöneten aynı yayın kuralı, dizi indeksleme yoluyla değerlerin ayarlanması için de geçerlidir. Basit bir durumda, aşağıdaki gibi şeyler yapabiliriz:

In [37]:

```
arr = np.zeros((4, 3))
```

In [38]:

```
arr[:] = 5
```

In [39]:

```
arr
```

Out[39]:

```
array([[5., 5., 5.],
       [5., 5., 5.],
       [5., 5., 5.],
       [5., 5., 5.]])
```

Bununla birlikte, dizinin sütunlarına yerleştirmek istediğimiz tek boyutlu bir değerler dizimiz varsa, şekil uyumlu olduğu sürece bunu yapabiliriz:

In [40]:

```
col = np.array([1.28, -0.42, 0.44, 1.6])
```

In [41]:

```
arr[:, :] = col[:, np.newaxis]
```

In [42]:

```
arr
```

Out[42]:

```
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 [43]:

```
arr[:2] = [[-1.37], [0.509]]
```

In [44]:

```
arr
```

Out[44]:

```
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 ufunc Kullanımı

Birçok NumPy kullanıcısı yalnızca evrensel işlevler tarafından sağlanan hızlı öğe bazlı işlemlerden yararlanırken, bazen döngüler olmadan daha kısa kod yazmanıza yardımcı olabilecek bir dizi ek özellik vardır.

ufunc Instance Methods

NumPy'nin ikili ufunc'larının her biri, belirli türdeki özel vektörleştirilmış işlemleri gerçekleştirmek için özel yöntemlere sahiptir. Bunlar Tablo A-2'de özetlenmiştir, ancak nasıl çalışıklarını göstermek için birkaç somut örnek vereceğim.

reduce, tek bir diziyi alır ve bir dizi ikili işlem gerçekleştirerek, isteğe bağlı olarak bir eksen boyunca değerlerini toplar. Örneğin, bir dizideki öğeleri toplamanın alternatif bir yolu, np.add.reduce kullanmaktadır.

In [45]:

```
arr = np.arange(10)
```

In [46]:

```
np.add.reduce(arr)
```

Out[46]:

```
45
```

In [47]:

```
arr.sum()
```

Out[47]:

```
45
```

Başlangıç değeri (toplama için 0) ufunc'a bağlıdır. Bir eksen geçilirse, küçük ültme o eksen boyunca gerçekleştirilir. Bu, belirli türdeki soruları kısa ve öz bir şekilde yanıtlanmanıza olanak tanır. Daha az önemsiz bir örnek olarak, bir dizinin her satırındaki değerlerin sıralı olup olmadığını kontrol etmek için np.logical_and kullanabiliriz.

In [48]:

```
np.random.seed(12346) # tekrarlanabilirlik için
```

In [49]:

```
arr = np.random.randn(5, 5)
```

In [50]:

```
arr[::-2].sort(1) # birkaç satırı sırala
```

In [51]:

```
arr[:, :-1] < arr[:, 1:]
```

Out[51]:

```
array([[ True,  True,  True,  True],
       [False,  True, False, False],
       [ True,  True,  True,  True],
       [ True, False,  True,  True],
       [ True,  True,  True,  True]])
```

In [52]:

```
np.logical_and.reduce(arr[:, :-1] < arr[:, 1:], axis=1)
```

Out[52]:

```
array([ True, False,  True, False,  True])
```

Logical_and.reduce ögesinin all yöntemine eşdeğer olduğunu unutmayın.

accumulate , cumsum toplamla ilişkili olduğu gibi azaltmakla ilgiliidir. Ara "accumulated" değerlerle aynı boyutta bir dizi oluşturur.

In [53]:

```
arr = np.arange(15).reshape((3, 5))
```

In [54]:

```
np.add.accumulate(arr, axis=1)
```

Out[54]:

```
array([[ 0,  1,  3,  6, 10],
       [ 5, 11, 18, 26, 35],
       [10, 21, 33, 46, 60]], dtype=int32)
```

output, iki dizi arasında ikili bir çapraz çarpım gerçekleştirir.

In [55]:

```
arr = np.arange(3).repeat([1, 2, 2])
```

In [56]:

```
arr
```

Out[56]:

```
array([0, 1, 1, 2, 2])
```

In [58]:

```
np.multiply.outer(arr, np.arange(5))
```

Out[58]:

```
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]])
```

output çıktısı, girdilerin boyutlarının toplamı olan bir boyuta sahip olacak tır.

In [59]:

```
x, y = np.random.randn(3, 4), np.random.randn(5)
```

In [60]:

```
result = np.subtract.outer(x, y)
```

In [61]:

```
result.shape
```

Out[61]:

```
(3, 4, 5)
```

Son yöntem olan indirgeme yöntemi, özünde dizi dilimlerinin bir araya toplanacağı bir dizi graplama işlemi olan "yerel küçültme" gerçekleştirir. Değerlerin nasıl bölüneceğini ve toplanacağını gösteren bir dizi "bin edges" kabul eder.

In [62]:

```
arr = np.arange(10)
```

In [63]:

```
np.add.reduceat(arr, [0, 5, 8])
```

Out[63]:

```
array([10, 18, 17], dtype=int32)
```

Sonuçlar arr [0: 5], arr [5: 8] ve arr [8:] üzerinde gerçekleştirilen indirimlerdir (burada toplamlar). Diğer yöntemlerde olduğu gibi, bir eksen bağımsız değişkeni iletebilirsiniz.

In [65]:

```
arr = np.multiply.outer(np.arange(4), np.arange(5))
```

In [66]:

```
arr
```

Out[66]:

```
array([[ 0,  0,  0,  0,  0],
       [ 0,  1,  2,  3,  4],
       [ 0,  2,  4,  6,  8],
       [ 0,  3,  6,  9, 12]])
```

In [67]:

```
np.add.reduceat(arr, [0, 2, 4], axis=1)
```

Out[67]:

```
array([[ 0,  0],
       [ 1,  5,  4],
       [ 2, 10,  8],
       [ 3, 15, 12]], dtype=int32)
```

In [68]:

```
Image(filename='img/Picture94.png')
```

Out[68]:

Table A-2. ufunc methods

Method	Description
reduce(x)	Aggregate values by successive applications of the operation
accumulate(x)	Aggregate values, preserving all partial aggregates
reduceat(x, bins)	"Local" reduce or "group by"; reduce contiguous slices of data to produce aggregated array
outer(x, y)	Apply operation to all pairs of elements in x and y; the resulting array has shape x.shape + y.shape

Python'da Yeni Ufunclar Yazma

Kendi NumPy ufunc'larınızı oluşturmak için bir dizi tesis vardır. En genel olanı NumPy C API'sini kullanmaktadır, ancak bu, bu kitabın kapsamı dışındadır. Bu bölümde, saf Python ufunclarına bakacağız.

`numpy.frompyfunc`, girdi ve çıktıların sayısı için bir belirtimle birlikte bir Python işlevini kabul eder. Örneğin, öğe bazında ekleyen basit bir işlev şu şekilde belirtilir:

In [69]:

```
def add_elements(x, y):
    return x + y
```

In [70]:

```
addThem = np.frompyfunc(add_elements, 2, 1)
```

In [71]:

```
addThem(np.arange(8), np.arange(8))
```

Out[71]:

```
array([0, 2, 4, 6, 8, 10, 12, 14], dtype=object)
```

Frompyfunc kullanılarak oluşturulan işlevler, her zaman uygun olmayan Python nesnelerinin dizilerini döndürür. Neyse ki, çıktı türünü belirlemenize izin veren alternatif (ancak biraz daha az özellikli) bir işlev olan `numpy.vectorize` vardır.

In [72]:

```
addThem = np.vectorize(add_elements, otypes=[np.float64])
```

In [73]:

```
addThem(np.arange(8), np.arange(8))
```

Out[73]:

```
array([ 0.,  2.,  4.,  6.,  8., 10., 12., 14.])
```

Bu işlevler ufunc benzeri işlevler yaratmanın bir yolunu sağlar, ancak çok yavaşırlar çünkü her bir öğeyi hesaplamak için bir Python işlev çağrıları gerektirirler, bu NumPy'nin C tabanlı ufunc döngülerinden çok daha yavaştır.

In [74]:

```
arr = np.random.randn(10000)
```

In [75]:

```
%timeit add_them(arr, arr)
```

2.1 ms ± 273 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

In [76]:

```
%timeit np.add(arr, arr)
```

4.17 µs ± 361 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

A.5 Yapılandırılmış ve Kayıt Dizileri

Şimdiye kadar ndarray'in homojen bir veri kabı olduğunu fark etmiş olabilirsiniz; yani, her bir ögenin dtype tarafından belirlenen aynı sayıda baytı kapladığı bir bellek bloğunu temsil eder. Üzeyde bu, heterojen veya tablo benzeri verileri temsil etmenize izin vermiyor gibi görünüyor. Yapılandırılmış bir dizi, her bir ögenin C'deki bir yapıyı (dolayısıyla "yapılardırılmış" adı) veya birden çok adlandırılmış alana sahip bir SQL tablosundaki bir satırı temsil ettiği düşünülebilecek bir ndarray'dir.

In [2]:

```
import numpy as np
dtype = [('x', np.float64), ('y', np.int32)]
```

In [3]:

```
sarr = np.array([(1.5, 6), (np.pi, -2)], dtype=dtype)
```

In [4]:

```
sarr
```

Out[4]:

```
array([(1.5, 6), (3.14159265, -2)],
      dtype=[('x', '<f8'), ('y', '<i4')])
```

Yapılardırılmış bir dtype belirlemenin birkaç yolu vardır (çevrimiçi NumPy bülgesine bakın). Tipik bir yol, (alan_adı, alan_verisi_türü) ile demetler listesidir.

Şimdi, dizinin elemanları, elemanlarına bir sözlük gibi erişilebilen tuple benzeri nesnelerdir.

In [5]:

```
sarr[0]
```

Out[5]:

```
(1.5, 6)
```

In [6]:

```
sarr[0]['y']
```

Out[6]:

```
6
```

Alan adları, `dtype.names` özniteliğinde saklanır. Yapılandırılmış dizideki bir alana eriştiğinizde, veriler üzerinde adım adım bir görünüm döndürülür, böylece hiçbir şey kopyalanmaz.

In [7]:

```
sarr['x']
```

Out[7]:

```
array([1.5, 3.14159265])
```

İç içe geçmiş tipler ve Çok Boyutlu Alanlar

Yapılandırılmış bir `dtype` belirtirken, ek olarak bir şekil (`int` veya `tuple` olarak) iletebilirsiniz.

In [8]:

```
dtype = [('x', np.int64, 3), ('y', np.int32)]
```

In [9]:

```
arr = np.zeros(4, dtype=dtype)
```

In [10]:

```
arr
```

Out[10]:

```
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 diziyi ifade eder.

In [11]:

```
arr[0]['x']
```

Out[11]:

```
array([0, 0, 0], dtype=int64)
```

Uygun bir şekilde, arr ['x'] 'e erişmek, önceki örneklerde olduğu gibi tek boyutlu bir dizi yerine iki boyutlu bir dizi döndürür.

In [12]:

```
arr['x']
```

Out[12]:

```
array([[0, 0, 0],  
       [0, 0, 0],  
       [0, 0, 0],  
       [0, 0, 0]], dtype=int64)
```

Bu, daha karmaşık, iç içe geçmiş yapıları bir dizide tek bir bellek bloğu olarak ifade etmenizi sağlar. Daha karmaşık yapılar oluşturmak için dtype'ları da iç içe yerlestirebilirsiniz. İşte bir örnek:

In [13]:

```
dtype = [('x', [(('a', 'f8'), ('b', 'f4'))]), ('y', np.int32)]
```

In [14]:

```
data = np.array([(1, 2), 5], ((3, 4), 6)), dtype=dtype)
```

In [15]:

```
data['x']
```

Out[15]:

```
array([(1., 2.), (3., 4.)], dtype=[('a', '<f8'), ('b', '<f4')])
```

In [16]:

```
data['y']
```

Out[16]:

```
array([5, 6])
```

In [17]:

```
data['x']['a']
```

Out[17]:

```
array([1., 3.])
```

Pandas DataFrame, hiyerarşik indekslemeye benzer olsa da, bu özelliği doğrudan desteklemez.

Neden Yapılandırılmış Diziler Kullanılır?

Örneğin bir panda DataFrame ile karşılaştırıldığında, NumPy yapısal dizileri nispeten düşük seviyeli bir araçtır. Bir bellek bloğunu, rastgele karmaşık içe geçmiş sütunlara sahip bir tablo yapısı olarak yorumlamak için bir yol sağlarlar. Dizideki her öğe bellekte sabit sayıda bayt olarak temsil edildiğiinden, yapılandırılmış diziler diske ve diske (bellek haritaları dahil) veri yazmak, ağ üzerinden taşımak ve diğer bu tür kullanımlar için çok hızlı ve verimli bir yol sağlar. Yapılandırılmış diziler için başka bir yaygın kullanım olarak, veri dosyalarını sabit uzunlukta kayıt bayt akışları olarak yazmak, verileri endüstrideki eski sistemlerde yaygın olarak bulunan C ve C++ kodunda serileştirmenin yaygın bir yoludur. Dosyanın biçimi bilindiği sürece (her kaydın boyutu ve her ögenin sırası, bayt boyutu ve veri türü), veriler np.fromfile ile belleğe okunabilir. Bunun gibi özel kullanımlar bu kitabın kapsamı dışındadır, ancak bu tür şeylerin mümkün olduğunu bilmeye değer.

A.6 Sıralama Hakkında Daha Fazla Bilgi

Python'un yerleşik listesi gibi, ndarray sıralama örneği yöntemi de yerinde sıralamadır, yani dizi içeriği yeni bir dizi oluşturmadan yeniden düzenlenir.

In [18]:

```
arr = np.random.randn(6)
```

In [19]:

```
arr.sort()
```

In [20]:

```
arr
```

Out[20]:

```
array([-0.6592693 , -0.54913708,  0.11702617,  0.12482565,  0.52495146,
       0.8050308 ])
```

Dizileri yerinde sıralarken, dizi farklı bir ndarray üzerinde bir görünümse, orijinal dizinin değiştirileceğini unutmayın.

In [21]:

```
arr = np.random.randn(3, 5)
```

In [22]:

```
arr
```

Out[22]:

```
array([[ 1.70464274, -0.64193199, -0.30126109, -0.18956835,  2.7474241
], [-1.34648997,  0.88614509,  0.75290673,  2.81032574, -0.8521913
9],
[ 0.01253846, -0.78653245, -1.45624071, -0.21058634,  0.1261062
3]])
```

In [23]:

```
arr[:, 0].sort() # ilk sütun değerlerini yerinde sıralama
```

In [24]:

```
arr
```

Out[24]:

```
array([[-1.34648997, -0.64193199, -0.30126109, -0.18956835,  2.7474241
], [ 0.01253846,  0.88614509,  0.75290673,  2.81032574, -0.8521913
9],
[ 1.70464274, -0.78653245, -1.45624071, -0.21058634,  0.1261062
3]])
```

Öte yandan numpy.sort, bir dizinin yeni, sıralanmış bir kopyasını oluşturur. Aksi takdirde, ndarray.sort ile aynı bağımsız değişkenleri (tür gibi) kabul eder.

In [25]:

```
arr = np.random.randn(5)
```

In [26]:

```
arr
```

Out[26]:

```
array([-0.95873777, -1.534479 ,  1.12872424, -1.51134213, -1.0613957
])
```

In [27]:

```
np.sort(arr)
```

Out[27]:

```
array([-1.534479 , -1.51134213, -1.0613957 , -0.95873777,  1.1287242
4])
```

In [28]:

```
arr
```

Out[28]:

```
array([-0.95873777, -1.534479 ,  1.12872424, -1.51134213, -1.0613957])
```

Bu sıralama yöntemlerinin tümü, veri bölümlerini geçirilen eksen boyunca bağımsız olarak sıralamak için bir eksen bağımsız değişkeni alır.

In [29]:

```
arr = np.random.randn(3, 5)
```

In [30]:

```
arr
```

Out[30]:

```
array([[ 0.02850531,  0.95235365,  1.89928808,  1.30244507, -0.58103664],
       [ 1.15788742,  0.9562647 ,  1.15018941,  1.0998472 , -0.07668368],
       [ 0.07843286, -0.86457295,  1.34462476,  0.56214941,  0.38815793]])
```

In [31]:

```
arr.sort(axis=1)
```

In [32]:

```
arr
```

Out[32]:

```
array([[-0.58103664,  0.02850531,  0.95235365,  1.30244507,  1.89928808],
       [-0.07668368,  0.9562647 ,  1.0998472 ,  1.15018941,  1.15788742],
       [-0.86457295,  0.07843286,  0.38815793,  0.56214941,  1.34462476]])
```

Sıralama yöntemlerinden hiçbirinin azalan düzende sıralama seçeneğine sahip olmadığını fark edebilirsiniz. Bu pratikte bir sorundur, çünkü dizi dilimleme görünümleri üretir, dolayısıyla bir kopya üretmez veya herhangi bir hesaplama işi gerektirmez. Çoğu Python kullanıcısı, bir liste değerleri için `[::-1]` değerlerinin bir listeyi ters sırada döndürdüğü "hileye" aşınadır. Aynısındarrays için de geçerlidir.

In [33]:

arr[:, ::-1]

Out[33]:

```
array([[ 1.89928808,  1.30244507,  0.95235365,  0.02850531, -0.5810366
4],
       [ 1.15788742,  1.15018941,  1.0998472 ,  0.9562647 , -0.0766836
8],
       [ 1.34462476,  0.56214941,  0.38815793,  0.07843286, -0.8645729
5]])
```

Dolaylı Türler: argsort ve lexsort

Veri analizinde, veri kümelerini bir veya daha fazla anahtarla yeniden sıralamanız gerekebilir. Örneğin, bazı öğrencilerle ilgili bir veri tablosunun soyadına ve ardından ada göre sıralanması gerekebilir. Bu, dolaylı bir sıralama örneğidir. Bir anahtar veya anahtarlar (bir değerler dizisi veya birden çok değer dizisi) verildiğinde, verileri sıralı sırayla nasıl yeniden sıralaya çağınızı söyleyen bir tamsayı dizini dizisi elde etmek istersiniz (bunlara ortak olarak dizin oluşturucular olarak atıfta bulunuyorum). Bunun için iki yöntem argsort ve numpy.lexsort'tur. Örnek olarak:

In [34]:

values = np.array([5, 0, 1, 3, 2])

In [35]:

indexer = values.argsort()

In [36]:

indexer

Out[36]:

array([1, 2, 4, 3, 0], dtype=int64)

In [37]:

values[indexer]

Out[37]:

array([0, 1, 2, 3, 5])

Daha karmaşık bir örnek olarak, bu kod iki boyutlu bir diziyi ilk satırına göre yeniden sıralar.

In [38]:

arr = np.random.randn(3, 5)

In [39]:

```
arr[0] = values
```

In [40]:

```
arr
```

Out[40]:

```
array([[ 5.          ,  0.          ,  1.          ,  3.          ,  2.
], [-0.10161081,  1.66925465, -0.03679411,  0.47800524,  0.3545059
8], [-0.23366614,  2.02017718, -0.36370545, -1.07748509, -0.8933173
]])
```

In [41]:

```
arr[:, arr[0].argsort()]
```

Out[41]:

```
array([[ 0.          ,  1.          ,  2.          ,  3.          ,  5.
], [ 1.66925465, -0.03679411,  0.35450598,  0.47800524, -0.1016108
1], [ 2.02017718, -0.36370545, -0.8933173 , -1.07748509, -0.2336661
4]])
```

lexsort, argsort'a benzer, ancak birden çok anahtar dizisinde dolaylı bir sözlükbilimsel sıralama gerçekleştirir. Ad ve soyadlarına göre tanımlanan bazı verileri sıralamak istediğimizi varsayıyalım.

In [43]:

```
first_name = np.array(['Bob', 'Jane', 'Steve', 'Bill', 'Barbara'])
```

In [44]:

```
last_name = np.array(['Jones', 'Arnold', 'Arnold', 'Jones', 'Walters'])
```

In [45]:

```
sorter = np.lexsort((first_name, last_name))
```

In [46]:

```
sorter
```

Out[46]:

```
array([1, 2, 3, 0, 4], dtype=int64)
```

In [47]:

```
zip(last_name[sorter], first_name[sorter])
```

Out[47]:

```
<zip at 0x18ab6250040>
```

lexsort, onu ilk kullandığınızda biraz kafa karıştırıcı olabilir, çünkü verileri sıralamak için kullanılan anahtarların sırası geçirilen son diziyle başlar. Burada last_name addan önce kullanılmıştır.

Series ve DataFrame'in sort_values yöntemi gibi pandas yöntemleri bu işlevle rin varyantları ile uygulanır (ayrıca eksik değerleri hesaba katın).

Alternatif Sıralama Algoritmaları

Kararlı bir sıralama algoritması, eşit öğelerin göreceli konumunu korur. Bu, özellikle göreli sıralamanın anlamlı olduğu dolaylı türlerde önemli olabilir.

In [48]:

```
values = np.array(['2:first', '2:second', '1:first', '1:second',
                  '1:third'])
```

In [49]:

```
key = np.array([2, 2, 1, 1, 1])
```

In [50]:

```
indexer = key.argsort(kind='mergesort')
```

In [51]:

```
indexer
```

Out[51]:

```
array([2, 3, 4, 0, 1], dtype=int64)
```

In [52]:

```
values.take(indexer)
```

Out[52]:

```
array(['1:first', '1:second', '1:third', '2:first', '2:second'],
      dtype='<U8')
```

Mevcut tek kararlı sıralama, garantili $O(n \log n)$ performansı (karşılıklık

buffları için) olan, ancak performansı varsayılan hızlı sıralama yönteminde n ortalama olarak daha kötü olan birleştirme sıralamasıdır. Mevcut yöntemlerin ve bunların göreceli performanslarının (ve performans garantilerinin) bir özeti için Tablo A-3'e bakın. Bu, çoğu kullanıcının düşünmek zorunda kalacağı bir şey değildir, ancak orada olduğunu bilmek yararlıdır.

In [54]:

```
from IPython.display import Image
Image(filename='img/Picture95.png')
```

Out[54]:

Table A-3. Array sorting methods

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

Sıralama hedeflerinden biri, bir dizideki en büyük veya en küçük öğeleri belirlemek olabilir. NumPy, bir diziyi k. En küçük öğe etrafında böülümlmek için optimize edilmiş yöntemler, `numpy.partition` ve `np.argpartition`'a sahiptir.

In [55]:

```
np.random.seed(12345)
```

In [56]:

```
arr = np.random.randn(20)
```

In [57]:

```
arr
```

Out[57]:

```
array([-0.20470766,  0.47894334, -0.51943872, -0.5557303 ,  1.9657805
7,
       1.39340583,  0.09290788,  0.28174615,  0.76902257,  1.2464347
4,
       1.00718936, -1.29622111,  0.27499163,  0.22891288,  1.3529168
4,
       0.88642934, -2.00163731, -0.37184254,  1.66902531, -0.4385697
4])
```

In [58]:

```
np.partition(arr, 3)
```

Out[58]:

```
array([-2.00163731, -1.29622111, -0.5557303 , -0.51943872, -0.3718425
4,
       -0.43856974, -0.20470766,  0.28174615,  0.76902257,  0.4789433
4,
       1.00718936,  0.09290788,  0.27499163,  0.22891288,  1.3529168
4,
       0.88642934,  1.39340583,  1.96578057,  1.66902531,  1.2464347
4])
```

Bölümü (`arr, 3`) çağırıldıktan sonra, sonuçtaki ilk üç öğe belirli bir sıraya sahip olmayan en küçük üç değerdir. `numpy.argpartition`, `numpy.argsort` sıralamasın a benzer şekilde, verileri eşdeğer sıraya göre yeniden düzenleyen dizinleri döndürür.

In [59]:

```
indices = np.argpartition(arr, 3)
```

In [60]:

```
indices
```

Out[60]:

```
array([16, 11,  3,  2, 17, 19,  0,  7,  8,  1, 10,  6, 12, 13, 14, 15,
5,
       4, 18,  9], dtype=int64)
```

In [61]:

```
arr.take(indices)
```

Out[61]:

```
array([-2.00163731, -1.29622111, -0.5557303 , -0.51943872, -0.3718425
4,
       -0.43856974, -0.20470766,  0.28174615,  0.76902257,  0.4789433
4,
       1.00718936,  0.09290788,  0.27499163,  0.22891288,  1.3529168
4,
       0.88642934,  1.39340583,  1.96578057,  1.66902531,  1.2464347
4])
```

numpy.searchsorted: Sıralanmış Bir Dizide Öğeleri Bulma

`searchsorted`, sıralanmış bir dizide ikili arama gerçekleştiren ve sıralılığı korumak için değerin girilmesi gereken dizideki konumu döndüren bir dizi yöntemidir.

In [62]:

```
arr = np.array([0, 1, 7, 12, 15])
```

In [63]:

```
arr.searchsorted(9)
```

Out[63]:

```
3
```

Ayrıca bir dizi dizini geri almak için bir dizi değer iletebilirsiniz.

In [64]:

```
arr.searchsorted([0, 8, 11, 16])
```

Out[64]:

```
array([0, 3, 3, 5], dtype=int64)
```

Searchsorted'ın 0 ögesi için 0 döndürüğünü fark etmiş olabilirsiniz. Bunun nedeni, varsayılan davranışın dizini eşit değerlerden oluşan bir grubun sol tarafında döndürmesidir.

In [65]:

```
arr = np.array([0, 0, 0, 1, 1, 1, 1])
```

In [66]:

```
arr.searchsorted([0, 1])
```

Out[66]:

```
array([0, 3], dtype=int64)
```

In [67]:

```
arr.searchsorted([0, 1], side='right')
```

Out[67]:

```
array([3, 7], dtype=int64)
```

Searchsorted'in başka bir uygulaması olarak, 0 ile 10.000 arasında bir değerler dizimiz ve verileri binlemek için kullanmak istediğimiz ayrı bir "bölüm kenarları" dizimiz olduğunu varsayalım.

In [68]:

```
data = np.floor(np.random.uniform(0, 10000, size=50))
```

In [69]:

```
bins = np.array([0, 100, 1000, 5000, 10000])
```

In [70]:

```
data
```

Out[70]:

```
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 veri noktasının hangi aralığa ait olduğu hakkında bir etiket elde etmek için (burada 1, grup [0, 100 anlamına gelir)), arama sıralı kullanabiliriz.

In [71]:

```
labels = bins.searchsorted(data)
```

In [72]:

```
labels
```

Out[72]:

```
array([4, 4, 4, 3, 2, 4, 4, 2, 3, 4, 4, 2, 4, 4, 4, 4, 4, 4, 2, 3, 3, 3,
       3,
       4, 3, 4, 3, 4, 4, 4, 3, 3, 4, 4, 4, 3, 3, 4, 3, 3, 4, 4, 4, 4, 4,
       4,
       4, 3, 3, 4, 4, 3], dtype=int64)
```

Bu, pandas groupby'yle birleştirildiğinde, verileri bölmek için kullanılabilir.

In [74]:

```
import pandas as pd
pd.Series(data).groupby(labels).mean()
```

Out[74]:

```
2    498.000000
3    3064.277778
4    7389.035714
dtype: float64
```

A.7 Numba ile Hızlı NumPy İşlevleri Yazma

Numba, CPU'lar, GPU'lar 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, bir for döngüsü kullanarak `(x - y).mean()` ifadesini hesaplayan saf bir Python işlevini düşünelim:

In [75]:

```
import numpy as np
```

In [76]:

```
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 [78]:

```
x = np.random.randn(10000000)
```

In [79]:

```
y = np.random.randn(10000000)
```

In [80]:

```
%timeit mean_distance(x, y)
```

```
1.44 µs ± 97.8 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

In [81]:

```
%timeit (x - y).mean()
```

```
85.9 ms ± 6.17 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

NumPy sürümü 100 kattan daha hızlıdır. Bu işlevi, numba.jit işlevini kullanarak derlenmiş bir Numba işlevine dönüştürebiliriz.

In [82]:

```
import numba as nb
```

In [83]:

```
numba_mean_distance = nb.jit(mean_distance)
```

Bunu bir dekoratör olarak da yazabilirdik.

In [85]:

```
@nb.jit
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
```

Ortaya çıkan işlev aslında vektörleştirilmiş NumPy sürümünden daha hızlıdır.

In [86]:

```
%timeit numba_mean_distance(x, y)
```

The slowest run took 13.86 times longer than the fastest. This could mean that an intermediate result is being cached.
 $2.11 \mu\text{s} \pm 3.1 \mu\text{s}$ per loop (mean \pm std. dev. of 7 runs, 1 loop each)

Numba, rastgele Python kodunu derleyemez, ancak sayısal algoritmalar yazmak için en yararlı olan önemli bir saf Python alt kümesini destekler. Numba, farklı donanım türlerini, derleme modlarını ve kullanıcı uzantılarını destekleyen derin bir kitaplıktır. Ayrıca, döngüler açık bir şekilde belirtmeden NumPy Python API'sinin önemli bir alt kümesini derleyebilir. Numba, nasıl derleneceğini bilmeli işlevler için CPython API çağrılarını değiştirirken makine koduna derlenebilir. Numba'nın jit işlevi, izin verilen kodu, herhangi bir Python C API çağrısı olmadan LLVM'de derlenebilen Python koduyla sınırlayan nopython = True seçeneğine sahiptir. jit (nopython = True) daha kısa bir takma adı sahiptir numba.njit.

Önceki örnekte şunları yazabilirdik:

In [87]:

```
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'nin bir Python uygulamasını ele alalım.

In [110]:

```
from numba import vectorize

@vectorize
def nb_add(x, y):
    return x + y
```

Şimdi elimizde:

In [111]:

```
x = np.arange(10)
```

In [112]:

```
nb_add(x, x)
```

Out[112]:

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18], dtype=int64)
```

```
nb_add.accumulate(x, 0)
array([ 0., 1., 3., 6., 10., 15., 21., 28., 36., 45.])
```

A.8 Gelişmiş Dizi Giriş ve Çıkışı

Bellek Eşlemeli Dosyalar

Bellek eşlemeli dosya, diskteki ikili verilerle bellek içi bir dizide depolananmiş gibi etkileşime girmeye yönelik yöntemdir. NumPy, ndarray benzeri bir memmap nesnesi uygular ve büyük bir dosyanın küçük bölümlerinin tüm diziyi belleğe okumanadan okunmasını ve yazılmasını sağlar. Ek olarak, bir memmap, bellek içi dizile aynı yöntemlere sahiptir ve bu nedenle, bir ndarray'in bekleniği birçok algoritmeye ikame edilebilir.

Yeni bir bellek haritası oluşturmak için np.memmap işlevini kullanın ve bir dosya yolu, dtype, şekil ve dosya modu iletin.

In [120]:

```
mmap = np.memmap('mymmap', dtype='float64', mode='w+',  
                  shape=(10000, 10000))
```

```
-----  
----  
OSError                                     Traceback (most recent call  
last)  
<ipython-input-120-615b37651eb9> in <module>  
----> 1 mmap = np.memmap('mymmap', dtype='float64', mode='w+',  
                      shape=(10000, 10000))  
  
-\anaconda3\lib\site-packages\numpy\core\memmap.py in __new__(subtype,  
filename, dtype, mode, offset, shape, order)  
    223         f_ctx = contextlib_nullcontext(filename)  
    224     else:  
--> 225         f_ctx = open(os_fspath(filename), ('r' if mode ==  
'c' else mode)+'b')  
    226  
    227     with f_ctx as fid:
```

OSError: [Errno 22] Invalid argument: 'mymmap'

In [114]:

```
mmap
```

```
-----  
----  
NameError                                     Traceback (most recent call  
last)  
<ipython-input-114-49f7bdb78b33> in <module>  
----> 1 mmap  
  
NameError: name 'mmap' is not defined
```

Bir memmap'i dilimlemek, diskteki verilerin görüntülerini döndürür.

In [115]:

```
section = mmap[:5]
```

```
-----  
----  
NameError                                     Traceback (most recent call  
last)  
<ipython-input-115-447bf9af22d1> in <module>  
----> 1 section = mmap[:5]  
  
NameError: name 'mmap' is not defined
```

Bunlara veri atarsanız, bellekte arabelleğe alınır (bir Python dosya nesnesi gibi), ancak flush çağırarak diske yazabilirsiniz.

In [116]:

```
section[:, :] = np.random.randn(5, 10000)
```

In [117]:

```
mmap.flush()
```

```
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-117-77a852a0fe13> in <module>
----> 1 mmap.flush()

NameError: name 'mmap' is not defined
```

In [118]:

```
mmap
```

```
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-118-49f7bdb78b33> in <module>
----> 1 mmap

NameError: name 'mmap' is not defined
```

In [119]:

```
del mmap
```

```
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-119-5e349765e424> in <module>
----> 1 del mmap

NameError: name 'mmap' is not defined
```

Bir bellek haritası kapsam dışında kaldığında ve çöp olarak toplandığında, tüm değişiklikler diske de temizlenir. Mevcut bir bellek haritasını açarken, dosya diskte meta verisi olmayan yalnızca bir ikili veri bloğu olduğundan, dtype ve şekli belirtmeniz gereklidir.

In [121]:

```
mmap = np.memmap('mymmap', dtype='float64', shape=(10000, 10000))
```

In [122]:

mmap

Out[122]:

```
memmap([[ 0.22580795, -0.69554784,  0.95864928, ..., -0.42383522,
          1.04535362,  0.78342041],
         [-0.62771634, -0.52746684, -0.23385603, ...,  0.74430743,
          -1.4885139 ,  0.98646325],
         [-0.73733238, -0.27137836,  0.74829801, ...,  0.35278418,
          -0.33351335,  0.28127671],
         ...,
         [ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.          ],
         [ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.          ],
         [ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.          ],
         [ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.          ]])
```

Bellek eşlemeleri, yapılandırılmış veya iç içe geçmiş tiplerle de çalışır.

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. HDF5 biçiminde yüzlerce gigabayt ve hatta terabayt veriyi güvenle depolayabilirsiniz.

A.9 Performans İpuçları

NumPy'yi kullanarak koddan iyi performans elde etmek genellikle basittir, çünkü dizi işlemleri tipik olarak nispeten çok yavaş olan saf Python döngüleri yerini alır. Aşağıdaki liste, akılda tutulması gereken bazı noktaları kısaca özetlemektedir:

- Python döngülerini ve koşullu mantığı dizi işlemlerine ve mantıksal dizi işlemlerine dönüştürün
- Mümkün olduğunda yayın kullanın
- Verilerin kopyalanmasını önlemek için dizi görünümleri (dilimleme) kullanın
- Ufuncs ve ufunc yöntemlerini kullanın

Yalnızca NumPy tarafından sağlanan yetenekleri tükettiğten sonra ihtiyacınız olan performansı elde edemezseniz.

Bitişik Belleğin Önemi

Bu konunun tam kapsamı bu kitabın kapsamı dışında biraz olsa da, bazı uygulamalarda bir dizinin bellek düzeni hesaplamaların hızını önemli ölçüde etkileyebilir. Bu, kısmen CPU'nun önbellek hiyerarşisiyle ilgili performans farklılıklarına dayanmaktadır; bitişik bellek bloklarına erişen işlemler (örneğin, bir C sipariş dizisinin satırlarının toplanması) genellikle en hızlı olacaktır çünkü bellek alt sistemi uygun bellek bloklarını ultra hızlı L1 veya L2 Cache önbelleğine tamponlayacaktır. Ayrıca, NumPy'nin C kod tabanı içindeki belirli kod yolları, genel adım adım bellek erişiminin önlenebileceği bitişik durum için optimize edilmiştir.

Bir dizinin bellek düzeninin bitişik olduğunu söylemek, öğelerin Fortran (satır büyük) veya C (satır majör) sıralamasına göre dizide göründükleri sırayla bellekte depolandığı anlamına gelir. Varsayılan olarak NumPy dizileri bitişik veya sadece bitişik olarak oluşturulur. Bir C-bitişik dizinin devri gibi bir sütun ana dizisinin bu nedenle Fortran-bitışik olduğu söylenir. Bu özellikler, ndarray üzerindeki flags özniteliği aracılığıyla açıkça kontrol edilebilir.

In [123]:

```
arr_c = np.ones((1000, 1000), order='C')
```

In [124]:

```
arr_f = np.ones((1000, 1000), order='F')
```

In [125]:

```
arr_c.flags
```

Out[125]:

```
C_CONTIGUOUS : True
F_CONTIGUOUS : False
OWNDATA : True
WRITEABLE : True
ALIGNED : True
WRITEBACKIFCOPY : False
UPDATEIFCOPY : False
```

In [126]:

```
arr_f.flags
```

Out[126]:

```
C_CONTIGUOUS : False
F_CONTIGUOUS : True
OWNDATA : True
WRITEABLE : True
ALIGNED : True
WRITEBACKIFCOPY : False
UPDATEIFCOPY : False
```

In [127]:

```
arr_f.flags.f_contiguous
```

Out[127]:

```
True
```

Bu örnekte, bu dizilerin satırlarını toplamak teorik olarak arr_c için arr_f'den daha hızlı olmalıdır çünkü satırlar bellekte bitişiktir.

In [128]:

```
%timeit arr_c.sum(1)
```

```
1.16 ms ± 193 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [129]:

```
%timeit arr_f.sum(1)
```

```
809 µs ± 83 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

NumPy'den daha fazla performans elde etmek istediğinizde, burası genellikle biraz çaba harcamak için bir yerdır. İstenilen hafıza sırasına sahip olmaya n bir diziniz varsa, kopyalayıp 'C' veya 'F' kullanabilirsiniz.

In [130]:

```
arr_f.copy('C').flags
```

Out[130]:

```
C_CONTIGUOUS : True
F_CONTIGUOUS : False
OWNDATA : True
WRITEABLE : True
ALIGNED : True
WRITEBACKIFCOPY : False
UPDATEIFCOPY : False
```

Bir dizi üzerinde bir görünüm oluştururken, sonucun bitişik olmasının garanti edilmediğini unutmayın!

In [131]:

```
arr_c[:50].flags.contiguous
```

Out[131]:

```
True
```

In [132]:

```
arr_c[:, :50].flags
```

Out[132]:

```
C_CONTIGUOUS : False
F_CONTIGUOUS : False
OWNDATA : False
WRITEABLE : True
ALIGNED : True
WRITEBACKIFCOPY : False
UPDATEIFCOPY : False
```

IPython Sistemi hakkında daha fazla bilgi

B.1 Using the Command History

IPython, yürüttüğünüz her komutun metnini içeren küçük bir disk üzerinde veritabanı tutar. Bu, çeşitli amaçlara hizmet eder:

- Daha önce yürütülen komutları minimum yazı ile arama, tamamlama ve yürütme
- Oturumlar arasında komut geçmişini sürdürme
- Giriş / çıkış geçmişini bir dosyaya kaydetme

Bu özellikler, tasarım gereği dizüstü bilgisayar her bir kod hücresinde giriş ve çıkışın bir kaydını tuttuğundan, kabukta dizüstü bilgisayardan daha kullanışlıdır.

Komut Geçmişini Arama ve Yeniden Kullanma

IPython kabuğu, önceki kodu veya diğer komutları aramanızı ve çalıştırmanızı sağlar. Bu kullanışlıdır, çünkü kendinizi genellikle % run komutu veya başka bir kod parçası gibi aynı komutları tekrarlarken bulabilirsiniz. Diyelim ki:

```
In[7]: %run first/second/third/data_script.py
```

ve sonra yalnızca yanlış bir hesaplama yaptığınızı bulmak için komut dosyasının sonuçlarını araştırdı (başarılı bir şekilde çalıştığını varsayıarak). Sorunu anladıkten ve data_script.py'yi değiştirdikten sonra, % run komutunun bir kaç harfini yazmaya başlayabilir ve ardından Ctrl-P tuş kombinasyonuna veya yukarı ok tuşuna basabilirsiniz. Bu, yazdığınız harflerle eşleşen önceki ilk komut için komut geçmişini arayacaktır. Ctrl-P veya yukarı ok tuşuna birden çok kez basmak geçmişte aramaya devam edecektir. Yürütmek istediğiniz komutu devrederseniz, korkmayın. Ctrl-N veya aşağı ok tuşlarına basarak komut geçmişinde ilerleyebilirsiniz. Bunu birkaç kez yaptıktan sonra düşünmeden bu tuşlara basmaya başlayabilirsiniz! Ctrl-R'yi kullanmak, bash kabuğu gibi Unix tarzı kabuklarda kullanılan okuma çizgisinin sağladığı aynı kısmi artımlı arama yeteneğini sağlar. Windows'ta, okuma satırı işlevselligi IPython tarafından taklit edilir. Bunu kullanmak için Ctrl-R tuşlarına basın ve ardındanaramak istediğiniz giriş satırında bulunan birkaç karakteri yazın:

```
In [1]: a_command = foo(x, y, z)
(reverse-i-search)`com': a_command = foo(x, y, z)
```

Ctrl-R tuşlarına basmak, yazdığınız karakterlerle eşleşen her satır için geçmişte gezinir.

Giriş ve Çıkış Değişkenleri

Bir fonksiyon çağrısının sonucunu bir değişkene atamayı unutmak çok sıkıcı olabilir. Bir IPython oturumu, hem giriş komutlarına hem de çıktı Python nesnelerine başvuruları özel değişkenlerde depolar. Önceki iki çıktı sırasıyla _ (bir alt çizgi) ve __ (iki alt çizgi) değişkenlerinde saklanır.

```
In [1]:
```

```
2 ** 27
```

```
Out[1]:
```

```
134217728
```

```
In [2]:
```

```
—
```

```
Out[2]:
```

```
134217728
```

Girdi değişkenleri, `_ix` gibi adlandırılan değişkenlerde saklanır; burada `x`, girdi satırı numarasıdır. Her girdi değişkeni için karşılık gelen bir çıktı değişkeni `_X` vardır. Yani, 27 numaralı giriş satırından sonra, iki yeni değişken `_27` (çıktı için) ve girdi için `_i27` olacaktır.

```
In [3]:
```

```
foo = 'bar'
```

```
In [8]:
```

```
foo
```

```
Out[8]:
```

```
'bar'
```

```
_i27  
u'foo'
```

```
_27  
'bar'
```

Girdi değişkenleri dizeler olduğundan, Python `exec` anahtar sözcüğü ile tekrar çalıştırılabilirler.

```
exec(_i27)
```

Çeşitli sihirli işlevler, girdi ve çıktı geçmişiyile çalışmanızı izin verir. `% hist`, girdi geçmişinin tamamını veya bir kısmını satır numaraları olsun v

eya olmasın yazdırabilir. % reset, etkileşimli ad alanını ve istege bağlı olarak giriş ve çıkış önbelleklerini temizlemek içindir. % Xdel sihirli işlev i, belirli bir nesneye yapılan tüm başvuruları IPython makinesinden kaldırma k için tasarlanmıştır. Daha fazla ayrıntı için bu sihirlerin her ikisinin belgelerine bakın.

B.2 İşletim Sistemiyle Etkileşim

IPython'un diğer bir özelliği, dosya sistemine ve işletim sistemi kabuğuna sorunsuz bir şekilde erişmenize izin vermesidir. Bu, diğer şeylerin yanı sıra, çoğu standart komut satırı eylemini Windows veya Unix (Linux, macOS) kabuğunda olduğu gibi IPython'dan çıkmak zorunda kalmadan gerçekleştirebileceğinizi anlamına gelir. Bu, kabuk komutlarını, dizinleri değiştirmeyi ve bir komutun sonuçlarını bir Python nesnesinde (liste veya dize) depolamayı içerir. Ayrıca basit komut takma ve dizin yerimi özelliklerini de vardır.

Kabuk komutlarını çağrırmak için işlevlerin ve sözdiziminin bir özeti için Tablo B-1'e bakın.

In [14]:

```
from IPython.display import Image
Image(filename='img/Picture96.png')
```

Out[14]:

Table B-1. IPython system-related commands

Command	Description
!cmd	Execute cmd in the system shell
output = !cmd args	Run cmd and store the stdout in output
%alias alias_name cmd	Define an alias for a system (shell) command
%bookmark	Utilize IPython's directory bookmarking system
%cd directory	Change system working directory to passed directory
%pwd	Return the current system working directory
%pushd directory	Place current directory on stack and change to target directory
%popd	Change to directory popped off the top of the stack
%dirs	Return a list containing the current directory stack
%dhist	Print the history of visited directories
%env	Return the system environment variables as a dict
%matplotlib	Configure matplotlib integration options

Shell Komutları ve Takma Adları

Python'da bir ünlem işaretiyile! Veya bang ile bir satır başlatmak, IPython'a sistem kabuğundaki çarpmadan sonra 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 atayarak bir değişkende saklayabilirsiniz.

In [15]:

```
ip_info = !ifconfig wlan0 | grep "inet "
```

In [16]:

```
ip_info[0].strip()
```

Out[16]:

```
"'ifconfig' is not recognized as an internal or external command,"
```

Döndürülen Python nesnesi ip_info aslında konsol çıktısının çeşitli sürümlerini içeren özel bir liste türüdür.

IPython ayrıca! Kullanılırken mevcut ortamda tanımlanan Python değerlerinin yerini alabilir. Bunu yapmak için, değişken adının önüne dolar işareteti \$ koynın.

In [17]:

```
foo = 'test*'
```

!ls \$foo

% Alias magic işlevi, kabuk komutları için özel kısayollar tanımlayabilir. Basit bir örnek olarak:

In [19]:

```
%alias ll ls -l
```

```
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 çok komutu aynı komut satırında olduğu gibi noktalı virgülle ayırarak çalıştırabilirsiniz.

In [21]:

```
%alias test_alias (cd examples; ls; cd ..)
```

test_alias

IPython'un, oturum kapanır kapanmaz etkileşimli olarak tanımladığınız tüm takma adları "unuttuğunu" fark edeceksiniz. Kalıcı takma adlar oluşturmak için konfigürasyon sistemini kullanmanız gerekecektir.

Dizin Yer İşareti Sistemi

IPython, çok kolay bir şekilde atlayabilmeniz için ortak dizinler için takma adlar kaydetmenize olanak tanıyan basit bir dizin yerimi sisteme sahiptir. Örneğin, bu kitap için ek materyallere işaret eden bir yerimi oluşturmaktan istedığınızı varsayıyalım:

In [23]:

```
%bookmark py4da /home/wesm/code/pydata-book
```

Bunu yaptıktan sonra, %cd büyüsünü kullandığımızda, tanımladığımız herhangi bir yer işaretini kullanabiliriz.

In [24]:

```
cd py4da
```

```
(bookmark:py4da) -> /home/wesm/code/pydata-book
[WinError 3] Sistem belirtilen yolu bulamıyor: '/home/wesm/code/pydata-book'
C:\Users\user
```

Bir yerimi adı, geçerli çalışma dizininizdeki bir dizin adıyla çakışırsa, yerimi konumunu geçersiz kılmak ve kullanmak için -b bayrağını kullanabilirsiniz. % Bookmark ile -l seçeneğini kullanmak tüm yer imlerinizi listeler.

In [25]:

```
%bookmark -l
```

```
Current bookmarks:  
py4da -> /home/wesm/code/pydata-book
```

Diğer adlardan farklı olarak yer imleri, IPython oturumları arasında otomatik olarak saklanır.

B.3 Yazılım Geliştirme Araçları

IPython, etkileşimli bilgi işlem ve veri keşfi için rahat bir ortam olmasını 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 istersiniz. Bunun için IPython, kullanımını kolay kod zamanlama ve profil oluşturma araçlarına sahiptir.

Etkileşimli Hata Ayıklayıcı

IPython'un hata ayıklayıcısı, istisna izlemelerindeki her satır için sekme zamanlama, sözdizimi vurgulama ve bağlam ile pdb'yi geliştirir. Kodda hata ayıklamanın en iyi zamanlarından biri, bir hata oluştuktan hemen sonradır. % Hata ayıklama komutu, bir istisnadan hemen sonra girildiğinde, "ölüm sonrası" hata ayıklayıcıyı çağırır ve sizi istisnanın ortaya çıktığı yığın çerçevesine bırakır.

```
run examples/ipython_bug.py
```

In [27]:

Image(filename='img/Picture97.png')

Out[27]:

```
-----  
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, rastgele Python kodunu çalıştırabilir ve her yığın çerçevesinin içindeki (yorumlayıcı tarafından "canlı tutulan") tüm nesneleri ve verileri keşfedebilirsiniz. Varsayılan olarak, hatanın meydana geldiği en düşük seviyeden başlarsınız. U (yükarı) ve d (aşağı) tuşlarına basarak yığın izleme seviyeleri arasında geçiş yapabilirsiniz.

In [28]:

Image(filename='img/Picture98.png')

Out[28]:

```

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 hata ayıklayıcıyı otomatik olarak çağırmasını sağlar; bu, birçok kullanıcının özellikle yararlı bulacağı bir moddur.

Ayrıca, özellikle her aşamada durumu incelemek için kesme noktaları ayarlama veya bir işlevin veya komut dosyasının yürütülmesinde adım adım ilerlemek istediğinizde, kod geliştirmeye yardımcı olmak için hata ayıklayıcıyı kullanmak kolaydır. Bunu başarmanın birkaç yolu var. Birincisi, aktarılan komut dosyasında herhangi bir kodu çalıştırmadan önce hata ayıklayıcıyı çağırın -d bayrağına sahip% run kullanmaktadır. Komut dosyasını girmek için hemen s (adı m) tuşuna basmalısınız.

run -d examples/ipython_bug.py

In [30]:

Image(filename='img/Picture99.png')

Out[30]:

```

In [5]: 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> s
--Call--
> /home/wesm/code/pydata-book/examples/ipython_bug.py(1)<module>()
1---> 1 def works_fine():
  2     a = 5
  3     b = 6

```

Bu noktadan sonra, dosya içinde nasıl ilerlemek istediğiniz size kalmıştır. Örneğin, önceki istisnada, works_fine yöntemini çağırmadan hemen önce bir kesme noktası ayarlayabilir ve c (devam) tuşuna basarak kesme noktasına ulaşana kadar betiği çalıştırabiliriz.

In [31]:

```
Image(filename='img/Picture100.png')
```

Out[31]:

```
ipdb> b 12
ipdb> c
> /home/wesm/code/pydata-book/examples/ipython_bug.py(12)calling_things()
  11 def calling_things():
  12     works_fine()
  13     throws_an_exception()
```

Bu noktada, bir sonraki satıra ilerlemek için n (sonraki) tuşuna basarak works_fine () içine geçebilir veya works_fine () işlemini yürütebilirsiniz.

In [32]:

```
Image(filename='img/Picture101.png')
```

Out[32]:

```
ipdb> n
> /home/wesm/code/pydata-book/examples/ipython_bug.py(13)calling_things()
  2     12     works_fine()
  3 ---> 13     throws_an_exception()
  14
```

Ardından, throws_an_exception'a geçebilir ve hatanın olduğu satıra ilerleyebilir ve kapsamındaki değişkenlere bakabiliriz. Hata ayıklayıcı komutlarının değişken adlarından öncelikli olduğunu unutmayın; bu gibi durumlarda, değişkenlere! içeriğini incelemek için:

In [33]:

Image(filename='img/Picture102.png')

Out[33]:

```
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
```

Etkileşimli hata ayıklayıcıyla yeterlilik geliştirmek, büyük ölçüde bir uygulama ve deneyim meselesidir. Hata ayıklayıcı komutlarının tam kataloğu için Tablo B-2'ye bakın. Bir IDE kullanmaya alışkinsanız, uşbirime dayalı hata ayıklayıcıyı ilk başta biraz affetmez bulabilirsiniz, ancak bu zamanla geliş

cektir. Bazı Python IDE'lerinin mükemmel GUI hata ayıklayıcıları vardır, bu nedenle çoğu kullanıcı kendileri için çalışan bir şey bulabilir.

In [34]:

```
Image(filename='img/Picture103.png')
```

Out[34]:

Table B-2. (I)Python debugger commands

Command	Action
h(elp)	Display command list
help <i>command</i>	Show documentation for <i>command</i>
c(ontinue)	Resume program execution
q(uit)	Exit debugger without executing any more code
b(reak) <i>number</i>	Set breakpoint at <i>number</i> in current file
b <i>path/to/file.py:number</i>	Set breakpoint at line <i>number</i> in specified file
s(tep)	Step into function call
n(ext)	Execute current line and advance to next line at current level
u(p)/d(own)	Move up/down in function call stack
a(rgs)	Show arguments for current function
debug <i>statement</i>	Invoke statement <i>statement</i> in new (recursive) debugger
l(ist) <i>statement</i>	Show current position and context at current level of stack
w(here)	Print full stack trace with context at current position

Hata ayıklayıcıyı kullanmanın diğer yolları

Hata ayıklayıcıyı çağrımanın birkaç başka yararlı yolu vardır. Birincisi, te melde "zayıf adının kesme noktası" olan özel bir set_trace işlevi (pdb.set_t race olarak adlandırılmıştır) kullanmaktadır.

In [35]:

```
from IPython.core.debugger import Pdb
```

In [36]:

```
def set_trace():
    Pdb(color_scheme='Linux').set_trace(sys._getframe().f_back)
```

In [37]:

```
def debug(f, *args, **kwargs):
    pdb = Pdb(color_scheme='Linux')
    return pdb.runcall(f, *args, **kwargs)
```

İlk işlev olan set_trace çok basittir. Daha yakından incelemek için kodunuzu n herhangi bir bölümünde geçici olarak durdurmak istediğiniz bir set_trace k ullanabilirsiniz (örneğin, bir istisna oluşmadan hemen önce):

run examples/ipython_bug.py

In [39]:

```
Image(filename='img/Picture104.png')
```

Out[39]:

```
> /home/wesm/code/pydata-book/examples/ipython_bug.py(16)calling_things()
  15     set_trace()
----> 16     throws_an_exception()
  17
```

C (coninue) tuşuna basmak, kodun herhangi bir zarar vermeden normal şekilde devam etmesine neden olur.

Az önce baktığımız hata ayıklama işlevi, rastgele bir işlev çağrısında etkil eşimli hata ayıklayıcıyı kolayca çalıştırmanızı sağlar. Diyelim ki aşağıdaki gibi bir fonksiyon yazmıştık ve mantığından geçmek istedik.

In [40]:

```
def f(x, y, z=1):
    tmp = x + y
    return tmp / z
```

Normalde f kullanmak f (1, 2, z = 3) gibi görünür. Bunun yerine f'ye adım atmak için, hata ayıklanacak ilk argüman olarak f'yi ve ardından f'ye iletilecek konumsal ve anahtar kelime argümanlarını iletin.

In []:

```
debug(f, 1, 2, z=3)
```

```
<ipython-input-37-75010aeaf1f4>:2: DeprecationWarning: The `color_sche
me` argument is deprecated since version 5.1
  pdb = Pdb(color_scheme='Linux')

> <ipython-input-40-359ec13d6433>(2)f()
  1 def f(x, y, z=1):
----> 2     tmp = x + y
  3     return tmp / z
```

Son olarak, hata ayıklayıcı% run ile birlikte kullanılabilir. % Run -d ile bir komut dosyası çalıştırıldığınızda, doğrudan hata ayıklayıcıya bırakılırsını z, herhangi bir kesme noktası ayarlamaya ve betiği başlatmaya hazır olursunu z.

In []:

```
%run -d examples/ipython_bug.py
```

Bir satır numarasıyla -b eklemek, hata ayıklayıcıyı önceden ayarlanmış bir k esme noktasıyla başlatır.

In []:

```
%run -d -b2 examples/ipython_bug.py
```

Zamanlama Kodu:% time ve% timeit

Daha büyük ölçekli veya daha uzun süreli veri analizi uygulamaları için, çes itli bileşenlerin veya tek tek ifadelerin veya işlev çağrılarının yürütme sü resini ölçmek isteyebilirsiniz. Karmaşık bir süreçte hangi işlevlerin en çok zaman aldığına dair bir rapor isteyebilirsiniz. Neyse ki IPython, kodunuzu g elistirirken ve test ederken bu bilgileri çok kolay bir şekilde almanızı sağ lar.

Yerleşik zaman modülüne ve zaman.clock ve time.time işlevlerini kullanarak e lle zamanlama kodu, aynı ilginç olmayan standart kodu yazmanız gerekiğinde n, genellikle sıkıcı ve tekrarlayıcıdır.

In []:

```
import time
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 işlemi sizin için otomatik leştirmek için iki sihirli işlevi vardır: %time ve %timeit.

% time, bir ifadeyi bir kez çalıştırarak toplam yürütme süresini bildirir. B üyük bir dizi listemiz olduğunu ve belirli bir önek ile başlayan tüm dizeler i seçenekin farklı yöntemlerini karşılaştırmak istediğimizi varsayıyalım. Burad a 600.000 dizeden oluşan basit bir liste ve yalnızca 'foo' ile başlayanları s eğmenin iki özdeş yöntemi.

In []:

```
# çok geniş bir dizi listesi
strings = ['foo', 'foobar', 'baz', 'qux',
           'python', 'Guido Van Rossum'] * 100000
```

In []:

```
method1 = [x for x in strings if x.startswith('foo')]
method2 = [x for x in strings if x[:3] == 'foo']
```

Performans açısından aynı olmaları gerekiyor gibi görünüyor, değil mi? % Tim e kullanarak kesin olarak kontrol edebiliriz.

In []:

```
%time method1 = [x for x in strings if x.startswith('foo')]
```

In []:

```
%time method2 = [x for x in strings if x[:3] == 'foo']
```

Duvar zamanı ("duvar saatı" nin kısaltması) ilgi çeken ana sayıdır. Bu neden le, ilk yöntem iki kattan fazla sürüyor gibi görünüyor, ancak bu çok kesin b ir ölçüm değil. Bu ifadeleri birden çok kez% zamanlamayı denerseniz, sonuçla rın biraz değişken olduğunu göreceksiniz. Daha hassas bir ölçüm elde etmek için kullanın

%timeit sihirli işlevi. Rasgele bir ifade verildiğinde, daha doğru bir ortal ama çalışma süresi üretmek için bir ifadeyi birden çok kez çalıştırınmak için bir buluşsal yöntem vardır.

In []:

```
%timeit [x for x in strings if x.startswith('foo')]
```

In []:

```
%timeit [x for x in strings if x[:3] == 'foo']
```

Görünüşte zararsız olan bu örnek, Python standart kitaplığının, NumPy'nin, p andaların ve bu kitapta kullanılan diğer kitaplıkların performans özellikler ini anlamaya değer olduğunu göstermektedir. Daha büyük ölçekli veri analizi uygulamalarında, bu milisaniyeler toplanmaya başlayacak

% timeit, özellikle mikrosaniye (saniyenin milyonda biri) veya nanosaniye (s aniyenin milyarda biri) düzeyinde bile çok kısa yürütme sürelerine sahip ifa deleri ve işlevleri analiz etmek için kullanışlıdır. Bunlar ömensiz miktarda zaman gibi görünebilir, ancak elbette 1 milyon kez çalıştırılan 20 mikrosani

ye bir işlev 15 saniye sürer
 5 mikrosaniye işlevinden daha uzun. Önceki örnekte, performans özelliklerini anlamak için iki dizgi işlemini doğrudan karşılaştırabiliriz.

In []:

```
x = 'foobar'
```

In []:

```
y = 'foo'
```

In []:

```
%timeit x.startswith(y)
```

In []:

```
%timeit x[:3] == y
```

Temel Profil Oluşturma:% prun ve% run -p

Profil oluşturma kodu, zamanın nerede harcandığının belirlenmesi dışında, zamanlama koduya yakından ilgiliidir. Ana Python profil oluşturma aracı, IPython'a hiç özgü olmayan cProfile modülüdür. cProfile, her işlevde ne kadar zaman harcandığını takip ederken bir programı veya herhangi bir rastgele kod bloğunu yürütür.

CProfile'ı kullanmanın yaygın bir yolu, komut satırında, tüm bir programı çağıştırmak ve işlev başına toplanan süreyi çıkarmaktır. Bir döngüde bazı doğrusal cebir yapan basit bir betiğiimiz olduğunu varsayıyalım.

In []:

```
import numpy as np
from numpy.linalg import eigvals

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)
```

Komut satırında aşağıdakileri kullanarak bu betiği cProfile üzerinden çalıştırabilirsiniz.

In []:

```
python -m cProfile cprof_example.py
```

Bunu denerseniz, çıktıının işlev adına göre sıralandığını göreceksiniz. Bu, en çok nerede zaman harcandığına dair bir fikir edinmeyi biraz zorlaştırır, bu nedenle -s işaretini kullanarak bir sıralama düzeni belirtmek çok yaygındır.

In []:

```
python -m cProfile -s cumulative cprof_example.py
```

In []:

```
Image(filename='img/Picture105.png')
```

Çıktının yalnızca ilk 15 satırı gösterilir. Her işlevde toplam ne kadar zaman harcadığını görmek için cumtime sütununu tarayarak okumak en kolayıdır. Bir işlev başka bir işlevi çağırırsa, saatin çalışmayı durdurmayacağını unutmayın. cProfile, her işlev çağrısının başlangıç ve bitiş zamanını kaydeder ve zamanlamayı oluşturmak için bunu kullanır.

Komut satırı kullanımına ek olarak cProfile, yeni bir işlem çalıştırılmaya ger ek kalmadan rastgele kod bloklarını profillemek için programlı olarak da kullanılabilir. IPython, % prun komutunu ve % run için -p seçeneğini kullanan bu özellik için 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 []:

```
%prun -l 7 -s cumulative run_experiment()
```

In []:

```
Image(filename='img/Picture106.png')
```

Benzer şekilde, % run -p -s kümülatif cprof_example.py'yi çağırırmak, IPython'dan asla ayrılmak zorunda kalmamanız dışında komut satırı yaklaşımıyla aynı etkiye sahiptir. Jupyter not defterinde, anbentire kod bloğunun profilini çekarmak için %% budama sihrini (% iki işaretti) kullanabilirsiniz. Bu, profil çıktılarıyla ayrı bir pencere açar. Bu, "Bu kod bloğunun çalışması neden bu kadar uzun sürdü?" Gibi sorulara muhtemelen hızlı yanıtlar almak için yararlı olabilir.

IPython veya Jupyter kullanırken profillerin anlaşılmasını kolaylaştırmaya yardımcı olan başka araçlar da vardır. Bunlardan biri, d3.js kullanarak profil sonuçlarının etkileşimli bir görselleştirmesini oluşturan SnakeViz'dir.

Bir Fonksiyon Satırını Satır Profilleme

Bazı durumlarda, %prun'dan (veya başka bir cProfile tabanlı profil yönteminden) edindiğiniz bilgiler, bir işlevin çalıştırılma süresilarındaki tüm hikâyeyi anlatmayabilir veya o kadar karmaşık olabilir ki, işlev adına göre toplanan sonuçların yorumlanması zordur. . Bu durum için line_profiler adında küçük bir kitaplık vardır (PyPI veya paket yönetim araçlarından biri aracılığıyla edinilebilir). Bir veya daha fazla işlevin satır satır profillemesini hesaplayan yeni bir %lprun sihirli işlevini etkinleştirerek bir IPython uzantısı içerir. Bu uzantıyı, IPython yapılandırmanızı değiştirerek etkinleştirebilirsiniz (IPython belgelerine veya bu bölümün ilerleyen kısımlarında yapılandırma ile ilgili bölüme bakın) aşağıdaki satırı içerecek şekilde:

In []:

```
# Yüklenenek IPython uzantılarının noktalı modül adlarının listesi.
c.TerminalIPythonApp.extensions = ['line_profiler']
```

Ayrıca şu komutu da çalıştırabilirsiniz:

In []:

```
%load_ext line_profiler
```

line_profiler programı olarak kullanılabilir ancak IPython'da etkileşimli olarak kullanıldığında belki de en güclüsüdür. Bazı NumPy dizi işlemlerini gerçekleştiren aşağıdaki koda sahip bir prof_mod modülünüz olduğunu varsayılmam.

In []:

```
from numpy.random import randn
```

In []:

```
def add_and_sum(x, y):
    added = x + y
    summed = added.sum(axis=1)
    return summed
def call_function():
    x = randn(1000, 1000)
    y = randn(1000, 1000)
    return add_and_sum(x, y)
```

Add_and_sum işlevinin performansını anlamak istersek, % prun bize şunu verir:

In []:

```
%run prof_mod
```

In []:

```
x = randn(3000, 3000)
```

In []:

```
y = randn(3000, 3000)
```

In []:

```
%prun add_and_sum(x, y)
```

Bu özellikle aydınlatıcı değil. Line_profiler IPython uzantısı etkinleştiril diğinde, yeni bir komut % lprun kullanılabilir. Kullanımdaki tek fark, profillemek istediğimiz işlevi veya işlevleri % lprun'a bildirmemiz gerektiyidir. Genel sözdizimi şöyledir:

In []:

```
%lprun -f func1 -f func2 statement_to_profile
```

Bu durumda, add_and_sum profilini oluşturmak istiyoruz, bu yüzden şunu çalıştırıyoruz:

In []:

```
%lprun -f add_and_sum add_and_sum(x, y)
```

In []:

```
Image(filename='img/Picture107.png')
```

Bunu yorumlamak çok daha kolay olabilir. Bu durumda, ifadede kullandığımız işlevin profilini çıkardık. Önceki modül koduna baktığımızda, add_and_sum'un yanı sıra call_function ve profile de çağrılabılır, böylece kodun performansının tam bir resmini elde edebiliriz.

In []:

```
%lprun -f add_and_sum -f call_function call_function()
```

In []:

```
Image(filename='img/Picture108.png')
```

B.4 IPython Kullanarak Üretken Kod Geliştirme İçin İpuçları

Geliştirmeyi, hata ayıklamayı ve nihayetinde interaktif olarak kullanmayı kolaylaşırıacak şekilde kod yazmak, birçok kullanıcı için bir paradigma değişikliği olabilir. Kod yeniden yükleme gibi bazı ayarlamalar ve kodlama stili endişeleri gerektirebilecek prosedür ayrıntıları vardır.

Bu özellikle bir şeyler ters gittiğinde ve sizin veya bir başkasının aylar veya yıllar önceden yazmış olabileceği bir koddaki bir hatayı teşhis etmeniz gerektiğinde önemlidir.

Modül Bağımlılıklarını Yeniden Yükleme

Python'da, `import some_lib` yazdığınızda, `some_lib`'deki kod çalıştırılır ve içinde tanımlanan tüm değişkenler, işlevler ve içe aktarmalar, yeni oluşturulmuş `some_lib` modül ad alanında depolanır. Bir dahaki sefere `import some_lib` yazdığınızda, mevcut modül ad alanına bir başvuru alacaksınız. Etkileşimde potansiyel zorluk

IPython kod geliştirme,örneğin% değişiklik yapmış olabileceğiniz başka bir module'le bağlı bir komut dosyası çalıştırığınızda gelir. `Test_script.py` dosyasında aşağıdaki koda sahip olduğumu varsayıyalım:

In []:

```
import some_lib
x = 5
y = [1, 2, 3, 4]
result = some_lib.get_answer(x, y)
```

% Run `test_script.py`'yi çalıştıracak ve sonra `some_lib.py`'yi değiştirecek olursanız, bir dahaki sefer% run `test_script.py`'yi çalıştırığınızda, Python'un "bir kez yükle" modül sistemi nedeniyle `some_lib.py`'nin eski sürümünü almaya devam edeceksiniz. Bu davranış bazlarından farklı MATLAB gibi kod değişikliklerini otomatik olarak yayan diğer veri analizi ortamları. Bununla başa çıkmak için birkaç seçenekiniz var. İlk yol, standart kitaplıktaki `importlib` modülündeki yeniden yükleme işlevini kullanmaktır.

In []:

```
import some_lib
import importlib
importlib.reload(some_lib)
```

Bu, `test_script.py`'yi her çalıştırığınızda `some_lib.py`'nin yeni bir kopyasını alacağınızı garanti eder. Açıkta kılınır ki, bağımlılıklar daha derine inerse, her yere yeniden yükleme kullanımları eklemek biraz zor olabilir. Bu sorun iç in IPython, modüllerin "derin" (özyinelemeli) yeniden yüklenmesi için özel bir `reload` işlevine (sihirli bir işlev değil) sahiptir. `Some_lib.py`'yi çalış tırırsam `reload(some_lib)` yazarsam, `bazı_lib`'i ve tüm bağımlılıklarını yen iden yüklemeyi deneyecektir. Bu maalesef her durumda çalışmayacaktır, ancak işe yaradığında IPython'u yeniden başlatmak zorunda kalmaktan daha iyidir.

Kod Tasarım İpuçları

Bunun basit bir tarifi yok, ancak işte kendi çalışmamda etkili bulduğum bazı üst düzey ilkeler.

İlgili nesneleri ve verileri canlı tutun

Aşağıdaki önemsiz örnek gibi bir yapıya sahip komut satırı için yazılmış bir program görmek alışılmadık bir durum değildir.

In []:

```
from my_functions import g

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 çalıştırırsak neyin yanlış gidebileceğini görüyor mus unuz? Tamamlandıktan sonra, ana işlevde tanımlanan sonuçların veya nesneleri n hiçbirine IPython kabuğundan erişilemez. Daha iyi bir yol, ana kodun doğru dan modülün genel ad alanında (veya modülün de içe aktarılabilir olmasını istiyorsanız, `if __name__ == '__main__':` bloğunda) yürütülmesini sağlamaktır.

Bu şekilde, kodu% çalıştırığınızda, esas olarak tanımlanan tüm değişkenler e bakabilirisiniz. Bu, Jupyter not defterindeki hücrelerde en üst düzey değiş kenleri tanımlamaya eşdeğerdir.

Düz iç içe olmaktan daha iyidir

Derinlemesine iç içe geçmiş kod, bir soğanın birçok katmanını düşünmemi sağlıyor. Bir işlevi test ederken veya hatalarını ayıklarken, ilgilenilen koda ulaşmak için soğanın kaç katmanını geri soymanız gereklidir? "Düz iç içe geçmekt en daha iyidir" fikri Zen of Python'un bir parçasıdır ve genellikle etkileşimi kullanım için kod geliştirmek için de geçerlidir. İşlevleri ve sınıfları olabildiğince ayırtılabilir ve modüler yapmak, onları test etmeyi (birim testleri yazıyorsanız), hata ayıklamayı ve etkileşimli olarak kullanmayı kolaylaştırır.

Daha uzun dosyalar korkusunun üstesinden gelin

Java (veya başka bir dil) geçmişinden geliyorsanız, dosyaları kısa tutmanız söylemiş olabilir. Birçok dilde, bu sağlam bir tavsiyedir; uzun uzunluk genellikle kötü bir "kod kokusudur", bu da yeniden düzenleme veya yeniden düzelenin gerekli olabileceğini gösterir. Bununla birlikte, IPython kullanara k kod geliştirirken, 10 küçük ama birbirine bağlı dosyaya (örneğin, her bir i 100 satırın altında) çalışmak, genel olarak iki veya üç uzun dosyadan daha fazla baş ağrısına neden olabilir. Daha az dosya, düzenleme sırasında da yeniden yüklenerek daha az modül ve dosyalar arasında daha az atlama anlamına gelir. Her biri yüksek iç bağlılığı sahip daha büyük modüller korumayı çok daha kullanışlı ve Pythonic buldum. Bir çözüme doğru yinelendikten sonra, bazen daha büyük dosyaları daha küçük dosyalar halinde yeniden düzenlemek mantıklı olacaktır. Açıkçası, bu argümanı uç noktaya taşımayı desteklemiyorum, bu da tüm kodunuzu tek bir canavarca dosyaya koymak olabilir. Büyük bir kod tabanı için mantıklı ve sezgisel bir modül ve paket yapısı bulmak genellikle biraz çalışma gerektirir, ancak ekiplerde doğru bir şekilde yer almak özellikle önemlidir. Her modül dahili olarak uyumlu olmalı ve her bir işlevsellik alanında 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 yapılandırmaya girmenize neden olabilir.

Kendi Sınıflarınızı IPython Dostu Yapmak

IPython, incelediğiniz herhangi bir nesnenin konsol dostu bir dize temsilini görüntülemek için her türlü çabayı gösterir. Dicts, listeler ve tuples gibi birçok nesne için, güzel biçimlendirmeyi yapmak için yerleşik pprint modülü kullanılır. Bununla birlikte, kullanıcı tanımlı sınıflarda, istenen dizge çıktımasını kendiniz oluşturmanız gereklidir. Aşağıdaki basit sınıfı sahip olduğumuza varsayıyalım.

`class Message:`

localhost:8888/notebooks/More on the IPython System.ipynb

21/24

```
def __init__(self, msg):
    self.msg = msg
```

Bunu yazdıysanız, sınıfınız için varsayılan çıktıının pek hoş olmadığını keşf ettiğinizde hayal kırıklığına uğrayacaksınız.

In []:

```
x = Message('I have a secret')
```

```
x
<__main__.Message instance at 0x60ebbd8>
```

IPython, `__repr__` sihirli yöntemini tarafından döndürülen dizeyi alır (output = repr (obj) yaparak) ve bunu konsola yazdırır. Böylece, daha yararlı bir çıktı elde etmek için önceki sınıfa basit bir `__repr__` method ekleyebiliriz.

In []:

```
class Message:
    def __init__(self, msg):
        self.msg = msg

    def __repr__(self):
        return 'Message: %s' % self.msg
```

In []:

```
x = Message('I have a secret')
```

```
x
Message: I have a secret
```

Profiller ve Yapılandırma

IPython ve Jupyter ortamlarının görünümünüün çoğu yönü (renkler, komut istem i, satırlar arası boşluk vb.) Ve davranışını kapsamlı bir yapılandırma sistemi aracılığıyla yapılandırılabilir. Yapılandırma yoluyla yapabileceğiniz bazı şeyler şunlardır:

- Renk düzenini değiştirin
- Giriş ve çıkış istemlerinin nasıl göründüğünü değiştirin veya Çıkış'tan sonra ve sonraki Giriş isteminden önceki boş satırı kaldırın
- Rasgele 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 sihri gibi her zaman açık IPython uzantılarını etkinleştirin
- Jupyter uzantılarını etkinleştirme
- Kendi sihirlerinizi veya sistem takma adlarını tanımlayın

IPython kabuğunun yapılandırmaları, genellikle kullanıcı ana dizinizdeki .ipython / dizininde bulunan özel ipython_config.py dosyalarında belirtilir.

Yapılardırma, belirli bir profile göre gerçekleştirilir. IPython'u normal olarak başlattığınızda, varsayılan olarak profile_default dizininde depolanan varsayılan profili yüklersiniz. Dolayısıyla, Linux işletim sistemimde varsayılan IPython yapılandırma dosyamın tam yolu şudur:

/home/wesm/.ipython/profile_default/ipython_config.py

Bu dosyayı sisteminizde başlatmak için terminalde çalıştırın:
ipython profile create

Size bu dosyadakilerin kanlı ayırtlarını vereceğim. Neyse ki, her bir yapılandırma seçeneğinin ne için olduğunu açıklayan yorumları var, bu yüzden onu düzeltmek ve özelleştirmek için okuyucuya bırakacağım. Ek bir kullanışlı özellilik, birden çok profile sahip olmanın mümkün olmasıdır. Belirli bir uygulama veya proje için uyarlanmış alternatif bir IPython yapılandırmasına sahip olmak istedığınızı varsayıyalım. Yeni bir profil oluşturmak, aşağıdaki gibi bir şey yazmak 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
```

Her zaman olduğu gibi, çevrimiçi IPython belgeleri, profiller ve yapılandırıma hakkında daha fazla bilgi için mükemmel bir kaynaktır.

Jupyter için yapılandırma 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 düzeltmekten sonra, aşağıdaki gibi farklı bir dosya olarak yeniden adlandırabilirsiniz:

```
$ mv ~ / .jupyter / jupyter_notebook_config.py ~ / .jupyter / my_custom_config.py
```

Jupyter'i başlatırken, --config argümanını ekleyebilirsiniz:

```
jupyter notebook --config = ~ / .jupyter / my_custom_config.py
```