

PYTHON FOR DATA ANALYSIS

2. PYTHON LANGUAGE BASICS, IPYTHON AND JUPYTER NOTEBOOKS

2.1 The Python Interpreter

In [1]:

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

Hello World

In [2]:

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

5

In [1]:

```
%run hello_world.py
```

Hello World

2.2 IPython Basics

In [1]:

```
import numpy as np
```

In [305]:

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

In [6]:

```
data
```

Out[6]:

```
{0: -0.1331897267177855,  
 1: -0.14133540955511498,  
 2: 0.4948213668633935,  
 3: -0.34629746415472423,  
 4: 0.06210058761402506,  
 5: 0.9787172150432486,  
 6: 0.3344235434695492}
```

In [7]:

```
from numpy.random import randn
```

In [8]:

```
data = {i : randn() for i in range(7)}
```

In [9]:

```
print(data)
```

```
{0: 1.1134608088668008, 1: 0.29512766884912456, 2: 0.4394993394580315, 3: 1.5526680861347917, 4: -0.8872660967930334, 5: -1.0874895156980338, 6: 0.4120998443821892}
```

Sekme Tamamlama

Tanımlanan verilerle ilgili işlem yapılmak istendiğinde 'TAB' tuşuna basıldığında nesneleri, işlevleri vs.karşımıza getirir.

```
In []: an_apple = 27
```

```
In []: an_example = 42
```

```
In []: an 'TAB'
```

```
an_apple and an_example any
```

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

```
In []: b.'TAB'
```

```
b.append b.count b.insert b.reverse
```

```
b.clear b.extend b.pop b.sort
```

```
b.copy b.index b.remove
```

```
In []: import datetime
```

```
In []: datetime.'TAB'
```

```
datetime.datetime datetime.MINYEAR datetime.timezone
```

```
datetime.datetime_CAPI datetime.time datetime.tzinfo
```

In [36]:

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

In [37]:

```
b?
```

In [38]:

```
print?
```

In [9]:

```
def add_numbers(a, b):
    return a + b
```

In [292]:

```
add_numbers?
```

In [293]:

```
add_numbers??
```

In [42]:

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

In [294]:

```
def f(x, y, z):
    return (x + y) / z
a = 5
b = 6
c = 7.5
result = f(a, b, c)
```

In [2]:

```
%run ipython_script_test.py
```

```
%run komutu alistirma
```

In [45]:

```
c
```

Out[45]:

```
7.5
```

In [8]:

```
result
```

Out[8]:

```
1.4666666666666666
```

In []:

```
print('%run komutu alistirma')
```

In [15]:

```
def f(x, y, z):
    return (x + y) / z
a = 5
b = 6
c = 7.5
result = f(a, b, c)
```

In [18]:

```
x = 5
z = 4
y = 7
if x > 5:
    x += 1
    y = 8
```

Panodan Kod Kullanımı

%paste ve %cpaste fonksiyonları panodaki metni alır ve ilgili yere yapıştırır.

```
In []: %paste
Out []: x = 5
         y = 7
         if x > 5:
             x += 1
             y = 8
```

In [306]:

```
a = np.random.randn(100, 100)
%timeit np.dot(a, a)
```

137 µs ± 18.5 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)

In [307]:

```
%debug?
```

In [308]:

```
%pwd
```

Out[308]:

```
'C:\\\\Users\\\\Kader\\\\Desktop'
```

In [54]:

```
foo = %pwd
```

In [55]:

```
foo
```

Out[55]:

```
'C:\\\\Users\\\\Kader\\\\Desktop'
```

In [58]:

%hist

```
print('Hello World')
a=5
print(a)
%run hello_world.py
import numpy as np
data = {i : np.random.randn() for i in range(7)}
data
from numpy.random import randn
data = {i : randn() for i in range(7)}
print(data)
an_apple = 27
an_example = 42
an_apple
an_example
b = [1, 2, 3]
b.append
b.count
b.insert
b.reverse
b.clear
b.extend
b.pop
b.sort
b.copy
b.index
b.remove
import datetime
datetime.date
datetime.MAXYEAR
datetime.timedelta
datetime.datetime
datetime.MINYEAR
datetime.timezone
datetime.datetime_CAPI
datetime.time
datetime.tzinfo
b = [1, 2, 3]
b?
print?
def add_numbers(a, b):
    """
    Add two numbers together
    Returns
    -----
    the_sum : type of arguments
    """
    return a + b
add_numbers?
add_numbers??
np.*load*?
def f(x, y, z):
    return (x + y) / z
a = 5
b = 6
c = 7.5
result = f(a, b, c)
%run ipython_script_test.py
```

```
c
result
%load ipython_script_test.py
def f(x, y, z):
    return (x + y) / z
a = 5
b = 6
c = 7.5
result = f(a, b, c)
%paste
%cpaste
a = np.random.randn(100, 100)
%timeit np.dot(a, a)
10000 loops, best of 3: 20.9 µs per loop
%debug?
%pwd
foo = %pwd
foo
%quickref
%magic
%hist
```

In [309]:

```
%matplotlib
```

Using matplotlib backend: Qt5Agg

In [60]:

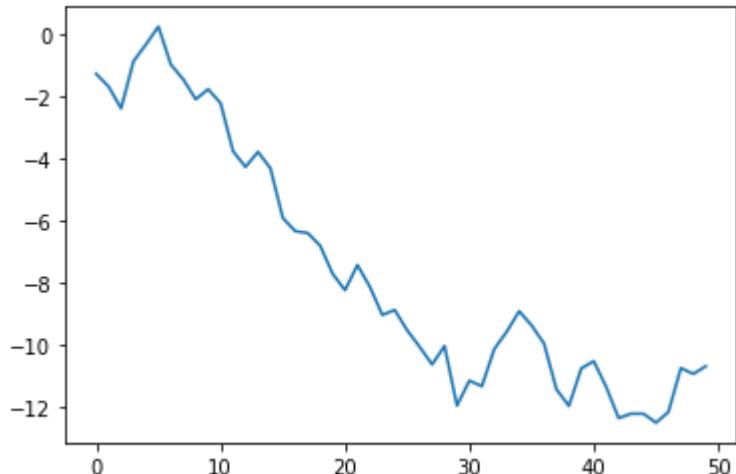
```
%matplotlib inline
```

In [61]:

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

Out[61]:

```
[<matplotlib.lines.Line2D at 0x2afb1d35f48>]
```



2.3 Python Language Basics

In [1]:

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

In [2]:

```
file_handle=[]
results = []
for line in file_handle:
    results.append(line.replace('foo', 'bar'))
```

In [3]:

```
print("Reached this line")
```

Reached this line

Fonksiyon ve Nesne Method Çağırımı

Parantez kullanarak ve argüman ileterek döndürülen değeri bir değişkene atayarak çağırabiliriz. Fonksiyonlar hem konumsal hem de anahtar kelime argümanlarını alabilir.

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

```
obj.some_method(x, y, z)
result = f(a, b, c, d=5, e='foo')
```

In [7]:

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

In [8]:

```
b = a
```

In [10]:

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

In [11]:

```
b
```

Out[11]:

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

In [316]:

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

In [317]:

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

In [318]:

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

In [319]:

```
data
```

Out[319]:

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

In [16]:

```
a = 5
```

In [17]:

```
type(a)
```

Out[17]:

```
int
```

In [18]:

```
a = 'foo'
```

In [19]:

```
type(a)
```

Out[19]:

```
str
```

In [20]:

```
'5' + 5
```

TypeError Traceback (most recent call last)
<ipython-input-20-4dd8efb5fac1> in <module>
----> 1 '5' + 5

TypeError: can only concatenate str (not "int") to str

In [21]:

```
a = 4.5
```

In [22]:

```
b = 2
```

In [320]:

```
print('a is {0}, b is {1}'.format(type(a), type(b)))
```

a is <class 'numpy.ndarray'>, b is <class 'int'>

In [24]:

```
a / b
```

Out[24]:

2.25

In [25]:

```
a = 5
```

In [26]:

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

Out[26]:

True

In [321]:

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

In [322]:

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

Out[322]:

True

In [29]:

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

Out[29]:

True

In [323]:

```
a = 'foo'
```

Niteliklere Erişme

Niteliklere 'tab' tuşuna basarak erişebiliriz;

```
a.'TAB'  
a.capitalize a.format a.isupper a.rindex a.strip  
a.center a.index a.join a.rjust a.swapcase  
a.count a.isalnum a.ljust a.rpartition a.title  
a.decode a.isalpha a.lower a.rsplit a.translate  
a.encode a.isdigit a.lstrip a.rstrip a.upper
```

```
a.endswith a.islower a.partition a.split a.zfill  
a.expandtabs a.isspace a.replace a.splitlines  
a.find a.istitle a.rfind a.startswith
```

In [33]:

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

Out[33]:

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

In [324]:

```
def isiterable(obj):  
    try:  
        iter(obj)  
        return True  
    except TypeError:  
        return False
```

In [325]:

```
isiterable('a string')
```

Out[325]:

```
True
```

In [326]:

```
isiterable([1, 2, 3])
```

Out[326]:

```
True
```

In [327]:

```
isiterable(5)
```

Out[327]:

```
False
```

Import İşlemi

Nesne bir listedir, eğer değilse onu liste olacak şekilde değiştiririz.

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

Aşağıdaki modüle sahip olduğumuzu varsayıyalım.

```
PI = 3.14159  
def f(x):  
    return x + 2  
def g(a, b):  
    return a + b
```

Tanımlanan değişkenlere ve işlevlere erişmek istiyorsak aynı dizinde başka bir dosya yapabiliriz.

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

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

'as' anahtarını kullanarak içe aktarmalara farklı değişken adları verebiliriz.

```
import some_module as sm  
from some_module import PI as pi, g as gf  
r1 = sm.f(pi)  
r2 = gf(6, pi)
```

In [53]:

```
5 - 7
```

Out[53]:

```
-2
```

In [54]:

```
12 + 21.5
```

Out[54]:

```
33.5
```

In [55]:

```
5 <= 2
```

Out[55]:

```
False
```

In [56]:

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

In [57]:

```
b = a
```

In [58]:

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

In [59]:

```
a is b
```

Out[59]:

True

In [60]:

```
a is not c
```

Out[60]:

True

In [61]:

```
a == c
```

Out[61]:

True

In [62]:

```
a is None
```

Out[62]:

False

In [1]:

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

In [2]:

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

In [3]:

```
a_list
```

Out[3]:

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

In [335]:

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

In [336]:

```
a_tuple[1] = 'four'
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-336-23fe12da1ba6> in <module>  
----> 1 a_tuple[1] = 'four'
```

TypeError: 'tuple' object does not support item assignment

In [337]:

```
ival = 17239871
```

In [7]:

```
ival ** 6
```

Out[7]:

```
26254519291092456596965462913230729701102721
```

In [8]:

```
fval = 7.243
```

In [9]:

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

In [10]:

```
3 / 2
```

Out[10]:

```
1.5
```

In [11]:

```
3 // 2
```

Out[11]:

```
1
```

In [12]:

```
a = 'one way of writing a string'  
b = "another way"  
c = """  
This is a longer string that  
spans multiple lines  
"""
```

In [13]:

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

Out[13]:

```
3
```

In [14]:

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

In [15]:

```
a[10] = 'f'
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-15-2151a30ed055> in <module>  
----> 1 a[10] = 'f'
```

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

In [16]:

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

In [17]:

```
b
```

Out[17]:

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

In [18]:

```
a
```

Out[18]:

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

In [19]:

```
a = 5.6
```

In [20]:

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

In [21]:

```
print(s)
```

```
5.6
```

In [22]:

```
s = 'python'
```

In [23]:

```
list(s)
```

Out[23]:

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

In [24]:

```
s[:3]
```

Out[24]:

```
'pyt'
```

In [25]:

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

In [27]:

```
print(s)
```

```
12\34
```

In [28]:

```
s = r'this\has\no\special\characters'
```

In [29]:

```
s
```

Out[29]:

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

In [30]:

```
a = 'this is the first half'
```

In [31]:

```
b = 'and this is the second half'
```

In [32]:

```
a + b
```

Out[32]:

```
'this is the first halfand this is the second half'
```

In [33]:

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

In [34]:

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

Out[34]:

```
'4.56 Argentine Pesos are worth US$1'
```

In [35]:

```
val = "español"
```

In [36]:

```
val
```

Out[36]:

```
'español'
```

In [37]:

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

In [38]:

```
val_utf8
```

Out[38]:

```
b'esp\u00e1\xc3\xb1ol'
```

In [39]:

```
type(val_utf8)
```

Out[39]:

```
bytes
```

In [40]:

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

Out[40]:

```
'espa\u00f1ol'
```

In [41]:

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

Out[41]:

```
b'esp\u00e1\xf1ol'
```

In [42]:

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

Out[42]:

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

In [43]:

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

Out[43]:

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

In [44]:

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

In [45]:

```
bytes_val
```

Out[45]:

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

In [46]:

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

In [47]:

```
decoded
```

Out[47]:

```
'this is bytes'
```

In [48]:

```
True and True
```

Out[48]:

```
True
```

In [49]:

```
False or True
```

Out[49]:

```
True
```

In [50]:

```
s = '3.14159'
```

In [51]:

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

In [52]:

```
type(fval)
```

Out[52]:

float

In [53]:

```
int(fval)
```

Out[53]:

3

In [54]:

```
bool(fval)
```

Out[54]:

True

In [55]:

```
bool(0)
```

Out[55]:

False

In [56]:

```
a = None
```

In [57]:

```
a is None
```

Out[57]:

True

In [58]:

```
b = 5
```

In [59]:

```
b is not None
```

Out[59]:

True

In [62]:

```
def add_and_maybe_multiply(a, b, c=None):
    result = a + b
    if c is not None:
        result = result * c
    return result
```

In [63]:

```
type(None)
```

Out[63]:

NoneType

In [64]:

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

In [65]:

```
dt = datetime(2011, 10, 29, 20, 30, 21)
```

In [66]:

```
dt.day
```

Out[66]:

29

In [67]:

```
dt.minute
```

Out[67]:

30

In [68]:

```
dt.date()
```

Out[68]:

```
datetime.date(2011, 10, 29)
```

In [69]:

```
dt.time()
```

Out[69]:

```
datetime.time(20, 30, 21)
```

In [70]:

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

Out[70]:

```
'10/29/2011 20:30'
```

In [71]:

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

Out[71]:

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

In [72]:

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

Out[72]:

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

In [73]:

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

In [74]:

```
delta = dt2 - dt
```

In [75]:

```
delta
```

Out[75]:

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

In [76]:

```
type(delta)
```

Out[76]:

```
datetime.timedelta
```

In [77]:

```
dt
```

Out[77]:

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

In [78]:

```
dt + delta
```

Out[78]:

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

In [92]:

```
x=10
```

In [93]:

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

In [94]:

```
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')
```

Positive and larger than or equal to 5

In [95]:

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

In [96]:

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

In [99]:

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

Made it

In [100]:

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

Out[100]:

True

In [30]:

```
collection = []
for value in collection:
    sequence = [1, 2, None, 4, None, 5]
    total = 0
    for value in sequence:
        if value is None:
            continue
        total += value
    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
```

In [31]:

```
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)
```

In [32]:

```
iterator=[]
for a, b, c in iterator:
    x = 256
    total = 0
    while x > 0:
        if total > 500:
            break
        total += x
        x = x // 2
```

In [33]:

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

```
positive!
```

In [114]:

```
range(10)
```

Out[114]:

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

In [115]:

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

Out[115]:

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

In [116]:

```
list(range(0, 20, 2))
```

Out[116]:

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

In [117]:

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

Out[117]:

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

In [118]:

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

In [119]:

```
sum = 0
for i in range(100000):
    if i % 3 == 0 or i % 5 == 0:
        sum += i
```

In [120]:

```
x = 5
```

In [121]:

```
'Non-negative' if x >= 0 else 'Negative'
```

Out[121]:

```
'Non-negative'
```

3. BUILT-IN DATA STRUCTURES, FUNCTIONS AND FILES

3.1 Data Structures and Sequences

In [129]:

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

In [130]:

```
tup
```

Out[130]:

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

In [131]:

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

In [132]:

```
nested_tup
```

Out[132]:

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

In [133]:

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

Out[133]:

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

In [134]:

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

In [135]:

```
tup
```

Out[135]:

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

In [136]:

```
tup[0]
```

Out[136]:

```
's'
```

In [137]:

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

In [138]:

```
tup[2] = False
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-138-b89d0c4ae599> in <module>  
----> 1 tup[2] = False
```

TypeError: 'tuple' object does not support item assignment

In [139]:

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

In [140]:

```
tup
```

Out[140]:

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

In [141]:

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

Out[141]:

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

In [142]:

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

Out[142]:

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

In [143]:

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

In [144]:

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

In [145]:

```
b
```

Out[145]:

```
5
```

In [147]:

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

In [148]:

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

In [149]:

```
d
```

Out[149]:

```
7
```

In [150]:

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

In [151]:

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

In [152]:

```
a
```

Out[152]:

```
1
```

In [153]:

```
b
```

Out[153]:

```
2
```

In [154]:

```
b, a = a, b
```

In [155]:

```
a
```

Out[155]:

```
2
```

In [156]:

```
b
```

Out[156]:

```
1
```

In [157]:

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

In [158]:

```
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
```

In [159]:

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

In [160]:

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

In [161]:

```
a, b
```

Out[161]:

```
(1, 2)
```

In [162]:

```
rest
```

Out[162]:

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

In [163]:

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

In [164]:

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

In [165]:

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

Out[165]:

```
4
```

In [166]:

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

In [7]:

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

In [8]:

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

In [9]:

```
b_list
```

Out[9]:

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

In [10]:

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

In [11]:

```
b_list
```

Out[11]:

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

In [12]:

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

In [13]:

```
gen
```

Out[13]:

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

In [14]:

```
list(gen)
```

Out[14]:

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

In [15]:

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

In [16]:

```
b_list
```

Out[16]:

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

In [17]:

```
b_list.insert(1, 'red')
```

In [18]:

```
b_list
```

Out[18]:

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

In [19]:

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

Out[19]:

```
'peekaboo'
```

In [20]:

```
b_list
```

Out[20]:

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

In [21]:

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

In [22]:

```
b_list
```

Out[22]:

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

In [23]:

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

In [24]:

```
b_list
```

Out[24]:

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

In [25]:

```
'dwarf' in b_list
```

Out[25]:

```
True
```

In [26]:

```
'dwarf' not in b_list
```

Out[26]:

```
False
```

In [27]:

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

Out[27]:

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

In [28]:

```
x = [4, None, 'foo']
```

In [29]:

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

In [30]:

```
x
```

Out[30]:

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

In [34]:

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

In [35]:

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

In [34]:

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

In [35]:

```
a.sort()
```

In [36]:

```
a
```

Out[36]:

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

In [37]:

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

In [38]:

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

In [39]:

```
b
```

Out[39]:

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

In [40]:

```
import bisect
```

In [41]:

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

In [42]:

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

Out[42]:

```
4
```

In [43]:

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

Out[43]:

```
6
```

In [44]:

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

In [45]:

```
c
```

Out[45]:

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

In [46]:

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

In [47]:

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

Out[47]:

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

In [48]:

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

In [49]:

```
seq
```

Out[49]:

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

In [50]:

```
seq[:5]
```

Out[50]:

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

In [51]:

```
seq[3:]
```

Out[51]:

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

In [52]:

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

Out[52]:

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

In [53]:

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

Out[53]:

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

In [54]:

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

Out[54]:

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

In [55]:

```
seq[::-1]
```

Out[55]:

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

YERLEŞİK SIRA İŞLEVLERİNİ NUMARALANDIRMAK

Bir dizi üzerinde yineleme yaparken dizinin kaydını tutmak gerekebilir. Numaralandırılarak bu işlev gerçekleştirilebilir.

```
i = 0
for value in collection:
    i += 1

for i, value in enumerate(collection):
```

In [60]:

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

In [61]:

```
mapping = {}
```

In [63]:

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

In [64]:

```
mapping
```

Out[64]:

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

In [65]:

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

Out[65]:

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

In [66]:

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

Out[66]:

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

In [67]:

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

In [68]:

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

In [69]:

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

In [70]:

```
list(zipped)
```

Out[70]:

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

In [71]:

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

In [72]:

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

Out[72]:

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

In [73]:

```
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
```

In [74]:

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

In [75]:

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

In [76]:

```
first_names
```

Out[76]:

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

In [77]:

```
last_names
```

Out[77]:

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

In [78]:

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

Out[78]:

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

In [79]:

```
empty_dict={}
```

In [80]:

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

In [81]:

```
d1
```

Out[81]:

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

In [82]:

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

In [83]:

```
d1
```

Out[83]:

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

In [84]:

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

Out[84]:

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

In [85]:

```
'b' in d1
```

Out[85]:

True

In [86]:

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

In [87]:

```
d1
```

Out[87]:

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

In [88]:

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

In [89]:

```
d1
```

Out[89]:

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

In [90]:

```
del d1[5]
```

In [91]:

```
d1
```

Out[91]:

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

In [92]:

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

In [93]:

```
ret
```

Out[93]:

```
'another value'
```

In [94]:

```
d1
```

Out[94]:

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

In [95]:

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

Out[95]:

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

In [96]:

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

Out[96]:

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

In [97]:

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

In [98]:

```
d1
```

Out[98]:

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

In [45]:

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

In [46]:

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

In [47]:

```
mapping
```

Out[47]:

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

Dizilerden Dikteler Oluşturmak

```
if key in some_dict:  
    value = some_dict[key].  
else:  
    value = default_value  
  
value = some_dict.get(key, default_value)
```

In [104]:

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

In [105]:

```
by_letter = {}
```

In [106]:

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

In [107]:

```
by_letter
```

Out[107]:

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

In [108]:

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

In [109]:

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

In [110]:

```
hash('string')
```

Out[110]:

```
-7258225898308840645
```

In [111]:

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

Out[111]:

```
1097636502276347782
```

In [112]:

```
hash((1, 2, [2, 3]))
```

TypeError

Traceback (most recent call last)

```
<ipython-input-112-8ffc25aff872> in <module>
```

```
----> 1 hash((1, 2, [2, 3]))
```

TypeError: unhashable type: 'list'

In [113]:

```
d = {}
```

In [114]:

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

In [115]:

```
d
```

Out[115]:

```
{(1, 2, 3): 5}
```

In [116]:

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

Out[116]:

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

In [117]:

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

Out[117]:

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

In [118]:

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

In [119]:

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

In [120]:

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

Out[120]:

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

In [121]:

```
a | b
```

Out[121]:

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

In [122]:

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

Out[122]:

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

In [123]:

```
a & b
```

Out[123]:

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

In [124]:

```
c = a.copy()
```

In [125]:

```
c |= b
```

In [126]:

```
c
```

Out[126]:

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

In [127]:

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

In [128]:

```
d &lt;= b
```

In [129]:

```
d
```

Out[129]:

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

In [130]:

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

In [131]:

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

In [132]:

```
my_set
```

Out[132]:

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

In [133]:

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

In [134]:

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

Out[134]:

```
True
```

In [135]:

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

Out[135]:

```
True
```

In [136]:

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

Out[136]:

```
True
```

In [139]:

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

In [140]:

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

Out[140]:

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

In [141]:

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

In [142]:

```
unique_lengths
```

Out[142]:

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

In [143]:

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

Out[143]:

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

In [144]:

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

In [145]:

```
loc_mapping
```

Out[145]:

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

In [146]:

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

In [147]:

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

In [148]:

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

In [149]:

```
result
```

Out[149]:

```
['Steven']
```

In [150]:

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

In [151]:

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

In [152]:

```
flattened
```

Out[152]:

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

In [153]:

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

In [154]:

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

Out[154]:

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

3.2 Functions

In [155]:

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

In [156]:

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

Out[156]:

```
45.0
```

In [157]:

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

Out[157]:

77

In [158]:

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

In [159]:

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

In [160]:

```
a = None
```

In [161]:

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

In [162]:

```
print(a)
```

[]

In [163]:

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

In [164]:

```
return_value = f()
```

In [165]:

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

In [166]:

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

In [167]:

```
import re
```

In [168]:

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

In [169]:

```
clean_strings(states)
```

Out[169]:

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

In [170]:

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

In [171]:

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

In [175]:

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

In [176]:

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

Out[176]:

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

In [177]:

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

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

In [178]:

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

In [179]:

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

In [180]:

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

In [181]:

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

Out[181]:

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

In [182]:

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

In [183]:

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

In [184]:

```
strings
```

Out[184]:

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

In [185]:

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

In [186]:

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

In [187]:

```
from functools import partial
add_five = partial(add_numbers, 5)
```

In [188]:

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

In [189]:

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

```
a  
b  
c
```

In [190]:

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

In [191]:

```
dict_iterator
```

Out[191]:

```
<dict_keyiterator at 0x11cec41c9a8>
```

In [192]:

```
list(dict_iterator)
```

Out[192]:

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

In [193]:

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

In [194]:

```
gen = squares()
```

In [195]:

```
gen
```

Out[195]:

```
<generator object squares at 0x0000011CEC410948>
```

In [196]:

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

```
Generating squares from 1 to 100
```

```
1 4 9 16 25 36 49 64 81 100
```

In [197]:

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

In [198]:

```
gen
```

Out[198]:

```
<generator object <genexpr> at 0x0000011CEC410DC8>
```

In [199]:

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

In [200]:

```
gen = _make_gen()
```

In [201]:

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

Out[201]:

```
328350
```

In [202]:

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

Out[202]:

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

In [203]:

```
import itertools
```

In [204]:

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

In [205]:

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

In [206]:

```
for letter, names in itertools.groupby(names, first_letter):
    print(letter, list(names))
```

```
A ['Alan', 'Adam']
```

```
W ['Wes', 'Will']
```

```
A ['Albert']
```

```
S ['Steven']
```

In [207]:

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

Out[207]:

```
1.2345
```

In [208]:

```
float('something')
```

```
-----  
ValueError
```

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

```
<ipython-input-208-2649e4ade0e6> in <module>
```

```
----> 1 float('something')
```

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

In [209]:

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

In [210]:

```
attempt_float('1.2345')
```

Out[210]:

1.2345

In [211]:

```
attempt_float('something')
```

Out[211]:

'something'

In [212]:

```
float((1, 2))
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-212-82f777b0e564> in <module>  
----> 1 float((1, 2))
```

TypeError: float() argument must be a string or a number, not 'tuple'

In [213]:

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

In [214]:

```
attempt_float((1, 2))
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-214-8b0026e9e6b7> in <module>  
----> 1 attempt_float((1, 2))  
  
<ipython-input-213-6209ddec2b5> 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'

Hatalar

Type Error dizi veya sayısal değer girişi olmayan durumlarda hata verebilir. Bu kodla bir çok istisna sağlayabiliriz:

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

Bu istisnayı önlemeden kod çalıştırırmak istersek:

```
f = open(path, 'w')  
try:  
    write_to_file(f)  
finally:  
    f.close()
```

Burada f hep kapalı olmalı.

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

In [222]:

%run examples/ipython_bug.py

```
-----
OSError                                                 Traceback (most recent call last)
~\anaconda3\lib\site-packages\IPython\core\magics\execution.py in run(self,
parameter_s, runner, file_finder)
    696         fpath = arg_lst[0]
--> 697         filename = file_finder(fpath)
    698     except IndexError:
~\anaconda3\lib\site-packages\IPython\utils\path.py in get_py_filename(name,
force_win32)
    108     else:
--> 109         raise IOError('File `%r` not found.' % name)
    110

```

OSError: File `examples/ipython_bug.py` not found.

During handling of the above exception, another exception occurred:

```
Exception                                                 Traceback (most recent call last)
<ipython-input-222-ac2d87c3d5c0> in <module>
----> 1 get_ipython().run_line_magic('run', 'examples/ipython_bug.py')

~\anaconda3\lib\site-packages\IPython\core\interactiveshell.py in run_line_m
agic(self, magic_name, line, _stack_depth)
    2324         kwargs['local_ns'] = sys._getframe(stack_depth).f_lo
cals
    2325         with self.builtin_trap:
--> 2326             result = fn(*args, **kwargs)
    2327         return result
    2328

<decorator-gen-60> in run(self, parameter_s, runner, file_finder)

~\anaconda3\lib\site-packages\IPython\core\magic.py in <lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

~\anaconda3\lib\site-packages\IPython\core\magics\execution.py in run(self,
parameter_s, runner, file_finder)
    706         if os.name == 'nt' and re.match(r"^\.*'$", fpath):
    707             warn('For Windows, use double quotes to wrap a filen
ame: %run "mypath\\myfile.py"')
--> 708             raise Exception(msg)
    709         except TypeError:
    710             if fpath in sys.meta_path:
```

Exception: File `examples/ipython_bug.py` not found.

3.3 Files and the Operating System

In [248]:

```
path = 'kaderdeneme1.txt'
```

In [249]:

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

In [250]:

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

In [251]:

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

In [252]:

```
lines
```

Out[252]:

```
['dosya yolu deneme kader']
```

In [253]:

```
f.close()
```

In [254]:

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

In [255]:

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

In [256]:

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

Out[256]:

```
'dosya yolu'
```

In [257]:

```
f2 = open(path, 'rb')
```

In [258]:

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

Out[258]:

```
b'dosya yolu'
```

In [259]:

```
f.tell()
```

Out[259]:

10

In [260]:

```
f2.tell()
```

Out[260]:

10

In [261]:

```
import sys
```

In [262]:

```
sys.getdefaultencoding()
```

Out[262]:

'utf-8'

In [263]:

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

Out[263]:

3

In [264]:

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

Out[264]:

'y'

In [265]:

```
f.close()
```

In [266]:

```
f2.close()
```

In [267]:

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

In [268]:

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

In [269]:

```
lines
```

Out[269]:

```
['dosya yolu deneme kader']
```

In [270]:

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

In [271]:

```
chars
```

Out[271]:

```
'dosya yolu'
```

In [272]:

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

In [273]:

```
data
```

Out[273]:

```
b'dosya yolu'
```

In [274]:

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

Out[274]:

```
'dosya yolu'
```

In [275]:

```
data[:4].decode('utf8')
```

Out[275]:

```
'dosy'
```

In [276]:

```
sink_path = 'sink.txt'
```

In [277]:

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

In [278]:

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

dosya yolu

In [279]:

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

In [280]:

```
f.read(5)
```

Out[280]:

'dosya'

In [281]:

```
f.seek(4)
```

Out[281]:

4

In [282]:

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

Out[282]:

'a'

In [283]:

```
f.close()
```

NUMPY BASICS: ARRAYS AND VECTORIZED COMPUTATION

In [28]:

```
import numpy as np
```

In [2]:

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

In [3]:

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

In [6]:

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

Wall time: 35 ms

In [8]:

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

Wall time: 1.4 s

4.1 The NumPy ndarray: A Multidimensional Array Object

In [9]:

```
import numpy as np
```

In [10]:

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

In [11]:

```
data
```

Out[11]:

```
array([[-0.97861289, -0.93533878,  1.78805848],  
      [ 0.14086001, -0.02697179, -2.02341481]])
```

In [12]:

```
data * 10
```

Out[12]:

```
array([[ -9.78612888,  -9.35338781,  17.88058477],  
      [ 1.40860008,  -0.26971786, -20.2341481 ]])
```

In [13]:

```
data + data
```

Out[13]:

```
array([[-1.95722578, -1.87067756,  3.57611695],  
      [ 0.28172002, -0.05394357, -4.04682962]])
```

In [14]:

```
data.shape
```

Out[14]:

```
(2, 3)
```

In [15]:

```
data.dtype
```

Out[15]:

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

In [16]:

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

In [17]:

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

In [18]:

```
arr1
```

Out[18]:

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

In [19]:

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

In [20]:

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

In [21]:

```
arr2
```

Out[21]:

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

In [22]:

```
arr2.ndim
```

Out[22]:

```
2
```

In [23]:

```
arr2.shape
```

Out[23]:

```
(2, 4)
```

In [24]:

```
arr1.dtype
```

Out[24]:

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

In [25]:

```
arr2.dtype
```

Out[25]:

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

In [26]:

```
np.zeros(10)
```

Out[26]:

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

In [27]:

```
np.zeros((3, 6))
```

Out[27]:

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

In [28]:

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

Out[28]:

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

In [29]:

```
np.arange(15)
```

Out[29]:

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

In [30]:

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

In [31]:

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

In [32]:

```
arr1.dtype
```

Out[32]:

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

In [33]:

```
arr2.dtype
```

Out[33]:

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

In [34]:

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

In [35]:

```
arr.dtype
```

Out[35]:

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

In [36]:

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

In [37]:

```
float_arr.dtype
```

Out[37]:

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

In [38]:

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

In [39]:

```
arr
```

Out[39]:

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

In [40]:

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

Out[40]:

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

In [41]:

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

In [42]:

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

Out[42]:

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

In [43]:

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

In [44]:

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

In [45]:

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

Out[45]:

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

In [46]:

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

In [47]:

```
empty_uint32
```

Out[47]:

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

In [48]:

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

In [49]:

```
arr
```

Out[49]:

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

In [50]:

```
arr * arr
```

Out[50]:

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

In [51]:

```
arr - arr
```

Out[51]:

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

In [52]:

```
1 / arr
```

Out[52]:

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

In [53]:

```
arr ** 0.5
```

Out[53]:

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

In [54]:

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

In [55]:

```
arr2
```

Out[55]:

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

In [56]:

```
arr2 > arr
```

Out[56]:

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

In [57]:

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

In [58]:

```
arr
```

Out[58]:

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

In [59]:

```
arr[5]
```

Out[59]:

```
5
```

In [60]:

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

Out[60]:

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

In [61]:

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

In [62]:

```
arr
```

Out[62]:

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

In [63]:

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

In [65]:

```
arr_slice
```

Out[65]:

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

In [66]:

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

In [67]:

```
arr
```

Out[67]:

```
array([ 0, 1, 2, 3, 4, 12, 12345, 12, 8, 9])
```

In [68]:

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

In [69]:

```
arr
```

Out[69]:

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

In [70]:

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

In [71]:

```
arr2d[2]
```

Out[71]:

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

In [72]:

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

Out[72]:

```
3
```

In [73]:

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

Out[73]:

```
3
```

In [74]:

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

In [75]:

```
arr3d
```

Out[75]:

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

In [76]:

```
arr3d[0]
```

Out[76]:

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

In [77]:

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

In [78]:

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

In [79]:

```
arr3d
```

Out[79]:

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

In [80]:

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

In [81]:

```
arr3d
```

Out[81]:

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

In [82]:

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

Out[82]:

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

In [83]:

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

In [84]:

```
x
```

Out[84]:

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

In [85]:

```
x[0]
```

Out[85]:

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

In [86]:

```
arr
```

Out[86]:

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

In [87]:

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

Out[87]:

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

In [88]:

```
arr2d
```

Out[88]:

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

In [89]:

```
arr2d[:2]
```

Out[89]:

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

In [90]:

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

Out[90]:

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

In [91]:

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

Out[91]:

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

In [92]:

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

Out[92]:

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

In [93]:

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

Out[93]:

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

In [94]:

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

In [95]:

```
arr2d
```

Out[95]:

```
array([[1, 0, 0],  
       [4, 0, 0],  
       [7, 8, 9]])
```

In [96]:

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

In [97]:

```
data = np.random.randn(7, 4)
```

In [98]:

```
names
```

Out[98]:

```
array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')
```

In [99]:

```
data
```

Out[99]:

```
array([[ 1.13237913,  0.83300502,  0.64302329, -0.16080444],
       [-0.64261124, -1.31234269, -0.61376166, -0.48223236],
       [-0.0570716 ,  2.35276411, -1.4489522 , -1.59448077],
       [-0.67657178,  0.27836821,  0.26087646,  0.17378025],
       [-0.13258782,  0.4367635 ,  2.42143324,  0.34685835],
       [-1.54374379, -1.28385591,  0.27688796, -0.98617892],
       [ 0.55816089, -0.27486899, -0.07003328, -0.10818291]])
```

In [100]:

```
names == 'Bob'
```

Out[100]:

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

In [101]:

```
data[names == 'Bob']
```

Out[101]:

```
array([[ 1.13237913,  0.83300502,  0.64302329, -0.16080444],
       [-0.67657178,  0.27836821,  0.26087646,  0.17378025]])
```

In [102]:

```
data[names == 'Bob', 2:]
```

Out[102]:

```
array([[ 0.64302329, -0.16080444],
       [ 0.26087646,  0.17378025]])
```

In [103]:

```
data[names == 'Bob', 3]
```

Out[103]:

```
array([-0.16080444,  0.17378025])
```

In [104]:

```
names != 'Bob'
```

Out[104]:

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

In [105]:

```
data[~(names == 'Bob')]
```

Out[105]:

```
array([[ -0.64261124, -1.31234269, -0.61376166, -0.48223236],
       [-0.0570716 ,  2.35276411, -1.4489522 , -1.59448077],
       [-0.13258782,  0.4367635 ,  2.42143324,  0.34685835],
       [-1.54374379, -1.28385591,  0.27688796, -0.98617892],
       [ 0.55816089, -0.27486899, -0.07003328, -0.10818291]])
```

In [106]:

```
cond = names == 'Bob'
```

In [107]:

```
data[~cond]
```

Out[107]:

```
array([[ -0.64261124, -1.31234269, -0.61376166, -0.48223236],
       [-0.0570716 ,  2.35276411, -1.4489522 , -1.59448077],
       [-0.13258782,  0.4367635 ,  2.42143324,  0.34685835],
       [-1.54374379, -1.28385591,  0.27688796, -0.98617892],
       [ 0.55816089, -0.27486899, -0.07003328, -0.10818291]])
```

In [108]:

```
mask = (names == 'Bob') | (names == 'Will')
```

In [109]:

```
mask
```

Out[109]:

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

In [110]:

```
data[mask]
```

Out[110]:

```
array([[ 1.13237913,  0.83300502,  0.64302329, -0.16080444],
       [-0.0570716 ,  2.35276411, -1.4489522 , -1.59448077],
       [-0.67657178,  0.27836821,  0.26087646,  0.17378025],
       [-0.13258782,  0.4367635 ,  2.42143324,  0.34685835]])
```

In [111]:

```
data[data < 0] = 0
```

In [112]:

```
data
```

Out[112]:

```
array([[1.13237913, 0.83300502, 0.64302329, 0.          ],
       [0.          , 0.          , 0.          , 0.          ],
       [0.          , 2.35276411, 0.          , 0.          ],
       [0.          , 0.27836821, 0.26087646, 0.17378025],
       [0.          , 0.4367635 , 2.42143324, 0.34685835],
       [0.          , 0.          , 0.27688796, 0.          ],
       [0.55816089, 0.          , 0.          , 0.          ]])
```

In [113]:

```
data[names != 'Joe'] = 7
```

In [114]:

```
data
```

Out[114]:

```
array([[7.          , 7.          , 7.          , 7.          , 7.          ],
       [0.          , 0.          , 0.          , 0.          , 0.          ],
       [7.          , 7.          , 7.          , 7.          , 7.          ],
       [7.          , 7.          , 7.          , 7.          , 7.          ],
       [7.          , 7.          , 7.          , 7.          , 7.          ],
       [0.          , 0.          , 0.27688796, 0.          , 0.          ],
       [0.55816089, 0.          , 0.          , 0.          , 0.          ]])
```

In [115]:

```
arr = np.empty((8, 4))
```

In [116]:

```
for i in range(8):
    arr[i] = i
```

In [117]:

```
arr
```

Out[117]:

```
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.]])
```

In [118]:

```
arr[[4, 3, 0, 6]]
```

Out[118]:

```
array([[4., 4., 4., 4.],
       [3., 3., 3., 3.],
       [0., 0., 0., 0.],
       [6., 6., 6., 6.]])
```

In [119]:

```
arr[[-3, -5, -7]]
```

Out[119]:

```
array([[5., 5., 5., 5.],
       [3., 3., 3., 3.],
       [1., 1., 1., 1.]])
```

In [120]:

```
arr = np.arange(32).reshape((8, 4))
```

In [121]:

```
arr
```

Out[121]:

```
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 [122]:

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

Out[122]:

```
array([ 4, 23, 29, 10])
```

In [123]:

```
arr[[1, 5, 7, 2]][[:, [0, 3, 1, 2]]]
```

Out[123]:

```
array([[ 4,  7,  5,  6],
       [20, 23, 21, 22],
       [28, 31, 29, 30],
       [ 8, 11,  9, 10]])
```

In [124]:

```
arr = np.arange(15).reshape((3, 5))
```

In [125]:

```
arr
```

Out[125]:

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

In [126]:

```
arr.T
```

Out[126]:

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

In [127]:

```
arr = np.random.randn(6, 3)
```

In [128]:

```
arr
```

Out[128]:

```
array([[-1.55912159, -1.13802867, -0.62186519],
       [ 0.99318756, -0.49710144,  1.01620262],
       [ 0.77704373, -0.58281065,  0.69900966],
       [ 1.24949894, -0.15882918,  0.90204217],
       [-0.40623551,  1.84269389,  0.13212215],
       [ 1.61040917, -0.60176701,  0.36701547]])
```

In [129]:

```
np.dot(arr.T, arr)
```

Out[129]:

```
array([[ 8.34077121, -1.08837495,  4.18647741],
       [-1.08837495,  5.66475836, -0.32551339],
       [ 4.18647741, -0.32551339,  2.87383527]])
```

In [130]:

```
arr = np.arange(16).reshape((2, 2, 4))
```

In [131]:

```
arr
```

Out[131]:

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

In [132]:

```
arr.transpose((1, 0, 2))
```

Out[132]:

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

In [133]:

```
arr
```

Out[133]:

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

In [134]:

```
arr.swapaxes(1, 2)
```

Out[134]:

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

4.2 Universal Functions: Fast Element-Wise Array Functions

In [135]:

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

In [136]:

```
arr
```

Out[136]:

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

In [137]:

```
np.sqrt(arr)
```

Out[137]:

```
array([0.          , 1.          , 1.41421356, 1.73205081, 2.          ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.          ])
```

In [138]:

```
np.exp(arr)
```

Out[138]:

```
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])
```

In [139]:

```
x = np.random.randn(8)
```

In [140]:

```
y = np.random.randn(8)
```

In [141]:

```
x
```

Out[141]:

```
array([-0.43444214,  0.41248202,  0.15946119,  0.56168226, -0.66345105,
       0.37719352, -1.04059444,  0.10512523])
```

In [142]:

```
y
```

Out[142]:

```
array([ 1.41806405, -1.43866355,  0.47241206,  0.90471771,  0.00820892,
       0.02832823, -1.46247601,  0.89287376])
```

In [143]:

```
np.maximum(x, y)
```

Out[143]:

```
array([ 1.41806405,  0.41248202,  0.47241206,  0.90471771,  0.00820892,
       0.37719352, -1.04059444,  0.89287376])
```

In [144]:

```
arr = np.random.randn(7) * 5
```

In [145]:

```
arr
```

Out[145]:

```
array([-9.51184353, 11.93795455, -4.27845218, 2.60367081, -8.26201352,
       -5.35439491, -0.57578805])
```

In [146]:

```
remainder, whole_part = np.modf(arr)
```

In [147]:

```
remainder
```

Out[147]:

```
array([-0.51184353, 0.93795455, -0.27845218, 0.60367081, -0.26201352,
       -0.35439491, -0.57578805])
```

In [148]:

```
whole_part
```

Out[148]:

```
array([-9., 11., -4., 2., -8., -5., -0.])
```

In [149]:

```
arr
```

Out[149]:

```
array([-9.51184353, 11.93795455, -4.27845218, 2.60367081, -8.26201352,
       -5.35439491, -0.57578805])
```

In [150]:

```
np.sqrt(arr)
```

...

In [151]:

```
np.sqrt(arr, arr)
```

...

In [152]:

```
arr
```

Out[152]:

```
array([      nan,  3.45513452,          nan,  1.61358942,          nan,
       nan,          nan])
```

4.3 Array-Oriented Programming with Arrays

In [153]:

```
points = np.arange(-5, 5, 0.01)
```

4.4 File Input and Output with Arrays

In [154]:

```
xs, ys = np.meshgrid(points, points)
```

In [155]:

```
ys
```

Out[155]:

```
array([[ -5. , -5. , -5. , ..., -5. , -5. , -5. ],
       [-4.99, -4.99, -4.99, ..., -4.99, -4.99, -4.99],
       [-4.98, -4.98, -4.98, ..., -4.98, -4.98, -4.98],
       ...,
       [ 4.97,  4.97,  4.97, ...,  4.97,  4.97,  4.97],
       [ 4.98,  4.98,  4.98, ...,  4.98,  4.98,  4.98],
       [ 4.99,  4.99,  4.99, ...,  4.99,  4.99,  4.99]])
```

In [156]:

```
z = np.sqrt(xs ** 2 + ys ** 2)
```

In [157]:

```
z
```

Out[157]:

```
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]])
```

In [158]:

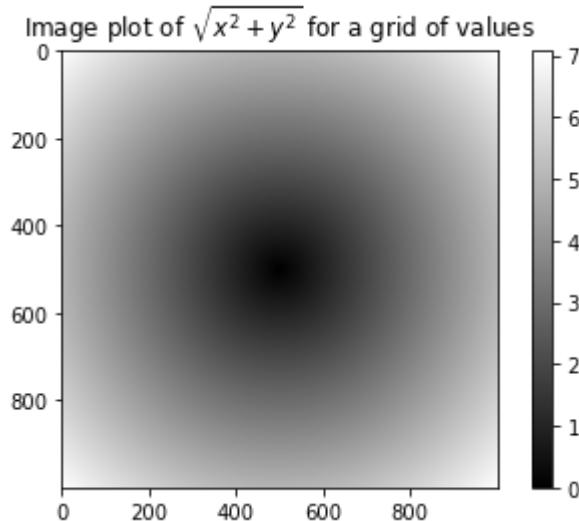
```
import matplotlib.pyplot as plt
```

In [161]:

```
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[161]:

```
Text(0.5, 1.0, 'Image plot of  $\sqrt{x^2 + y^2}$  for a grid of values')
```



In [162]:

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

In [163]:

```
yarr = np.array([2.1, 2.2, 2.3, 2.4, 2.5])
```

In [164]:

```
cond = np.array([True, False, True, True, False])
```

In [165]:

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

In [166]:

```
result
```

Out[166]:

```
[1.1, 2.2, 1.3, 1.4, 2.5]
```

In [167]:

```
result = np.where(cond, xarr, yarr)
```

In [168]:

```
result
```

Out[168]:

```
array([1.1, 2.2, 1.3, 1.4, 2.5])
```

In [169]:

```
arr = np.random.randn(4, 4)
```

In [170]:

```
arr
```

Out[170]:

```
array([[-0.52442222, -0.67275346,  0.29386238, -0.55371318],
       [-0.21856838, -0.38363399, -1.4231228 , -0.99084378],
       [ 2.01092063,  2.68026592,  0.77700496,  0.98833326],
       [ 2.22563473,  0.55869953, -0.29800649,  1.36130314]])
```

In [171]:

```
arr > 0
```

Out[171]:

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

In [172]:

```
np.where(arr > 0, 2, -2)
```

Out[172]:

```
array([[-2, -2,  2, -2],
       [-2, -2, -2, -2],
       [ 2,  2,  2,  2],
       [ 2,  2, -2,  2]])
```

In [173]:

```
np.where(arr > 0, 2, arr)
```

Out[173]:

```
array([[-0.52442222, -0.67275346,  2.          , -0.55371318],
       [-0.21856838, -0.38363399, -1.4231228 , -0.99084378],
       [ 2.          ,  2.          ,  2.          ,  2.          ],
       [ 2.          ,  2.          , -0.29800649,  2.          ]])
```

In [174]:

```
arr = np.random.randn(5, 4)
```

In [175]:

```
arr
```

Out[175]:

```
array([[-1.46513175, -0.58716369,  0.16852988, -1.91935094],  
      [-0.64932491, -0.38275299, -0.62312036,  0.72632543],  
      [-0.14236024,  1.08424061,  2.02939834,  0.62161416],  
      [-0.40626315, -0.11733732,  0.22194803, -1.06636995],  
      [-1.49154861,  0.86840703,  0.4156475 , -0.09686933]])
```

In [176]:

```
arr.mean()
```

Out[176]:

```
-0.14057411287150873
```

In [177]:

```
np.mean(arr)
```

Out[177]:

```
-0.14057411287150873
```

In [178]:

```
arr.sum()
```

Out[178]:

```
-2.8114822574301748
```

In [179]:

```
arr.mean(axis=1)
```

Out[179]:

```
array([-0.95077912, -0.23221821,  0.89822322, -0.3420056 , -0.07609085])
```

In [180]:

```
arr.sum(axis=0)
```

Out[180]:

```
array([-4.15462865,  0.86539364,  2.21240338, -1.73465063])
```

In [181]:

```
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7])
```

In [182]:

```
arr.cumsum()
```

Out[182]:

```
array([ 0,  1,  3,  6, 10, 15, 21, 28], dtype=int32)
```

In [183]:

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

In [184]:

```
arr
```

Out[184]:

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

In [185]:

```
arr.cumsum(axis=0)
```

Out[185]:

```
array([[ 0,  1,  2],
       [ 3,  5,  7],
       [ 9, 12, 15]], dtype=int32)
```

In [186]:

```
arr.cumprod(axis=1)
```

Out[186]:

```
array([[ 0,  0,  0],
       [ 3, 12, 60],
       [ 6, 42, 336]], dtype=int32)
```

In [187]:

```
arr = np.random.randn(100)
```

In [188]:

```
(arr > 0).sum()
```

Out[188]:

```
51
```

In [189]:

```
bools = np.array([False, False, True, False])
```

In [190]:

```
bools.any()
```

Out[190]:

```
True
```

In [191]:

```
bools.all()
```

Out[191]:

```
False
```

In [192]:

```
arr = np.random.randn(6)
```

In [193]:

```
arr
```

Out[193]:

```
array([-1.86673297,  0.95579303, -0.22849914,  1.58890135,  1.21069676,
       -1.11876258])
```

In [194]:

```
arr.sort()
```

In [195]:

```
arr
```

Out[195]:

```
array([-1.86673297, -1.11876258, -0.22849914,  0.95579303,  1.21069676,
       1.58890135])
```

In [196]:

```
arr = np.random.randn(5, 3)
```

In [197]:

```
arr
```

Out[197]:

```
array([[ 0.51779175,  0.39849151, -0.6199541 ],
       [-2.30946476,  0.10110847, -0.434208 ],
       [-0.09250618, -2.60323199,  0.89315334],
       [-0.13912661,  0.06363791,  0.60158102],
       [-0.87911051, -0.10305209, -1.02013071]])
```

In [198]:

```
arr.sort(1)
```

In [199]:

```
arr
```

Out[199]:

```
array([[-0.6199541 ,  0.39849151,  0.51779175],  
       [-2.30946476, -0.434208 ,  0.10110847],  
       [-2.60323199, -0.09250618,  0.89315334],  
       [-0.13912661,  0.06363791,  0.60158102],  
       [-1.02013071, -0.87911051, -0.10305209]])
```

In [200]:

```
large_arr = np.random.randn(1000)
```

In [201]:

```
large_arr.sort()
```

In [202]:

```
large_arr[int(0.05 * len(large_arr))]
```

Out[202]:

```
-1.5693947001899085
```

In [203]:

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

In [204]:

```
np.unique(names)
```

Out[204]:

```
array(['Bob', 'Joe', 'Will'], dtype='<U4')
```

In [205]:

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

In [206]:

```
np.unique(ints)
```

Out[206]:

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

In [207]:

```
sorted(set(names))
```

Out[207]:

```
['Bob', 'Joe', 'Will']
```

In [208]:

```
values = np.array([6, 0, 0, 3, 2, 5, 6])
```

In [209]:

```
np.in1d(values, [2, 3, 6])
```

Out[209]:

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

In [210]:

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

In [211]:

```
np.save('some_array', arr)
```

In [212]:

```
np.load('some_array.npy')
```

Out[212]:

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

In [213]:

```
np.savez('array_archive.npz', a=arr, b=arr)
```

In [214]:

```
arch = np.load('array_archive.npz')
```

In [215]:

```
arch['b']
```

Out[215]:

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

In [216]:

```
np.savez_compressed('arrays_compressed.npz', a=arr, b=arr)
```

4.5 Linear Algebra

In [217]:

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

In [218]:

```
y = np.array([[6., 23.], [-1, 7], [8, 9]])
```

In [219]:

```
x
```

Out[219]:

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

In [220]:

```
y
```

Out[220]:

```
array([[ 6., 23.],  
       [-1.,  7.],  
       [ 8.,  9.]])
```

In [221]:

```
x.dot(y)
```

Out[221]:

```
array([[ 28., 64.],  
       [ 67., 181.]])
```

In [222]:

```
np.dot(x, y)
```

Out[222]:

```
array([[ 28., 64.],  
       [ 67., 181.]])
```

In [223]:

```
np.dot(x, np.ones(3))
```

Out[223]:

```
array([ 6., 15.])
```

In [224]:

```
x @ np.ones(3)
```

Out[224]:

```
array([ 6., 15.])
```

In [225]:

```
from numpy.linalg import inv, qr
```

In [226]:

```
X = np.random.randn(5, 5)
```

In [227]:

```
mat = X.T.dot(X)
```

In [228]:

```
inv(mat)
```

Out[228]:

```
array([[ 1.13886787,  0.80737848,  1.65930912, -0.90231063, -0.94305267],
       [ 0.80737848,  1.36534966,  2.10897892, -1.72043603, -1.43905945],
       [ 1.65930912,  2.10897892,  3.74508124, -2.6238966 , -2.49949948],
       [-0.90231063, -1.72043603, -2.6238966 ,  2.58389503,  1.72744792],
       [-0.94305267, -1.43905945, -2.49949948,  1.72744792,  2.03131108]])
```

In [229]:

```
mat.dot(inv(mat))
```

Out[229]:

```
array([[ 1.00000000e+00, -6.00605379e-17,  2.44883719e-15,
        4.83488680e-16, -1.17985795e-15],
       [ 5.91607620e-16,  1.00000000e+00,  1.61122654e-16,
        -2.93797939e-15, -1.32996385e-15],
       [-2.16995080e-16, -3.94370067e-16,  1.00000000e+00,
        4.91485372e-16,  1.12944813e-15],
       [-3.71687794e-16,  2.80887277e-16, -1.45853698e-15,
        1.00000000e+00,  1.54695459e-15],
       [-4.13863431e-16,  9.40547785e-16, -2.84070223e-16,
        -4.06922283e-16,  1.00000000e+00]])
```

In [230]:

```
q, r = qr(mat)
```

In [231]:

```
r
```

Out[231]:

```
array([[ -7.05596718, -4.36579148,  10.55155812,   2.07958974,
        4.98593508],
       [ 0.          , -11.25178643,   2.45274806,  -4.10330067,
        -1.54616088],
       [ 0.          ,  0.          , -2.09066217,  -0.68750431,
        -2.26747246],
       [ 0.          ,  0.          ,  0.          , -1.13455489,
        1.24160338],
       [ 0.          ,  0.          ,  0.          ,  0.          ,
        0.24755176]])
```

4.6 Pseudorandom Number Generation

In [232]:

```
samples = np.random.normal(size=(4, 4))
```

In [233]:

```
samples
```

Out[233]:

```
array([[-0.50910818,  0.41783668,  0.94906066,  0.06721888],  
      [-0.62766718,  0.77326766,  1.0249124 , -0.16444846],  
      [ 1.78482592, -0.91728494,  0.2002229 , -1.15872862],  
      [ 0.27065929, -0.72500929, -1.46617355, -0.62151305]])
```

In [234]:

```
from random import normalvariate
```

In [235]:

```
N = 1000000
```

In [236]:

```
%timeit samples = [normalvariate(0, 1) for _ in range(N)]
```

```
1.29 s ± 111 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [237]:

```
%timeit np.random.normal(size=N)
```

```
40 ms ± 3.7 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [238]:

```
np.random.seed(1234)
```

In [239]:

```
rng = np.random.RandomState(1234)
```

In [240]:

```
rng.randn(10)
```

Out[240]:

```
array([ 0.47143516, -1.19097569,  1.43270697, -0.3126519 , -0.72058873,  
      0.88716294,  0.85958841, -0.6365235 ,  0.01569637, -2.24268495])
```

4.7 Example: Random Walks

In [241]:

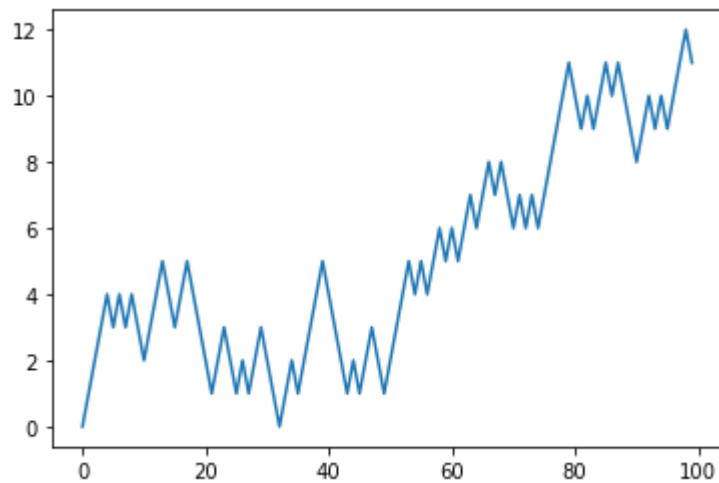
```
import random
position = 0
walk = [position]
steps = 1000
for i in range(steps):
    step = 1 if random.randint(0, 1) else -1
    position += step
    walk.append(position)
```

In [242]:

```
plt.plot(walk[:100])
```

Out[242]:

```
[<matplotlib.lines.Line2D at 0x24b392c7608>]
```



In [243]:

```
nsteps = 1000
```

In [244]:

```
draws = np.random.randint(0, 2, size=nsteps)
```

In [245]:

```
steps = np.where(draws > 0, 1, -1)
```

In [246]:

```
walk = steps.cumsum()
```

In [247]:

```
walk.min()
```

Out[247]:

```
-9
```

In [248]:

```
walk.max()
```

Out[248]:

60

In [249]:

```
(np.abs(walk) >= 10).argmax()
```

Out[249]:

297

In [250]:

```
nwalks = 5000
```

In [251]:

```
nsteps = 1000
```

In [252]:

```
draws = np.random.randint(0, 2, size=(nwalks, nsteps))
```

In [253]:

```
steps = np.where(draws > 0, 1, -1)
```

In [254]:

```
walks = steps.cumsum(1)
```

In [255]:

```
walks
```

Out[255]:

```
array([[ 1,   2,   3, ...,  46,   47,  46],
       [ 1,   0,   1, ...,  40,   41,  42],
       [ 1,   2,   3, ..., -26,  -27, -28],
       ...,
       [ 1,   0,   1, ...,  64,   65,  66],
       [ 1,   2,   1, ...,   2,    1,   0],
       [-1,  -2,  -3, ...,  32,  33,  34]], dtype=int32)
```

In [256]:

```
walks.max()
```

Out[256]:

122

In [257]:

```
walks.min()
```

Out[257]:

```
-128
```

In [258]:

```
hits30 = (np.abs(walks) >= 30).any(1)
```

In [259]:

```
hits30
```

Out[259]:

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

In [260]:

```
hits30.sum()
```

Out[260]:

```
3368
```

In [261]:

```
crossing_times = (np.abs(walks[hits30]) >= 30).argmax(1)
```

In [262]:

```
crossing_times.mean()
```

Out[262]:

```
509.99762470308787
```

In [263]:

```
steps = np.random.normal(loc=0, scale=0.25, size=(nwalks, nsteps))
```

GETTING STARTED WITH PANDAS

In [2]:

```
import pandas as pd
```

In [3]:

```
from pandas import Series, DataFrame
```

5.1 Introduction to pandas Data Structures

In [5]:

```
obj = pd.Series([4, 7, -5, 3])
```

In [6]:

```
obj
```

Out[6]:

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

In [7]:

```
obj.values
```

Out[7]:

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

In [8]:

```
obj.index
```

Out[8]:

```
RangeIndex(start=0, stop=4, step=1)
```

In [9]:

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

In [10]:

```
obj2
```

Out[10]:

```
d    4  
b    7  
a   -5  
c    3  
dtype: int64
```

In [11]:

```
obj2.index
```

Out[11]:

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

In [12]:

```
obj2['a']
```

Out[12]:

```
-5
```

In [13]:

```
obj2['d'] = 6
```

In [14]:

```
obj2[['c', 'a', 'd']]
```

Out[14]:

```
c    3  
a   -5  
d    6  
dtype: int64
```

In [15]:

```
obj2[obj2 > 0]
```

Out[15]:

```
d    6  
b    7  
c    3  
dtype: int64
```

In [16]:

```
obj2 * 2
```

Out[16]:

```
d    12  
b    14  
a   -10  
c     6  
dtype: int64
```

In [17]:

```
import numpy as np
```

In [18]:

```
np.exp(obj2)
```

Out[18]:

```
d    403.428793  
b   1096.633158  
a     0.006738  
c    20.085537  
dtype: float64
```

In [19]:

```
'b' in obj2
```

Out[19]:

True

In [20]:

```
'e' in obj2
```

Out[20]:

False

In [21]:

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

In [22]:

```
obj3 = pd.Series(sdata)
```

In [23]:

```
obj3
```

Out[23]:

```
Ohio      35000
Texas     71000
Oregon    16000
Utah      5000
dtype: int64
```

In [24]:

```
states = ['California', 'Ohio', 'Oregon', 'Texas']
```

In [25]:

```
obj4 = pd.Series(sdata, index=states)
```

In [26]:

```
obj4
```

Out[26]:

```
California      NaN
Ohio        35000.0
Oregon      16000.0
Texas       71000.0
dtype: float64
```

In [27]:

```
pd.isnull(obj4)
```

Out[27]:

```
California    True
Ohio          False
Oregon         False
Texas          False
dtype: bool
```

In [28]:

```
pd.notnull(obj4)
```

Out[28]:

```
California    False
Ohio          True
Oregon         True
Texas          True
dtype: bool
```

In [29]:

```
obj4.isnull()
```

Out[29]:

```
California    True
Ohio          False
Oregon         False
Texas          False
dtype: bool
```

In [30]:

```
obj3
```

Out[30]:

```
Ohio      35000
Texas     71000
Oregon    16000
Utah      5000
dtype: int64
```

In [31]:

```
obj4
```

Out[31]:

```
California      NaN
Ohio        35000.0
Oregon      16000.0
Texas        71000.0
dtype: float64
```

In [32]:

```
obj3 + obj4
```

Out[32]:

```
California      NaN
Ohio          70000.0
Oregon        32000.0
Texas         142000.0
Utah          NaN
dtype: float64
```

In [33]:

```
obj4.name = 'population'
```

In [34]:

```
obj4.index.name = 'state'
```

In [35]:

```
obj4
```

Out[35]:

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

In [36]:

```
obj
```

Out[36]:

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

In [37]:

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

In [38]:

```
obj
```

Out[38]:

```
Bob      4
Steve    7
Jeff     -5
Ryan     3
dtype: int64
```

In [39]:

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

In [40]:

```
frame
```

Out[40]:

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

In [41]:

```
frame.head()
```

Out[41]:

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

In [42]:

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

Out[42]:

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

In [43]:

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

In [44]:

```
frame2
```

Out[44]:

	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 [45]:

```
frame2.columns
```

Out[45]:

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

In [46]:

```
frame2['state']
```

Out[46]:

```
one      Ohio
two      Ohio
three    Ohio
four    Nevada
five    Nevada
six    Nevada
Name: state, dtype: object
```

In [47]:

```
frame2.year
```

Out[47]:

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

In [48]:

```
frame2.loc['three']
```

Out[48]:

```
year      2002
state    Ohio
pop       3.6
debt      NaN
Name: three, dtype: object
```

In [49]:

```
frame2['debt'] = 16.5
```

In [50]:

```
frame2
```

Out[50]:

	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 [51]:

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

In [52]:

```
frame2
```

Out[52]:

	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

In [53]:

```
val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])
```

In [54]:

```
frame2['debt'] = val
```

In [55]:

```
frame2
```

Out[55]:

	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

In [56]:

```
frame2['eastern'] = frame2.state == 'Ohio'
```

In [57]:

```
frame2
```

Out[57]:

	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 [58]:

```
del frame2['eastern']
```

In [59]:

```
frame2.columns
```

Out[59]:

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

In [60]:

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

In [61]:

```
frame3 = pd.DataFrame(pop)
```

In [62]:

```
frame3
```

Out[62]:

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

In [63]:

```
frame3.T
```

Out[63]:

	2001	2002	2000
Nevada	2.4	2.9	NaN
Ohio	1.7	3.6	1.5

In [64]:

```
pd.DataFrame(pop, index=[2001, 2002, 2003])
```

Out[64]:

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2003	NaN	NaN

In [65]:

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

In [66]:

```
pd.DataFrame(pdata)
```

Out[66]:

	Ohio	Nevada
2001	1.7	2.4
2002	3.6	2.9

In [67]:

```
frame3.index.name = 'year'; frame3.columns.name = 'state'
```

In [68]:

```
frame3
```

Out[68]:

state	Nevada	Ohio
year		
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

In [69]:

```
frame3.values
```

Out[69]:

```
array([[2.4, 1.7],  
       [2.9, 3.6],  
       [nan, 1.5]])
```

In [70]:

```
frame2.values
```

Out[70]:

```
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)
```

In [71]:

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

In [72]:

```
index = obj.index
```

In [73]:

```
index
```

Out[73]:

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

In [74]:

```
index[1:]
```

Out[74]:

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

In [75]:

```
labels = pd.Index(np.arange(3))
```

In [76]:

```
labels
```

Out[76]:

```
Int64Index([0, 1, 2], dtype='int64')
```

In [77]:

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

In [78]:

```
obj2
```

Out[78]:

```
0    1.5  
1   -2.5  
2    0.0  
dtype: float64
```

In [79]:

```
obj2.index is labels
```

Out[79]:

```
True
```

In [80]:

```
frame3
```

Out[80]:

state	Nevada	Ohio
year		
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

In [81]:

```
frame3.columns
```

Out[81]:

```
Index(['Nevada', 'Ohio'], dtype='object', name='state')
```

In [82]:

```
'Ohio' in frame3.columns
```

Out[82]:

```
True
```

In [83]:

```
2003 in frame3.index
```

Out[83]:

```
False
```

In [84]:

```
dup_labels = pd.Index(['foo', 'foo', 'bar', 'bar'])
```

In [85]:

```
dup_labels
```

Out[85]:

```
Index(['foo', 'foo', 'bar', 'bar'], dtype='object')
```

5.2 Essential Functionality

In [86]:

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

In [87]:

```
obj
```

Out[87]:

```
d    4.5  
b    7.2  
a   -5.3  
c    3.6  
dtype: float64
```

In [88]:

```
obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
```

In [89]:

```
obj2
```

Out[89]:

```
a   -5.3  
b    7.2  
c    3.6  
d    4.5  
e    NaN  
dtype: float64
```

In [90]:

```
obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])
```

In [91]:

```
obj3
```

Out[91]:

```
0      blue
2    purple
4   yellow
dtype: object
```

In [92]:

```
obj3.reindex(range(6), method='ffill')
```

Out[92]:

```
0      blue
1      blue
2    purple
3    purple
4   yellow
5   yellow
dtype: object
```

In [93]:

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

In [94]:

```
frame
```

Out[94]:

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

In [95]:

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

In [96]:

frame2

Out[96]:

	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

In [97]:

states = ['Texas', 'Utah', 'California']

In [98]:

frame.reindex(columns=states)

Out[98]:

	Texas	Utah	California
a	1	NaN	2
c	4	NaN	5
d	7	NaN	8

Desteklenmeyen Kod

Bu kod satırında; list-like'ları ve ya .loc[] kısımlarını herhangi bir eksik etiketle geçirmek artık desteklenmiyor.

frame.loc[['a', 'b', 'c', 'd'], states].

In [99]:

obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])

In [100]:

obj

Out[100]:

```
a    0.0
b    1.0
c    2.0
d    3.0
e    4.0
dtype: float64
```

In [101]:

new_obj = obj.drop('c')

In [102]:

```
new_obj
```

Out[102]:

```
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64
```

In [103]:

```
obj.drop(['d', 'c'])
```

Out[103]:

```
a    0.0
b    1.0
e    4.0
dtype: float64
```

In [104]:

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

In [105]:

```
data
```

Out[105]:

	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 [106]:

```
data.drop(['Colorado', 'Ohio'])
```

Out[106]:

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

In [107]:

```
data.drop('two', axis=1)
```

Out[107]:

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

In [108]:

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

Out[108]:

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

In [109]:

```
obj.drop('c', inplace=True)
```

In [110]:

```
obj
```

Out[110]:

```
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64
```

In [111]:

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

In [112]:

```
obj
```

Out[112]:

```
a    0.0  
b    1.0  
c    2.0  
d    3.0  
dtype: float64
```

In [113]:

```
obj['b']
```

Out[113]:

```
1.0
```

In [114]:

```
obj[1]
```

Out[114]:

```
1.0
```

In [115]:

```
obj[2:4]
```

Out[115]:

```
c    2.0  
d    3.0  
dtype: float64
```

In [116]:

```
obj[['b', 'a', 'd']]
```

Out[116]:

```
b    1.0  
a    0.0  
d    3.0  
dtype: float64
```

In [117]:

```
obj[[1, 3]]
```

Out[117]:

```
b    1.0  
d    3.0  
dtype: float64
```

In [118]:

```
obj[obj < 2]
```

Out[118]:

```
a    0.0  
b    1.0  
dtype: float64
```

In [119]:

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

Out[119]:

```
b    1.0  
c    2.0  
dtype: float64
```

In [120]:

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

In [121]:

```
obj
```

Out[121]:

```
a    0.0  
b    5.0  
c    5.0  
d    3.0  
dtype: float64
```

In [122]:

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

In [123]:

```
data
```

Out[123]:

	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 [124]:

```
data['two']
```

Out[124]:

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

In [125]:

```
data[['three', 'one']]
```

Out[125]:

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

In [126]:

```
data[:2]
```

Out[126]:

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

In [127]:

```
data[data['three'] > 5]
```

Out[127]:

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

In [128]:

```
data < 5
```

Out[128]:

	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 [129]:

```
data[data < 5] = 0
```

In [130]:

```
data
```

Out[130]:

	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

In [131]:

```
data.loc['Colorado', ['two', 'three']]
```

Out[131]:

```
two      5
three    6
Name: Colorado, dtype: int32
```

In [132]:

```
data.iloc[2, [3, 0, 1]]
```

Out[132]:

```
four     11
one      8
two      9
Name: Utah, dtype: int32
```

In [133]:

```
data.iloc[2]
```

Out[133]:

```
one      8
two      9
three    10
four     11
Name: Utah, dtype: int32
```

In [134]:

```
data.iloc[[1, 2], [3, 0, 1]]
```

Out[134]:

	four	one	two
Colorado	7	0	5
Utah	11	8	9

In [135]:

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

Out[135]:

```
Ohio      0
Colorado  5
Utah     9
Name: two, dtype: int32
```

In [136]:

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

Out[136]:

	one	two	three
Colorado	0	5	6
Utah	8	9	10
New York	12	13	14

In [137]:

```
ser = pd.Series(np.arange(3.))
```

In [138]:

```
ser
```

Out[138]:

```
0    0.0
1    1.0
2    2.0
dtype: float64
```

In [139]:

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

In [140]:

```
ser2[-1]
```

Out[140]:

```
2.0
```

In [141]:

```
ser[:1]
```

Out[141]:

```
0    0.0  
dtype: float64
```

In [142]:

```
ser.loc[:1]
```

Out[142]:

```
0    0.0  
1    1.0  
dtype: float64
```

In [143]:

```
ser.iloc[:1]
```

Out[143]:

```
0    0.0  
dtype: float64
```

In [144]:

```
s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])
```

In [145]:

```
s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],  
               index=['a', 'c', 'e', 'f', 'g'])
```

In [146]:

```
s1
```

Out[146]:

```
a    7.3  
c   -2.5  
d    3.4  
e    1.5  
dtype: float64
```

In [147]:

```
s2
```

Out[147]:

```
a    -2.1
c     3.6
e    -1.5
f     4.0
g     3.1
dtype: float64
```

In [148]:

```
s1 + s2
```

Out[148]:

```
a     5.2
c     1.1
d    NaN
e     0.0
f    NaN
g    NaN
dtype: float64
```

In [149]:

```
df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)), columns=list('bcd'),
                   index=['Ohio', 'Texas', 'Colorado'])
```

In [150]:

```
df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)), columns=list('bde'),
                   index=['Utah', 'Ohio', 'Texas', 'Oregon'])
```

In [151]:

```
df1
```

Out[151]:

	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 [152]:

```
df2
```

Out[152]:

	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 [153]:

```
df1 + df2
```

Out[153]:

	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

In [154]:

```
df1 = pd.DataFrame({'A': [1, 2]})
```

In [155]:

```
df2 = pd.DataFrame({'B': [3, 4]})
```

In [156]:

```
df1
```

Out[156]:

A
0 1
1 2

In [157]:

```
df2
```

Out[157]:

B
0 3
1 4

In [158]:

```
df1 = df2
```

Out[158]:

	A	B
0	NaN	NaN
1	NaN	NaN

In [159]:

```
df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),  
                   columns=list('abcd'))
```

In [160]:

```
df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),  
                   columns=list('abcde'))
```

In [161]:

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

In [162]:

```
df1
```

Out[162]:

	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 [163]:

```
df2
```

Out[163]:

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

In [164]:

```
df1 + df2
```

Out[164]:

	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 [165]:

```
df1.add(df2, fill_value=0)
```

Out[165]:

	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 [166]:

```
1 / df1
```

Out[166]:

	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 [167]:

```
df1.rdiv(1)
```

Out[167]:

	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 [168]:

```
df1.reindex(columns=df2.columns, fill_value=0)
```

Out[168]:

	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

In [169]:

```
arr = np.arange(12.).reshape((3, 4))
```

In [170]:

```
arr
```

Out[170]:

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

In [171]:

```
arr[0]
```

Out[171]:

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

In [172]:

```
arr - arr[0]
```

Out[172]:

```
array([[0., 0., 0., 0.],
       [4., 4., 4., 4.],
       [8., 8., 8., 8.]])
```

In [173]:

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

In [174]:

```
series = frame.iloc[0]
```

In [175]:

```
frame
```

Out[175]:

	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 [176]:

```
series
```

Out[176]:

```
b      0.0
d      1.0
e      2.0
Name: Utah, dtype: float64
```

In [177]:

```
frame - series
```

Out[177]:

	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

In [178]:

```
series2 = pd.Series(range(3), index=['b', 'e', 'f'])
```

In [179]:

```
frame + series2
```

Out[179]:

	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

In [180]:

```
series3 = frame['d']
```

In [181]:

```
frame
```

Out[181]:

	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 [182]:

```
series3
```

Out[182]:

```
Utah      1.0
Ohio      4.0
Texas     7.0
Oregon    10.0
Name: d, dtype: float64
```

In [183]:

```
frame.sub(series3, axis='index')
```

Out[183]:

	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

In [184]:

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

In [185]:

```
frame
```

Out[185]:

	b	d	e
Utah	-0.083668	-1.878915	-1.786479
Ohio	1.265967	1.352018	-0.856850
Texas	0.336862	-0.590033	0.121906
Oregon	1.646787	-0.125270	1.140563

In [186]:

```
np.abs(frame)
```

Out[186]:

	b	d	e
Utah	0.083668	1.878915	1.786479
Ohio	1.265967	1.352018	0.856850
Texas	0.336862	0.590033	0.121906
Oregon	1.646787	0.125270	1.140563

In [187]:

```
f = lambda x: x.max() - x.min()
```

In [188]:

```
frame.apply(f)
```

Out[188]:

```
b      1.730455
d      3.230934
e      2.927043
dtype: float64
```

In [189]:

```
frame.apply(f, axis='columns')
```

Out[189]:

```
Utah      1.795247
Ohio      2.208868
Texas     0.926895
Oregon    1.772057
dtype: float64
```

In [190]:

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

In [191]:

```
frame.apply(f)
```

Out[191]:

	b	d	e
min	-0.083668	-1.878915	-1.786479
max	1.646787	1.352018	1.140563

In [192]:

```
format = lambda x: '%.2f' % x
```

In [193]:

```
frame.applymap(format)
```

Out[193]:

	b	d	e
Utah	-0.08	-1.88	-1.79
Ohio	1.27	1.35	-0.86
Texas	0.34	-0.59	0.12
Oregon	1.65	-0.13	1.14

In [194]:

```
frame['e'].map(format)
```

Out[194]:

```
Utah      -1.79
Ohio      -0.86
Texas      0.12
Oregon     1.14
Name: e, dtype: object
```

In [195]:

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

In [196]:

```
obj.sort_index()
```

Out[196]:

```
a    1
b    2
c    3
d    0
dtype: int64
```

In [197]:

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

In [198]:

```
frame.sort_index()
```

Out[198]:

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

In [199]:

```
frame.sort_index(axis=1)
```

Out[199]:

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

In [200]:

```
frame.sort_index(axis=1, ascending=False)
```

Out[200]:

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

In [201]:

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

In [202]:

```
obj.sort_values()
```

Out[202]:

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

In [203]:

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

In [204]:

```
obj.sort_values()
```

Out[204]:

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

In [205]:

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

In [206]:

```
frame
```

Out[206]:

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

In [207]:

```
frame.sort_values(by='b')
```

Out[207]:

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

In [208]:

```
frame.sort_values(by=['a', 'b'])
```

Out[208]:

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

In [209]:

```
obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
```

In [210]:

```
obj.rank()
```

Out[210]:

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

In [211]:

```
obj.rank(method='first')
```

Out[211]:

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

In [212]:

```
obj.rank(ascending=False, method='max')
```

Out[212]:

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

In [213]:

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

In [214]:

```
frame
```

Out[214]:

	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 [215]:

```
frame.rank(axis='columns')
```

Out[215]:

	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

In [216]:

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

In [217]:

```
obj
```

Out[217]:

```
a    0
a    1
b    2
b    3
c    4
dtype: int64
```

In [218]:

```
obj.index.is_unique
```

Out[218]:

```
False
```

In [219]:

```
obj['a']
```

Out[219]:

```
a    0  
a    1  
dtype: int64
```

In [220]:

```
obj['c']
```

Out[220]:

```
4
```

In [221]:

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

In [222]:

```
df
```

Out[222]:

	0	1	2
a	1.144871	0.438762	1.141320
a	1.968487	-0.223418	-1.071772
b	0.625030	-0.104022	-0.671640
b	0.394987	-0.056002	-0.391660

In [223]:

```
df.loc['b']
```

Out[223]:

	0	1	2
b	0.625030	-0.104022	-0.67164
b	0.394987	-0.056002	-0.39166

5.3 Summarizing and Computing Descriptive Statistics

In [224]:

```
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 [225]:

```
df
```

Out[225]:

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

In [226]:

```
df.sum()
```

Out[226]:

```
one      9.25
two     -5.80
dtype: float64
```

In [227]:

```
df.sum(axis='columns')
```

Out[227]:

```
a      1.40
b      2.60
c      0.00
d     -0.55
dtype: float64
```

In [228]:

```
df.mean(axis='columns', skipna=False)
```

Out[228]:

```
a      NaN
b     1.300
c      NaN
d    -0.275
dtype: float64
```

In [229]:

```
df.idxmax()
```

Out[229]:

```
one    b
two    d
dtype: object
```

In [230]:

```
df.cumsum()
```

Out[230]:

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

In [231]:

```
df.describe()
```

Out[231]:

	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

In [232]:

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

In [233]:

```
obj.describe()
```

Out[233]:

count	16
unique	3
top	a
freq	8
dtype:	object

In [234]:

```
import pandas_datareader.data as web
all_data = {ticker: web.get_data_yahoo(ticker)
            for ticker in ['AAPL', 'IBM', 'MSFT', 'GOOG']}
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 [235]:

```
returns = price.pct_change()
```

In [236]:

```
returns.tail()
```

Out[236]:

	AAPL	IBM	MSFT	GOOG
Date				
2020-11-05	0.035494	0.025648	0.031887	0.008141
2020-11-06	-0.001136	-0.006361	0.001926	-0.000919
2020-11-09	-0.019968	0.027756	-0.023824	0.000710
2020-11-10	-0.003009	0.020601	-0.033793	-0.012825
2020-11-11	0.029323	-0.000891	0.028008	0.008021

In [237]:

```
returns['MSFT'].corr(returns['IBM'])
```

Out[237]:

0.562528897725141

In [238]:

```
returns['MSFT'].cov(returns['IBM'])
```

Out[238]:

0.0001600215259902638

In [239]:

```
returns.MSFT.corr(returns.IBM)
```

Out[239]:

0.562528897725141

In [240]:

```
returns.corr()
```

Out[240]:

	AAPL	IBM	MSFT	GOOG
AAPL	1.000000	0.480694	0.718947	0.657034
IBM	0.480694	1.000000	0.562529	0.523105
MSFT	0.718947	0.562529	1.000000	0.780319
GOOG	0.657034	0.523105	0.780319	1.000000

In [241]:

```
returns.cov()
```

Out[241]:

	AAPL	IBM	MSFT	GOOG
AAPL	0.000361	0.000148	0.000240	0.000208
IBM	0.000148	0.000263	0.000160	0.000141
MSFT	0.000240	0.000160	0.000308	0.000228
GOOG	0.000208	0.000141	0.000228	0.000278

In [242]:

```
returns.corrwith(returns.IBM)
```

Out[242]:

```
AAPL    0.480694
IBM    1.000000
MSFT    0.562529
GOOG    0.523105
dtype: float64
```

In [243]:

```
returns.corrwith(volume)
```

Out[243]:

```
AAPL   -0.097509
IBM   -0.078047
MSFT   -0.097972
GOOG   -0.163017
dtype: float64
```

In [244]:

```
obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])
```

In [245]:

```
uniques = obj.unique()
```

In [246]:

```
uniques
```

Out[246]:

```
array(['c', 'a', 'd', 'b'], dtype=object)
```

In [247]:

```
obj.value_counts()
```

Out[247]:

```
a    3  
c    3  
b    2  
d    1  
dtype: int64
```

In [248]:

```
pd.value_counts(obj.values, sort=False)
```

Out[248]:

```
d    1  
c    3  
b    2  
a    3  
dtype: int64
```

In [249]:

```
obj
```

Out[249]:

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

In [250]:

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

In [251]:

```
mask
```

Out[251]:

```
0    True
1   False
2   False
3   False
4   False
5    True
6    True
7    True
8    True
dtype: bool
```

In [252]:

```
obj[mask]
```

Out[252]:

```
0    c
5    b
6    b
7    c
8    c
dtype: object
```

In [253]:

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

In [254]:

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

In [255]:

```
pd.Index(unique_vals).get_indexer(to_match)
```

Out[255]:

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

In [256]:

```
data = pd.DataFrame({'Qu1': [1, 3, 4, 3, 4],
                     'Qu2': [2, 3, 1, 2, 3],
                     'Qu3': [1, 5, 2, 4, 4]})
```

In [257]:

```
data
```

Out[257]:

	Qu1	Qu2	Qu3
0	1	2	1
1	3	3	5
2	4	1	2
3	3	2	4
4	4	3	4

In [258]:

```
result = data.apply(pd.value_counts).fillna(0)
```

In [259]:

```
result
```

Out[259]:

	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

6. DATA LOADING, STORAGE AND FILE FORMATS

6.1 Reading and Writing Data in Text Format

In [1]:

```
import pandas as pd
```

In [2]:

```
import numpy as np
```

In [3]:

```
!type "C:\Users\Kader\Desktop\examples\ex1.csv"
```

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

In [4]:

```
df = pd.read_csv('examples/ex1.csv')
```

In [5]:

```
df
```

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

In [6]:

```
pd.read_table('examples/ex1.csv', sep=',')
```

Out[6]:

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

In [7]:

```
!type "C:\Users\Kader\Desktop\examples\ex2.csv"
```

1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo

In [8]:

```
pd.read_csv('examples/ex2.csv', header=None)
```

Out[8]:

	0	1	2	3	4
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

In [9]:

```
pd.read_csv('examples/ex2.csv', names=['a', 'b', 'c', 'd', 'message'])
```

Out[9]:

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

In [10]:

```
names = ['a', 'b', 'c', 'd', 'message']
```

In [11]:

```
pd.read_csv('examples/ex2.csv', names=names, index_col='message')
```

Out[11]:

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

In [12]:

```
!type "C:\Users\Kader\Desktop\examples\csv_mindex.csv"
```

```
key1,key2,value1,value2
one,a,1,2
one,b,3,4
one,c,5,6
one,d,7,8
two,a,9,10
two,b,11,12
two,c,13,14
two,d,15,16
```

In [13]:

```
parsed = pd.read_csv('examples/csv_mindex.csv',
                      index_col=['key1', 'key2'])
```

In [14]:

parsed

Out[14]:

	value1	value2
key1	key2	
	a	1
	b	2
one	c	3
	d	4
	a	5
two	b	6
	c	7
	d	8
	a	9
	b	10
	c	11
	d	12
	a	13
	b	14
	c	15
	d	16

In [15]:

list(open('examples/ex3.txt'))

Out[15]:

```
[ '           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']
```

In [16]:

result = pd.read_table('examples/ex3.txt', sep='\s+')

In [17]:

result

Out[17]:

	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

In [18]:

```
!type "C:\Users\Kader\Desktop\examples\ex4.csv"
```

```
# hey!
a,b,c,d,message
# just wanted to make things more difficult for you
# who reads CSV files with computers, anyway?
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

In [19]:

```
pd.read_csv('examples/ex4.csv', skiprows=[0, 2, 3])
```

Out[19]:

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

In [20]:

```
result = pd.read_csv('examples/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

In [23]:

```
result = pd.read_csv('examples/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

In [25]:

```
sentinels = {'message': ['foo', 'NA'], 'something': ['two']}
```

In [26]:

```
pd.read_csv('examples/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]:

```
pd.options.display.max_rows = 10
```

In [28]:

```
result = pd.read_csv('examples/ex6.csv')
```

In [29]:

result

Out[29]:

	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	O

10000 rows × 5 columns

In [30]:

pd.read_csv('examples/ex6.csv', nrows=5)

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

In [31]:

chunker = pd.read_csv('examples/ex6.csv', chunksize=1000)

In [32]:

chunker

Out[32]:

<pandas.io.parsers.TextFileReader at 0x1b40ee388c8>

In [33]:

chunker = pd.read_csv('examples/ex6.csv', chunksize=1000)

In [34]:

```
tot = pd.Series([])
for piece in chunker:
    tot = tot.add(piece['key'].value_counts(), fill_value=0)
tot = tot.sort_values(ascending=False)
```

...

In [35]:

```
tot[:10]
```

Out[35]:

```
E    368.0
X    364.0
L    346.0
O    343.0
Q    340.0
M    338.0
J    337.0
F    335.0
K    334.0
H    330.0
dtype: float64
```

In [36]:

```
data = pd.read_csv('examples/ex5.csv')
```

In [37]:

```
data
```

Out[37]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

In [38]:

```
data.to_csv('examples/out.csv')
```

In [39]:

```
!type "C:\Users\Kader\Desktop\examples\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
```

In [40]:

```
import sys
```

In [41]:

```
data.to_csv(sys.stdout, sep='|')
```

```
|something|a|b|c|d|message  
0|one|1|2|3.0|4|  
1|two|5|6||8|world  
2|three|9|10|11.0|12|foo
```

In [42]:

```
data.to_csv(sys.stdout, na_rep='NULL')
```

```
,something,a,b,c,d,message  
0,one,1,2,3.0,4,NULL  
1,two,5,6,NULL,8,world  
2,three,9,10,11.0,12,foo
```

In [43]:

```
data.to_csv(sys.stdout, index=False, header=False)
```

```
one,1,2,3.0,4,  
two,5,6,,8,world  
three,9,10,11.0,12,foo
```

In [44]:

```
data.to_csv(sys.stdout, index=False, columns=['a', 'b', 'c'])
```

```
a,b,c  
1,2,3.0  
5,6,  
9,10,11.0
```

In [45]:

```
dates = pd.date_range('1/1/2000', periods=7)
```

In [46]:

```
ts = pd.Series(np.arange(7), index=dates)
```

In [47]:

```
ts.to_csv('examples/tseries.csv')
```

In [48]:

```
!type "C:\Users\Kader\Desktop\examples\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
```

In [49]:

```
!type "C:\Users\Kader\Desktop\examples\ex7.csv"  
"a","b","c"  
"1","2","3"  
"1","2","3"
```

In [50]:

```
import csv
```

In [51]:

```
f = open('examples/ex7.csv')
```

In [52]:

```
reader = csv.reader(f)
```

In [53]:

```
for line in reader:  
    print(line)
```

```
['a', 'b', 'c']  
['1', '2', '3']  
['1', '2', '3']
```

In [54]:

```
with open('examples/ex7.csv') as f:  
    lines = list(csv.reader(f))
```

In [55]:

```
header, values = lines[0], lines[1:]
```

In [56]:

```
data_dict = {h: v for h, v in zip(header, zip(*values))}
```

In [57]:

```
data_dict
```

Out[57]:

```
{'a': ('1', '1'), 'b': ('2', '2'), 'c': ('3', '3')}
```

In [58]:

```
class my_dialect(csv.Dialect):  
    lineterminator = '\n'  
    delimiter = ';'   
    quotechar = '"'   
    quoting = csv.QUOTE_MINIMAL
```

CSV Lehçesi

csv.reader anahtar sözcüklerine bir altsınıf tanımlamadan parametreler verebiliriz:

```
reader = csv.reader(f, dialect=my_dialect)
reader = csv.reader(f, delimiter='|')
```

In [60]:

```
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'])
```

In [61]:

```
import json
```

In [62]:

```
obj = """
{"name": "Wes",
"places_lived": ["United States", "Spain", "Germany"],
"pet": null,
"siblings": [{"name": "Scott", "age": 30, "pets": ["Zeus", "Zuko"]},
 {"name": "Katie", "age": 38,
 "pets": ["Sixes", "Stache", "Cisco"]}]]
```

In [63]:

```
result = json.loads(obj)
```

In [64]:

```
result
```

Out[64]:

```
{'name': 'Wes',
'places_lived': ['United States', 'Spain', 'Germany'],
'pet': None,
'siblings': [{'name': 'Scott', 'age': 30, 'pets': ['Zeus', 'Zuko']},
 {'name': 'Katie', 'age': 38, 'pets': ['Sixes', 'Stache', 'Cisco']}]}
```

In [65]:

```
asjson = json.dumps(result)
```

In [66]:

```
siblings = pd.DataFrame(result['siblings'], columns=['name', 'age'])
```

In [67]:

```
siblings
```

Out[67]:

	name	age
0	Scott	30
1	Katie	38

In [68]:

```
!type "C:\Users\Kader\Desktop\examples\example.json"
```

```
[{"a": 1, "b": 2, "c": 3},  
 {"a": 4, "b": 5, "c": 6},  
 {"a": 7, "b": 8, "c": 9}]
```

In [69]:

```
data = pd.read_json('examples/example.json')
```

In [70]:

```
data
```

Out[70]:

	a	b	c
0	1	2	3
1	4	5	6
2	7	8	9

In [71]:

```
print(data.to_json())
```

```
{"a": {"0": 1, "1": 4, "2": 7}, "b": {"0": 2, "1": 5, "2": 8}, "c": {"0": 3, "1": 6, "2": 9}}
```

In [72]:

```
print(data.to_json(orient='records'))
```

```
[{"a": 1, "b": 2, "c": 3}, {"a": 4, "b": 5, "c": 6}, {"a": 7, "b": 8, "c": 9}]
```

In [73]:

```
tables = pd.read_html('examples/fdic_failed_bank_list.html')
```

In [74]:

`len(tables)`

Out[74]:

1

In [75]:

`failures = tables[0]`

In [76]:

`failures.head()`

Out[76]:

	Bank Name	City	ST	CERT	Acquiring Institution	Closing Date	Updated Date
0	Allied Bank	Mulberry	AR	91	Today's Bank	September 23, 2016	November 17, 2016
1	The Woodbury Banking Company	Woodbury	GA	11297	United Bank	August 19, 2016	November 17, 2016
2	First CornerStone Bank	King of Prussia	PA	35312	First-Citizens Bank & Trust Company	May 6, 2016	September 6, 2016
3	Trust Company Bank	Memphis	TN	9956	The Bank of Fayette County	April 29, 2016	September 6, 2016
4	North Milwaukee State Bank	Milwaukee	WI	20364	First-Citizens Bank & Trust Company	March 11, 2016	June 16, 2016

In [77]:

`close_timestamps = pd.to_datetime(failures['Closing Date'])`

In [78]:

`close_timestamps.dt.year.value_counts()`

Out[78]:

```

2010    157
2009    140
2011     92
2012     51
2008     25
...
2004      4
2001      4
2007      3
2003      3
2000      2
Name: Closing Date, Length: 15, dtype: int64

```

In [79]:

`from lxml import objectify`

In [80]:

```
path = 'examples/xml1.xml'  
parsed = objectify.parse(open(path))  
root = parsed.getroot()
```

In [81]:

```
data = []
```

In [82]:

```
skip_fields = ['PARENT_SEQ', 'INDICATOR_SEQ',  
               'DESIRED_CHANGE', 'DECIMAL_PLACES']
```

Burada, lxml.objectify kullanarak dosyayı ayırtırır ve dosyanın köküne bir referans alırız. root.INDICATOR , her XML ögesini veren bir oluşturucu döndürür. Her kayıt için veri değerlerine bir dizi etiket adı yerleştirilir.

```
for elt in root.INDICATOR:  
    el_data = {}  
    for child in elt.getchildren():  
        if child.tag in skip_fields:  
            continue  
        el_data[child.tag] = child.pyval  
    data.append(el_data)
```

In [83]:

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

In [84]:

```
perf.head()
```

Out[84]:

—

In [85]:

```
from io import StringIO  
tag = '<a href="http://www.google.com">Google</a>'  
root = objectify.parse(StringIO(tag)).getroot()
```

In [86]:

```
root
```

Out[86]:

```
<Element a at 0x1b4101ea7c8>
```

In [87]:

```
root.get('href')
```

Out[87]:

```
'http://www.google.com'
```

In [88]:

```
root.text
```

Out[88]:

```
'Google'
```

6.2 Binary Data Formats

In [89]:

```
frame = pd.read_csv('examples/ex1.csv')
```

In [90]:

```
frame
```

Out[90]:

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

In [91]:

```
frame.to_pickle('examples/frame_pickle')
```

In [92]:

```
pd.read_pickle('examples/frame_pickle')
```

Out[92]:

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

In [93]:

```
frame = pd.DataFrame({'a': np.random.randn(100)})
```

In [94]:

```
store = pd.HDFStore('mydata.h5')
```

In [95]:

```
store['obj1'] = frame
```

In [96]:

```
store['obj1_col'] = frame['a']
```

In [97]:

```
store
```

Out[97]:

```
<class 'pandas.io.pytables.HDFStore'>
File path: mydata.h5
```

In [98]:

```
store['obj1']
```

Out[98]:

	a
0	-1.140216
1	-0.474919
2	-0.969032
3	-1.194343
4	-1.139544
...	...
95	-0.909514
96	-1.843600
97	0.180415
98	-1.013965
99	0.084011

100 rows × 1 columns

In [99]:

```
store.put('obj2', frame, format='table')
```

In [100]:

```
store.select('obj2', where=['index >= 10 and index <= 15'])
```

Out[100]:

	a
10	-1.644969
11	0.598612
12	0.671541
13	0.481664
14	-0.016005
15	-1.099209

In [101]:

```
store.close()
```

In [102]:

```
frame.to_hdf('mydata.h5', 'obj3', format='table')
```

In [103]:

```
pd.read_hdf('mydata.h5', 'obj3', where=['index < 5'])
```

Out[103]:

	a
0	-1.140216
1	-0.474919
2	-0.969032
3	-1.194343
4	-1.139544

In [104]:

```
xlsx = pd.ExcelFile('examples/ex1.xlsx')
```

In [105]:

```
pd.read_excel(xlsx, 'Sheet1')
```

Out[105]:

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

In [106]:

```
frame = pd.read_excel('examples/ex1.xlsx', 'Sheet1')
```

In [107]:

```
frame
```

Out[107]:

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

In [108]:

```
writer = pd.ExcelWriter('examples/ex2.xlsx')
```

In [109]:

```
frame.to_excel(writer, 'Sayfa1')
```

In [110]:

```
writer.save()
```

In [111]:

```
frame.to_excel('examples/ex2.xlsx')
```

6.3 Interacting with Web APIs

In [112]:

```
import requests
```

In [113]:

```
url = 'https://api.github.com/repos/pandas-dev/pandas/issues'
```

In [114]:

```
resp = requests.get(url)
```

In [115]:

```
resp
```

Out[115]:

```
<Response [200]>
```

In [116]:

```
data = resp.json()
```

In [117]:

```
data[0]['title']
```

Out[117]:

```
'BUG: pylint with merge of pandas==1.1.5 raises a RecursionError'
```

In [118]:

```
issues = pd.DataFrame(data, columns=['number', 'title',
                                         'labels', 'state'])
```

In [119]:

issues

Out[119]:

number		title	labels	state
0	38355	BUG: pylint with merge of pandas==1.1.5 raises...	[{"id": 76811, "node_id": "MDU6TGFiZWw3NjgxMQ="}	open
1	38354	TST: a weird bug with numpy + DataFrame with d...	[]	open
2	38353	PERF: IntervalIndex.isin	[]	open
3	38352	ENH: column-wise DataFrame.fillna with Series/...	[]	open
4	38351	BUG: item_cache invalidation	[]	open
...
25	38316	BUG: IntervalIndex.astype("category") doesn't ...	[{"id": 76811, "node_id": "MDU6TGFiZWw3NjgxMQ="}	open
26	38315	API: add EA._from_scalars / stricter casting o...	[{"id": 35818298, "node_id": "MDU6TGFiZWwzNTgx..."}]	open
27	38312	PERF: Period constructor parses input twice	[{"id": 76811, "node_id": "MDU6TGFiZWw3NjgxMQ="}]	open
28	38311	DOC: Wrong output in the example of Timedelta....	[{"id": 134699, "node_id": "MDU6TGFiZWwxMzQ2OT..."}]	open
29	38308	REF: implement Index._get_indexer	[]	open

30 rows × 4 columns

6.4 Interacting with Databases

In [120]:

import sqlite3

In [121]:

```
query = """
CREATE TABLE test
(a VARCHAR(20), b VARCHAR(20),
c REAL, d INTEGER
);"""
```

In [122]:

con = sqlite3.connect('mydata.sqlite')

In [125]:

con.commit()

In [126]:

```
data = [('Atlanta', 'Georgia', 1.25, 6),
        ('Tallahassee', 'Florida', 2.6, 3),
        ('Sacramento', 'California', 1.7, 5)]
```

In [127]:

```
stmt = "INSERT INTO test VALUES(?, ?, ?, ?, ?)"
```

In [128]:

```
con.executemany(stmt, data)
```

Out[128]:

```
<sqlite3.Cursor at 0x1b410b03b20>
```

In [129]:

```
con.commit()
```

In [130]:

```
cursor = con.execute('select * from test')
```

In [131]:

```
rows = cursor.fetchall()
```

In [132]:

```
rows
```

Out[132]:

```
[('Atlanta', 'Georgia', 1.25, 6),
 ('Tallahassee', 'Florida', 2.6, 3),
 ('Sacramento', 'California', 1.7, 5),
 ('Atlanta', 'Georgia', 1.25, 6),
 ('Tallahassee', 'Florida', 2.6, 3),
 ('Sacramento', 'California', 1.7, 5),
 ('Atlanta', 'Georgia', 1.25, 6),
 ('Tallahassee', 'Florida', 2.6, 3),
 ('Sacramento', 'California', 1.7, 5),
 ('Atlanta', 'Georgia', 1.25, 6),
 ('Tallahassee', 'Florida', 2.6, 3),
 ('Sacramento', 'California', 1.7, 5)]
```

In [133]:

```
cursor.description
```

Out[133]:

```
(('a', None, None, None, None, None),
 ('b', None, None, None, None, None),
 ('c', None, None, None, None, None),
 ('d', None, None, None, None, None))
```

In [134]:

```
pd.DataFrame(rows, columns=[x[0] for x in cursor.description])
```

Out[134]:

	a	b	c	d
0	Atlanta	Georgia	1.25	6
1	Tallahassee	Florida	2.60	3
2	Sacramento	California	1.70	5
3	Atlanta	Georgia	1.25	6
4	Tallahassee	Florida	2.60	3
...
7	Tallahassee	Florida	2.60	3
8	Sacramento	California	1.70	5
9	Atlanta	Georgia	1.25	6
10	Tallahassee	Florida	2.60	3
11	Sacramento	California	1.70	5

12 rows × 4 columns

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
3	Atlanta	Georgia	1.25	6
4	Tallahassee	Florida	2.60	3
...
7	Tallahassee	Florida	2.60	3
8	Sacramento	California	1.70	5
9	Atlanta	Georgia	1.25	6
10	Tallahassee	Florida	2.60	3
11	Sacramento	California	1.70	5

12 rows × 4 columns

7. DATA CLEANING AND PREPARATION

7.1 Handling Missing Data

In [138]:

```
string_data = pd.Series(['aardvark', 'artichoke', np.nan, 'avocado'])
```

In [139]:

```
string_data
```

Out[139]:

```
0    aardvark
1    artichoke
2      NaN
3    avocado
dtype: object
```

In [140]:

```
string_data.isnull()
```

Out[140]:

```
0    False
1    False
2    True
3    False
dtype: bool
```

In [141]:

```
string_data[0] = None
```

In [142]:

```
string_data.isnull()
```

Out[142]:

```
0    True  
1   False  
2    True  
3   False  
dtype: bool
```

In [143]:

```
from numpy import nan as NA
```

In [144]:

```
data = pd.Series([1, NA, 3.5, NA, 7])
```

In [145]:

```
data.dropna()
```

Out[145]:

```
0    1.0  
2    3.5  
4    7.0  
dtype: float64
```

In [146]:

```
data[data.notnull()]
```

Out[146]:

```
0    1.0  
2    3.5  
4    7.0  
dtype: float64
```

In [147]:

```
data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],  
                     [NA, NA, NA], [NA, 6.5, 3.]])
```

In [148]:

```
cleaned = data.dropna()
```

In [149]:

```
data
```

Out[149]:

	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 [150]:

```
cleaned
```

Out[150]:

	0	1	2
0	1.0	6.5	3.0

In [151]:

```
data.dropna(how='all')
```

Out[151]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
3	NaN	6.5	3.0

In [152]:

```
data[4] = NA
```

In [153]:

```
data
```

Out[153]:

	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 [154]:

```
data.dropna(axis=1, how='all')
```

Out[154]:

	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 [155]:

```
df = pd.DataFrame(np.random.randn(7, 3))
```

In [156]:

```
df.iloc[:4, 1] = NA
```

In [157]:

```
df.iloc[:2, 2] = NA
```

In [158]:

```
df
```

Out[158]:

	0	1	2
0	0.164140	NaN	NaN
1	0.254921	NaN	NaN
2	1.873093	NaN	-0.926607
3	-0.540286	NaN	0.469361
4	-0.197793	1.184040	0.773902
5	0.199574	1.139050	-0.179459
6	-0.943022	0.883367	1.411129

In [159]:

```
df.dropna()
```

Out[159]:

	0	1	2
4	-0.197793	1.184040	0.773902
5	0.199574	1.139050	-0.179459
6	-0.943022	0.883367	1.411129

In [160]:

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

Out[160]:

	0	1	2
2	1.873093	NaN	-0.926607
3	-0.540286	NaN	0.469361
4	-0.197793	1.184040	0.773902
5	0.199574	1.139050	-0.179459
6	-0.943022	0.883367	1.411129

In [161]:

```
df.fillna(0)
```

Out[161]:

	0	1	2
0	0.164140	0.000000	0.000000
1	0.254921	0.000000	0.000000
2	1.873093	0.000000	-0.926607
3	-0.540286	0.000000	0.469361
4	-0.197793	1.184040	0.773902
5	0.199574	1.139050	-0.179459
6	-0.943022	0.883367	1.411129

In [162]:

```
df.fillna({1: 0.5, 2: 0})
```

Out[162]:

	0	1	2
0	0.164140	0.500000	0.000000
1	0.254921	0.500000	0.000000
2	1.873093	0.500000	-0.926607
3	-0.540286	0.500000	0.469361
4	-0.197793	1.184040	0.773902
5	0.199574	1.139050	-0.179459
6	-0.943022	0.883367	1.411129

In [163]:

```
df.fillna(0, inplace=True)
```

In [164]:

```
df
```

Out[164]:

	0	1	2
0	0.164140	0.000000	0.000000
1	0.254921	0.000000	0.000000
2	1.873093	0.000000	-0.926607
3	-0.540286	0.000000	0.469361
4	-0.197793	1.184040	0.773902
5	0.199574	1.139050	-0.179459
6	-0.943022	0.883367	1.411129

In [165]:

```
df = pd.DataFrame(np.random.randn(6, 3))
```

In [166]:

```
df.iloc[2:, 1] = NA
```

In [167]:

```
df.iloc[4:, 2] = NA
```

In [168]:

```
df
```

Out[168]:

	0	1	2
0	0.812731	-0.474345	-0.744843
1	-0.161890	0.598268	-1.428514
2	-0.179994	NaN	-0.970464
3	-0.213703	NaN	-0.503832
4	-0.576697	NaN	NaN
5	-2.106264	NaN	NaN

In [169]:

```
df.fillna(method='ffill')
```

Out[169]:

	0	1	2
0	0.812731	-0.474345	-0.744843
1	-0.161890	0.598268	-1.428514
2	-0.179994	0.598268	-0.970464
3	-0.213703	0.598268	-0.503832
4	-0.576697	0.598268	-0.503832
5	-2.106264	0.598268	-0.503832

In [170]:

```
df.fillna(method='ffill', limit=2)
```

Out[170]:

	0	1	2
0	0.812731	-0.474345	-0.744843
1	-0.161890	0.598268	-1.428514
2	-0.179994	0.598268	-0.970464
3	-0.213703	0.598268	-0.503832
4	-0.576697	NaN	-0.503832
5	-2.106264	NaN	-0.503832

In [171]:

```
data = pd.Series([1., NA, 3.5, NA, 7])
```

In [172]:

```
data.fillna(data.mean())
```

Out[172]:

```
0    1.000000
1    3.833333
2    3.500000
3    3.833333
4    7.000000
dtype: float64
```

7.2 Data Transformation

In [173]:

```
data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],
                     'k2': [1, 1, 2, 3, 3, 4, 4]})
```

In [174]:

```
data
```

Out[174]:

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4

In [175]:

```
data.duplicated()
```

Out[175]:

```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
```

In [176]:

```
data.drop_duplicates()
```

Out[176]:

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4

In [177]:

```
data['v1'] = range(7)
```

In [178]:

```
data.drop_duplicates(['k1'])
```

Out[178]:

	k1	k2	v1
0	one	1	0
1	two	1	1

In [179]:

```
data.drop_duplicates(['k1', 'k2'], keep='last')
```

Out[179]:

	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

In [180]:

```
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 [181]:

```
data
```

Out[181]:

	food	ounces
0	bacon	4.0
1	pulled pork	3.0
2	bacon	12.0
3	Pastrami	6.0
4	corned beef	7.5
5	Bacon	8.0
6	pastrami	3.0
7	honey ham	5.0
8	nova lox	6.0

In [182]:

```
meat_to_animal = {  
    'bacon': 'pig',  
    'pulled pork': 'pig',  
    'pastrami': 'cow',  
    'corned beef': 'cow',  
    'honey ham': 'pig',  
    'nova lox': 'salmon'  
}
```

In [183]:

```
lowercased = data['food'].str.lower()
```

In [184]:

```
lowercased
```

Out[184]:

```
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 [185]:

```
data['animal'] = lowercased.map(meat_to_animal)
```

In [186]:

```
data
```

Out[186]:

	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 [187]:

```
data['food'].map(lambda x: meat_to_animal[x.lower()])
```

Out[187]:

```
0      pig
1      pig
2      pig
3     cow
4     cow
5      pig
6     cow
7      pig
8    salmon
Name: food, dtype: object
```

In [188]:

```
data = pd.Series([1., -999., 2., -999., -1000., 3.])
```

In [189]:

```
data
```

Out[189]:

```
0      1.0
1    -999.0
2      2.0
3    -999.0
4   -1000.0
5      3.0
dtype: float64
```

In [190]:

```
data.replace(-999, np.nan)
```

Out[190]:

```
0      1.0
1      NaN
2      2.0
3      NaN
4   -1000.0
5      3.0
dtype: float64
```

In [191]:

```
data.replace([-999, -1000], np.nan)
```

Out[191]:

```
0    1.0
1    NaN
2    2.0
3    NaN
4    NaN
5    3.0
dtype: float64
```

In [192]:

```
data.replace([-999, -1000], [np.nan, 0])
```

Out[192]:

```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

In [193]:

```
data.replace({-999: np.nan, -1000: 0})
```

Out[193]:

```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

In [194]:

```
data = pd.DataFrame(np.arange(12).reshape((3, 4)),
                     index=['Ohio', 'Colorado', 'New York'],
                     columns=['one', 'two', 'three', 'four'])
```

In [195]:

```
transform = lambda x: x[:4].upper()
```

In [196]:

```
data.index.map(transform)
```

Out[196]:

```
Index(['OHIO', 'COLO', 'NEW '], dtype='object')
```

In [197]:

```
data.index = data.index.map(transform)
```

In [198]:

```
data
```

Out[198]:

	one	two	three	four
OHIO	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

In [199]:

```
data.rename(index=str.title, columns=str.upper)
```

Out[199]:

	ONE	TWO	THREE	FOUR
Ohio	0	1	2	3
Colo	4	5	6	7
New	8	9	10	11

In [200]:

```
data.rename(index={'OHIO': 'INDIANA'},  
           columns={'three': 'peekaboo'})
```

Out[200]:

	one	two	peekaboo	four
INDIANA	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

In [201]:

```
data.rename(index={'OHIO': 'INDIANA'}, inplace=True)
```

In [202]:

```
data
```

Out[202]:

	one	two	three	four
INDIANA	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

In [203]:

```
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

In [204]:

```
bins = [18, 25, 35, 60, 100]
```

In [205]:

```
cats = pd.cut(ages, bins)
```

In [206]:

```
cats
```

Out[206]:

```
[(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]]
```

In [207]:

```
cats.codes
```

Out[207]:

```
array([0, 0, 0, 1, 0, 0, 2, 1, 3, 2, 2, 1], dtype=int8)
```

In [208]:

```
cats.categories
```

Out[208]:

```
IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]),  
              closed='right',  
              dtype='interval[int64]')
```

In [209]:

```
pd.value_counts(cats)
```

Out[209]:

```
(18, 25]      5  
(35, 60]      3  
(25, 35]      3  
(60, 100]     1  
dtype: int64
```

In [210]:

```
pd.cut(ages, [18, 26, 36, 61, 100], right=False)
```

Out[210]:

```
[[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) < [61, 100)]
```

In [211]:

```
group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']
```

In [212]:

```
pd.cut(ages, bins, labels=group_names)
```

Out[212]:

```
[Youth, Youth, Youth, YoungAdult, Youth, ..., YoungAdult, Senior, MiddleAge
d, MiddleAged, YoungAdult]
Length: 12
Categories (4, object): [Youth < YoungAdult < MiddleAged < Senior]
```

In [213]:

```
data = np.random.rand(20)
```

In [214]:

```
pd.cut(data, 4, precision=2)
```

Out[214]:

```
[(0.47, 0.69], (0.029, 0.25], (0.69, 0.92], (0.47, 0.69], (0.029, 0.25],
..., (0.47, 0.69], (0.47, 0.69], (0.69, 0.92], (0.25, 0.47], (0.25, 0.47])
Length: 20
Categories (4, interval[float64]): [(0.029, 0.25] < (0.25, 0.47] < (0.47, 0.
69] < (0.69, 0.92)]
```

In [215]:

```
data = np.random.randn(1000)
```

In [216]:

```
cats = pd.qcut(data, 4)
```

In [217]:

cats

Out[217]:

```
[(-0.0203, 0.608], (0.608, 3.952], (0.608, 3.952], (-0.0203, 0.608], (-0.0203, 0.608], ..., (0.608, 3.952], (-0.0203, 0.608], (0.608, 3.952], (-0.729, -0.0203], (-0.729, -0.0203])]
```

Length: 1000

```
Categories (4, interval[float64]): [(-3.7729999999999997, -0.729] < (-0.729, -0.0203] < (-0.0203, 0.608] < (0.608, 3.952)]
```

In [218]:

pd.value_counts(cats)

Out[218]:

(0.608, 3.952]	250
(-0.0203, 0.608]	250
(-0.729, -0.0203]	250
(-3.7729999999999997, -0.729]	250

dtype: int64

In [219]:

pd.qcut(data, [0, 0.1, 0.5, 0.9, 1.])

Out[219]:

```
[(-0.0203, 1.27], (1.27, 3.952], (-0.0203, 1.27], (-0.0203, 1.27], (-0.0203, 1.27], ..., (-0.0203, 1.27], (-0.0203, 1.27], (1.27, 3.952], (-1.326, -0.0203], (-1.326, -0.0203])]
```

Length: 1000

```
Categories (4, interval[float64]): [(-3.7729999999999997, -1.326] < (-1.326, -0.0203] < (-0.0203, 1.27] < (1.27, 3.952)]
```

In [220]:

data = pd.DataFrame(np.random.randn(1000, 4))

In [221]:

data.describe()

Out[221]:

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	-0.042950	-0.043312	-0.008712	0.030418
std	1.005750	0.978492	1.013142	1.009662
min	-3.259720	-3.191609	-2.787412	-3.597708
25%	-0.693330	-0.682616	-0.674683	-0.622991
50%	-0.024937	-0.052297	-0.028983	0.059467
75%	0.599331	0.635536	0.665827	0.677136
max	2.997451	3.052445	3.178509	3.180646

In [222]:

```
col = data[2]
```

In [223]:

```
col[np.abs(col) > 3]
```

Out[223]:

```
224    3.091222
732    3.178509
Name: 2, dtype: float64
```

In [224]:

```
data[(np.abs(data) > 3).any(1)]
```

Out[224]:

	0	1	2	3
26	-3.259720	-1.698871	1.164691	-0.151138
31	0.392494	-1.259042	0.180128	-3.597708
199	-1.962987	-1.608089	-0.241299	-3.156348
224	-1.642968	0.369607	3.091222	-0.279516
549	-3.179174	-0.807890	-0.343622	1.083949
...
772	1.252777	-1.159586	-1.656036	3.180646
895	-0.654832	-1.406271	0.414081	-3.064328
906	0.184750	-1.356650	-0.460952	3.098103
962	-3.199767	-0.429709	-0.701625	-0.649733
986	-1.087994	3.052445	-0.165891	-0.217274

13 rows × 4 columns

In [225]:

```
data[np.abs(data) > 3] = np.sign(data) * 3
```

In [226]:

```
data.describe()
```

Out[226]:

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	-0.042311	-0.043093	-0.008982	0.030957
std	1.003799	0.977486	1.012320	1.006164
min	-3.000000	-3.000000	-2.787412	-3.000000
25%	-0.693330	-0.682616	-0.674683	-0.622991
50%	-0.024937	-0.052297	-0.028983	0.059467
75%	0.599331	0.635536	0.665827	0.677136
max	2.997451	3.000000	3.000000	3.000000

In [227]:

```
np.sign(data).head()
```

Out[227]:

	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

In [228]:

```
df = pd.DataFrame(np.arange(5 * 4).reshape((5, 4)))
```

In [229]:

```
sampler = np.random.permutation(5)
```

In [230]:

```
sampler
```

Out[230]:

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

In [231]:

```
df
```

Out[231]:

	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 [232]:

```
df.take(sampler)
```

Out[232]:

	0	1	2	3
0	0	1	2	3
2	8	9	10	11
1	4	5	6	7
4	16	17	18	19
3	12	13	14	15

In [233]:

```
df.sample(n=3)
```

Out[233]:

	0	1	2	3
1	4	5	6	7
0	0	1	2	3
4	16	17	18	19

In [234]:

```
choices = pd.Series([5, 7, -1, 6, 4])
```

In [235]:

```
draws = choices.sample(n=10, replace=True)
```

In [236]:

```
draws
```

Out[236]:

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

In [237]:

```
df = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'], 'data1': range(6)})
```

In [238]:

```
pd.get_dummies(df['key'])
```

Out[238]:

	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

In [239]:

```
dummies = pd.get_dummies(df['key'], prefix='key')
```

In [240]:

```
df_with_dummy = df[['data1']].join(dummies)
```

In [241]:

df_with_dummy

Out[241]:

	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

In [242]:

mnames = ['movie_id', 'title', 'genres']

In [243]:

movies = pd.read_table('examples/movies.dat', sep='::', header=None, names=mnames)

...

In [244]:

movies[:10]

Out[244]:

	movie_id	title	genres
0	movie_id	title	genres
1	0 1 Toy Story (1995) Animation Children's Comedy	NaN	NaN
2	1 2 Jumanji (1995) Adventure Children's Fantasy	NaN	NaN
3	2 3 Grumpier Old Men (1995) Comedy Romance	NaN	NaN
4	3 4 Waiting to Exhale (1995) Comedy Drama	NaN	NaN
5	4 5 Father of the Bride Part II (1995) Comedy	NaN	NaN
6	5 6 Heat (1995) Action Crime Thriller	NaN	NaN
7	6 7 Sabrina (1995) Comedy Romance	NaN	NaN
8	7 8 Tom and Huck (1995) Adventure Children's	NaN	NaN
9	8 9 Sudden Death (1995) Action	NaN	NaN

In [245]:

all_genres = []

In [246]:

for x in movies.genres:
 all_genres.extend(str(x).split('|'))

In [247]:

```
genres = pd.unique(all_genres)
```

In [248]:

```
genres
```

Out[248]:

```
array(['nan'], dtype=object)
```

In [249]:

```
zero_matrix = np.zeros((len(movies), len(genres)))
```

In [250]:

```
dummies = pd.DataFrame(zero_matrix, columns=genres)
```

In [251]:

```
gen = movies.genres[0]
```

In [253]:

```
str(gen).split('|')
```

Out[253]:

```
['nan']
```

In [254]:

```
dummies.columns.get_indexer(str(gen).split('|'))
```

Out[254]:

```
array([0], dtype=int64)
```

In [255]:

```
for i, gen in enumerate(movies.genres):
    indices = dummies.columns.get_indexer(str(gen).split('|'))
    dummies.iloc[i, indices] = 1
```

In [256]:

```
movies_windic = movies.join(dummies.add_prefix('Genre_'))
```

In [257]:

```
movies_windic.iloc[0]
```

Out[257]:

```
movie_id      movie_id title genres
title                   NaN
genres                  NaN
Genre_nan                1
Name: 0, dtype: object
```

In [258]:

```
np.random.seed(12345)
```

In [259]:

```
values = np.random.rand(10)
```

In [260]:

```
values
```

Out[260]:

```
array([0.92961609, 0.31637555, 0.18391881, 0.20456028, 0.56772503,
       0.5955447 , 0.96451452, 0.6531771 , 0.74890664, 0.65356987])
```

In [261]:

```
bins = [0, 0.2, 0.4, 0.6, 0.8, 1]
```

In [262]:

```
pd.get_dummies(pd.cut(values, bins))
```

Out[262]:

	(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 Manipulation

In [263]:

```
val = 'a,b, guido'
```

In [264]:

```
val.split(',')
```

Out[264]:

```
['a', 'b', ' guido']
```

In [265]:

```
pieces = [x.strip() for x in val.split(',')]
```

In [266]:

```
pieces
```

Out[266]:

```
['a', 'b', 'guido']
```

In [267]:

```
first, second, third = pieces
```

In [268]:

```
first + '::' + second + '::' + third
```

Out[268]:

```
'a::b::guido'
```

In [269]:

```
:::.join(pieces)
```

Out[269]:

```
'a::b::guido'
```

In [270]:

```
'guido' in val
```

Out[270]:

```
True
```

In [271]:

```
val.index(',')
```

Out[271]:

```
1
```

In [272]:

```
val.find(':')
```

Out[272]:

```
-1
```

In [273]:

```
val.index(':')
```

ValueError

Traceback (most recent call last)

```
<ipython-input-273-2c016e7367ac> in <module>
```

```
----> 1 val.index(':')
```

ValueError: substring not found

In [274]:

```
val.count(',')
```

Out[274]:

```
2
```

In [275]:

```
val.replace(',', '::')
```

Out[275]:

```
'a::b:: guido'
```

In [276]:

```
val.replace(',', '')
```

Out[276]:

```
'ab guido'
```

In [277]:

```
import re
```

In [278]:

```
text = "foo bar\tbaz \tqux"
```

In [279]:

```
re.split('\s+', text)
```

Out[279]:

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

In [280]:

```
regex = re.compile('\s+')
```

In [281]:

```
regex.split(text)
```

Out[281]:

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

In [282]:

```
regex.findall(text)
```

Out[282]:

```
[' ', '\t ', '\t']
```

In [283]:

```
text = """Dave dave@google.com
Steve steve@gmail.com
Rob rob@gmail.com
Ryan ryan@yahoo.com
"""
```

In [284]:

```
pattern = r'[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}'
```

In [285]:

```
regex = re.compile(pattern, flags=re.IGNORECASE)
```

In [286]:

```
regex.findall(text)
```

Out[286]:

```
['dave@google.com', 'steve@gmail.com', 'rob@gmail.com', 'ryan@yahoo.com']
```

In [287]:

```
m = regex.search(text)
```

In [288]:

```
m
```

Out[288]:

```
<re.Match object; span=(5, 20), match='dave@google.com'>
```

In [289]:

```
text[m.start():m.end()]
```

Out[289]:

```
'dave@google.com'
```

In [290]:

```
print(regex.match(text))
```

```
None
```

In [291]:

```
print(regex.sub('REDACTED', text))
```

```
Dave REDACTED
```

```
Steve REDACTED
```

```
Rob REDACTED
```

```
Ryan REDACTED
```

In [292]:

```
pattern = r'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})'
```

In [293]:

```
regex = re.compile(pattern, flags=re.IGNORECASE)
```

In [294]:

```
m = regex.match('wesm@bright.net')
```

In [295]:

```
m.groups()
```

Out[295]:

```
('wesm', 'bright', 'net')
```

In [296]:

```
regex.findall(text)
```

Out[296]:

```
[('dave', 'google', 'com'),  
 ('steve', 'gmail', 'com'),  
 ('rob', 'gmail', 'com'),  
 ('ryan', 'yahoo', 'com')]
```

In [297]:

```
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
```

In [298]:

```
data = {'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com',
        'Rob': 'rob@gmail.com', 'Wes': np.nan}
```

In [299]:

```
data = pd.Series(data)
```

In [300]:

```
data
```

Out[300]:

```
Dave      dave@google.com
Steve     steve@gmail.com
Rob       rob@gmail.com
Wes          NaN
dtype: object
```

In [301]:

```
data.isnull()
```

Out[301]:

```
Dave      False
Steve     False
Rob      False
Wes       True
dtype: bool
```

In [302]:

```
data.str.contains('gmail')
```

Out[302]:

```
Dave      False
Steve     True
Rob      True
Wes       NaN
dtype: object
```

In [303]:

```
pattern
```

Out[303]:

```
'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.\([A-Z]{2,4}\)'
```

In [304]:

```
data.str.findall(pattern, flags=re.IGNORECASE)
```

Out[304]:

```
Dave      [(dave, google, com)]
Steve     [(steve, gmail, com)]
Rob       [(rob, gmail, com)]
Wes        NaN
dtype: object
```

In [305]:

```
matches = data.str.match(pattern, flags=re.IGNORECASE)
```

In [306]:

```
matches
```

Out[306]:

```
Dave    True
Steve   True
Rob     True
Wes     NaN
dtype: object
```

Gömülü listelerdeki öğelere erişmek için dizin aktarma fonksiyonları:

```
matches.str.get(1)
matches.str[0].
```

In [307]:

```
data.str[:5]
```

Out[307]:

```
Dave    dave@
Steve   steve
Rob    rob@g
Wes     NaN
dtype: object
```

8. DATA WRANGLING: JOIN, COMBINE AND RESHAPE

8.1 Hierarchical Indexing

In [308]:

```
import numpy as np
```

In [309]:

```
import pandas as pd
```

In [310]:

```
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 [311]:

```
data
```

Out[311]:

```
a    1    1.007189
     2   -1.296221
     3    0.274992
b    1    0.228913
     3    1.352917
c    1    0.886429
     2   -2.001637
d    2   -0.371843
     3    1.669025
dtype: float64
```

In [312]:

```
data.index
```

Out[312]:

```
MultiIndex([(('a', 1),
              ('a', 2),
              ('a', 3),
              ('b', 1),
              ('b', 3),
              ('c', 1),
              ('c', 2),
              ('d', 2),
              ('d', 3)],
             ))
```

In [313]:

```
data['b']
```

Out[313]:

```
1    0.228913
3    1.352917
dtype: float64
```

In [314]:

```
data['b':'c']
```

Out[314]:

```
b    1    0.228913
     3    1.352917
c    1    0.886429
     2   -2.001637
dtype: float64
```

In [315]:

```
data.loc[['b', 'd']]
```

Out[315]:

```
b    1    0.228913
     3    1.352917
d    2   -0.371843
     3    1.669025
dtype: float64
```

In [316]:

```
data.loc[:, 2]
```

Out[316]:

```
a   -1.296221
c   -2.001637
d   -0.371843
dtype: float64
```

In [317]:

```
data.unstack()
```

Out[317]:

	1	2	3
a	1.007189	-1.296221	0.274992
b	0.228913	NaN	1.352917
c	0.886429	-2.001637	NaN
d	NaN	-0.371843	1.669025

In [318]:

```
data.unstack().stack()
```

Out[318]:

```
a    1    1.007189  
     2   -1.296221  
     3    0.274992  
b    1    0.228913  
     3   1.352917  
c    1    0.886429  
     2   -2.001637  
d    2   -0.371843  
     3    1.669025  
dtype: float64
```

In [319]:

```
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 [320]:

```
frame
```

Out[320]:

	Ohio	Colorado	
	Green	Red	Green
a	1	0	1
	2	3	4
b	1	6	7
	2	9	10

In [321]:

```
frame.index.names = ['key1', 'key2']
```

In [322]:

```
frame.columns.names = ['state', 'color']
```

In [323]:

frame

Out[323]:

	state	Ohio	Colorado	
	color	Green	Red	Green
key1	key2			
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

In [324]:

frame['Ohio']

Out[324]:

	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 sütunların seviye adlarına sahip bir DataFrame şu şekilde oluşturulabilir:

```
MultiIndex.from_arrays([['Ohio', 'Ohio', 'Colorado'], ['Green', 'Red', 'Green']], names=['state', 'color'])
```

In [325]:

frame.swaplevel('key1', 'key2')

Out[325]:

	state	Ohio	Colorado	
	color	Green	Red	Green
key2	key1			
1	a	0	1	2
2	a	3	4	5
1	b	6	7	8
2	b	9	10	11

In [326]:

```
frame.sort_index(level=1)
```

Out[326]:

	state	Ohio	Colorado	
	color	Green	Red	Green
key1	key2			
	a	1	0	1
	b	1	6	7
	a	2	3	4
	b	2	9	10

In [327]:

```
frame.swaplevel(0, 1).sort_index(level=0)
```

Out[327]:

	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

In [328]:

```
frame.sum(level='key2')
```

Out[328]:

	state	Ohio	Colorado	
	color	Green	Red	Green
key2				
1		6	8	10
2		12	14	16

In [329]:

```
frame.sum(level='color', axis=1)
```

Out[329]:

	color	Green	Red
key1	key2		
a	1	2	1
	2	8	4
b	1	14	7
	2	20	10

In [330]:

```
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 [331]:

```
frame
```

Out[331]:

	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

In [332]:

```
frame2 = frame.set_index(['c', 'd'])
```

In [333]:

```
frame2
```

Out[333]:

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

In [334]:

```
frame.set_index(['c', 'd'], drop=False)
```

Out[334]:

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

In [335]:

```
frame2.reset_index()
```

Out[335]:

	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

In [336]:

```
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'], 'data1': range(7)})
```

In [337]:

```
df2 = pd.DataFrame({'key': ['a', 'b', 'd'], 'data2': range(3)})
```

In [338]:

```
df1
```

Out[338]:

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	a	5
6	b	6

In [339]:

```
df2
```

Out[339]:

	key	data2
0	a	0
1	b	1
2	d	2

In [340]:

```
pd.merge(df1, df2)
```

Out[340]:

	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

In [341]:

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

Out[341]:

	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

In [342]:

```
df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'], 'data1': range(7)})
```

In [517]:

```
df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'], 'data2': range(3)})
```

In [518]:

```
pd.merge(df3, df4, left_on='lkey', right_on='rkey')
```

Out[518]:

	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

In [519]:

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

Out[519]:

	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 [520]:

```
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'], 'data1': range(6)})
```

In [521]:

```
df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'], 'data2': range(5)})
```

In [522]:

```
df1
```

Out[522]:

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	b	5

In [523]:

```
df2
```

Out[523]:

	key	data2
0	a	0
1	b	1
2	a	2
3	b	3
4	d	4

In [524]:

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

Out[524]:

	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
...
6	c	3	NaN
7	a	4	0.0
8	a	4	2.0
9	b	5	1.0
10	b	5	3.0

11 rows × 3 columns

In [525]:

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

Out[525]:

	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

In [526]:

```
left = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
                     'key2': ['one', 'two', 'one'],
                     'lval': [1, 2, 3]})
```

In [527]:

```
right = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
                      'key2': ['one', 'one', 'one', 'two'],
                      'rval': [4, 5, 6, 7]})
```

In [528]:

```
pd.merge(left, right, on=['key1', 'key2'], how='outer')
```

Out[528]:

	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

In [529]:

```
pd.merge(left, right, on='key1')
```

Out[529]:

	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 [530]:

```
pd.merge(left, right, on='key1', suffixes=('_left', '_right'))
```

Out[530]:

	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 [531]:

```
left1 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'], 'value': range(6)})
```

In [532]:

```
right1 = pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
```

In [533]:

```
left1
```

Out[533]:

	key	value
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

In [534]:

```
right1
```

Out[534]:

	group_val
a	3.5
b	7.0

In [535]:

```
pd.merge(left1, right1, left_on='key', right_index=True)
```

Out[535]:

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0

In [536]:

```
pd.merge(left1, right1, left_on='key', right_index=True, how='outer')
```

Out[536]:

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0
5	c	5	NaN

In [537]:

```
lefth = pd.DataFrame({'key1': ['Ohio', 'Ohio', 'Ohio',
                               'Nevada', 'Nevada'],
                      'key2': [2000, 2001, 2002, 2001, 2002],
                      'data': np.arange(5.)})
```

In [538]:

```
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 [539]:

```
lefth
```

Out[539]:

	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 [540]:

```
righth
```

Out[540]:

		event1	event2
Nevada	2001	0	1
	2000	2	3
Ohio	2000	4	5
	2000	6	7
	2001	8	9
	2002	10	11

In [541]:

```
pd.merge(lefth, righth, left_on=['key1', 'key2'], right_index=True)
```

Out[541]:

	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 [542]:

```
pd.merge(lefth, righth, left_on=['key1', 'key2'],
         right_index=True, how='outer')
```

Out[542]:

	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

In [543]:

```
left2 = pd.DataFrame([[1., 2.], [3., 4.], [5., 6.]],
                     index=['a', 'c', 'e'],
                     columns=['Ohio', 'Nevada'])
```

In [544]:

```
right2 = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [13, 14]],
                      index=['b', 'c', 'd', 'e'],
                      columns=['Missouri', 'Alabama'])
```

In [545]:

left2

Out[545]:

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

In [546]:

```
right2
```

Out[546]:

	Missouri	Alabama
b	7.0	8.0
c	9.0	10.0
d	11.0	12.0
e	13.0	14.0

In [547]:

```
pd.merge(left2, right2, how='outer', left_index=True, right_index=True)
```

Out[547]:

	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

In [548]:

```
left2.join(right2, how='outer')
```

Out[548]:

	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

In [549]:

```
left1.join(right1, on='key')
```

Out[549]:

	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

In [550]:

```
another = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [16., 17.]],
                      index=['a', 'c', 'e', 'f'],
                      columns=['New York', 'Oregon'])
```

In [551]:

```
another
```

Out[551]:

	New York	Oregon
a	7.0	8.0
c	9.0	10.0
e	11.0	12.0
f	16.0	17.0

In [552]:

```
left2.join([right2, another])
```

Out[552]:

	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 [553]:

```
left2.join([right2, another], how='outer')
```

Out[553]:

	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

In [554]:

```
arr = np.arange(12).reshape((3, 4))
```

In [555]:

```
arr
```

Out[555]:

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

In [556]:

```
np.concatenate([arr, arr], axis=1)
```

Out[556]:

```
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]])
```

In [557]:

```
s1 = pd.Series([0, 1], index=['a', 'b'])
```

In [558]:

```
s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])
```

In [559]:

```
s3 = pd.Series([5, 6], index=['f', 'g'])
```

In [560]:

```
pd.concat([s1, s2, s3])
```

Out[560]:

```
a    0  
b    1  
c    2  
d    3  
e    4  
f    5  
g    6  
dtype: int64
```

In [561]:

```
pd.concat([s1, s2, s3], axis=1)
```

Out[561]:

	0	1	2
a	0.0	NaN	NaN
b	1.0	NaN	NaN
c	NaN	2.0	NaN
d	NaN	3.0	NaN
e	NaN	4.0	NaN
f	NaN	NaN	5.0
g	NaN	NaN	6.0

In [562]:

```
s4 = pd.concat([s1, s3])
```

In [563]:

```
s4
```

Out[563]:

```
a    0  
b    1  
f    5  
g    6  
dtype: int64
```

In [564]:

```
pd.concat([s1, s4], axis=1)
```

Out[564]:

	0	1
a	0.0	0
b	1.0	1
f	NaN	5
g	NaN	6

In [565]:

```
pd.concat([s1, s4], axis=1, join='inner')
```

Out[565]:

	0	1
a	0	0
b	1	1

Join_axes ile diğer eksenlerde kullanılacak eksenleri bile belirtebiliriz:

```
pd.concat([s1, s4], axis=1, join_axes=[['a', 'c', 'b', 'e']])
```

In [567]:

```
result = pd.concat([s1, s1, s3], keys=['one', 'two', 'three'])
```

In [568]:

```
result
```

Out[568]:

one	a	0
	b	1
two	a	0
	b	1
three	f	5
	g	6

dtype: int64

In [569]:

```
result.unstack()
```

Out[569]:

	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

In [570]:

```
pd.concat([s1, s2, s3], axis=1, keys=['one', 'two', 'three'])
```

Out[570]:

	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

In [571]:

```
df1 = pd.DataFrame(np.arange(6).reshape(3, 2), index=['a', 'b', 'c'], columns=['one', 'two'])
```

In [572]:

```
df2 = pd.DataFrame(5 + np.arange(4).reshape(2, 2), index=['a', 'c'], columns=['three', 'four'])
```

In [573]:

```
df1
```

Out[573]:

	one	two
a	0	1
b	2	3
c	4	5

In [574]:

```
df2
```

Out[574]:

	three	four
a	5	6
c	7	8

In [575]:

```
pd.concat([df1, df2], axis=1, keys=['level1', 'level2'])
```

Out[575]:

	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 [576]:

```
pd.concat({'level1': df1, 'level2': df2}, axis=1)
```

Out[576]:

	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 [577]:

```
pd.concat([df1, df2], axis=1, keys=['level1', 'level2'], names=['upper', 'lower'])
```

Out[577]:

	upper		level1		level2	
	lower	one	two	three	four	
a		0	1	5.0	6.0	
b		2	3	NaN	NaN	
c		4	5	7.0	8.0	

In [578]:

```
df1 = pd.DataFrame(np.random.randn(3, 4), columns=['a', 'b', 'c', 'd'])
```

In [579]:

```
df2 = pd.DataFrame(np.random.randn(2, 3), columns=['b', 'd', 'a'])
```

In [580]:

```
df1
```

Out[580]:

	a	b	c	d
0	-0.438570	-0.539741	0.476985	3.248944
1	-1.021228	-0.577087	0.124121	0.302614
2	0.523772	0.000940	1.343810	-0.713544

In [581]:

```
df2
```

Out[581]:

	b	d	a
0	-0.831154	-2.370232	-1.860761
1	-0.860757	0.560145	-1.265934

In [582]:

```
pd.concat([df1, df2], ignore_index=True)
```

Out[582]:

	a	b	c	d
0	-0.438570	-0.539741	0.476985	3.248944
1	-1.021228	-0.577087	0.124121	0.302614
2	0.523772	0.000940	1.343810	-0.713544
3	-1.860761	-0.831154	NaN	-2.370232
4	-1.265934	-0.860757	NaN	0.560145

In [583]:

```
a = pd.Series([np.nan, 2.5, np.nan, 3.5, 4.5, np.nan], index=['f', 'e', 'd', 'c', 'b', 'a'])
```

In [584]:

```
b = pd.Series(np.arange(len(a)), dtype=np.float64), index=['f', 'e', 'd', 'c', 'b', 'a'])
```

In [585]:

```
b[-1] = np.nan
```

In [586]:

```
a
```

Out[586]:

```
f      NaN
e      2.5
d      NaN
c      3.5
b      4.5
a      NaN
dtype: float64
```

In [587]:

```
b
```

Out[587]:

```
f      0.0
e      1.0
d      2.0
c      3.0
b      4.0
a      NaN
dtype: float64
```

In [588]:

```
np.where(pd.isnull(a), b, a)
```

Out[588]:

```
array([0. , 2.5, 2. , 3.5, 4.5, nan])
```

In [589]:

```
b[:-2].combine_first(a[2:])
```

Out[589]:

```
a      NaN
b      4.5
c      3.0
d      2.0
e      1.0
f      0.0
dtype: float64
```

In [590]:

```
df1 = pd.DataFrame({'a': [1., np.nan, 5., np.nan],
                    'b': [np.nan, 2., np.nan, 6.],
                    'c': range(2, 18, 4)})
```

In [591]:

```
df2 = pd.DataFrame({'a': [5., 4., np.nan, 3., 7.],
                    'b': [np.nan, 3., 4., 6., 8.]})
```

In [592]:

```
df1
```

Out[592]:

	a	b	c
0	1.0	NaN	2
1	NaN	2.0	6
2	5.0	NaN	10
3	NaN	6.0	14

In [593]:

```
df2
```

Out[593]:

	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 [594]:

```
df1.combine_first(df2)
```

Out[594]:

	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 Reshaping and Pivoting

In [595]:

```
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 [596]:

```
data
```

Out[596]:

```
number  one  two  three
```

```
state
```

state	number	one	two	three
Ohio	0	1	2	
Colorado	3	4	5	

In [597]:

```
result = data.stack()
```

In [598]:

```
result
```

Out[598]:

```
state      number
Ohio      one        0
          two        1
          three      2
Colorado  one        3
          two        4
          three      5
dtype: int32
```

In [599]:

```
result.unstack()
```

Out[599]:

```
number  one  two  three
```

```
state
```

state	number	one	two	three
Ohio	0	1	2	
Colorado	3	4	5	

In [600]:

```
result.unstack(0)
```

Out[600]:

	state	Ohio	Colorado
number			
one	0	3	
two	1	4	
three	2	5	

In [601]:

```
result.unstack('state')
```

Out[601]:

	state	Ohio	Colorado
number			
one	0	3	
two	1	4	
three	2	5	

In [602]:

```
s1 = pd.Series([0, 1, 2, 3], index=['a', 'b', 'c', 'd'])
```

In [603]:

```
s2 = pd.Series([4, 5, 6], index=['c', 'd', 'e'])
```

In [604]:

```
data2 = pd.concat([s1, s2], keys=['one', 'two'])
```

In [605]:

```
data2
```

Out[605]:

one	a	0
	b	1
	c	2
	d	3
two	c	4
	d	5
	e	6
		dtype: int64

In [606]:

```
data2.unstack()
```

Out[606]:

	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 [607]:

```
data2.unstack()
```

Out[607]:

	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 [608]:

```
data2.unstack().stack()
```

Out[608]:

```
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 [609]:

```
data2.unstack().stack(dropna=False)
```

Out[609]:

```
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
```

In [610]:

```
df = pd.DataFrame({'left': result, 'right': result + 5},
                  columns=pd.Index(['left', 'right'], name='side'))
```

In [611]:

df

Out[611]:

	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 [612]:

df.unstack('state')

Out[612]:

	side	left	right	
state	Ohio	Colorado	Ohio	Colorado
number				
one	0	3	5	8
two	1	4	6	9
three	2	5	7	10

In [613]:

df.unstack('state').stack('side')

Out[613]:

	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

In [614]:

data = pd.read_csv('examples/macrodata.csv')

In [615]:

```
data.head()
```

Out[615]:

	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbilrate	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 [616]:

```
periods = pd.PeriodIndex(year=data.year, quarter=data.quarter, name='date')
```

In [617]:

```
columns = pd.Index(['realgdp', 'infl', 'unemp'], name='item')
```

In [618]:

```
data = data.reindex(columns=columns)
```

In [619]:

```
data.index = periods.to_timestamp('D', 'end')
```

In [620]:

```
ldata = data.stack().reset_index().rename(columns={0: 'value'})
```

In [621]:

ladata[:10]

Out[621]:

	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

In [622]:

pivoted = ladata.pivot('date', 'item', 'value')

In [623]:

pivoted

Out[623]:

	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

In [624]:

ladata['value2'] = np.random.randn(len(ladata))

In [625]:

ladata[:10]

Out[625]:

	date	item	value	value2
0	1959-03-31 23:59:59.999999999	realgdp	2710.349	0.119827
1	1959-03-31 23:59:59.999999999	infl	0.000	-1.063512
2	1959-03-31 23:59:59.999999999	unemp	5.800	0.332883
3	1959-06-30 23:59:59.999999999	realgdp	2778.801	-2.359419
4	1959-06-30 23:59:59.999999999	infl	2.340	-0.199543
5	1959-06-30 23:59:59.999999999	unemp	5.100	-1.541996
6	1959-09-30 23:59:59.999999999	realgdp	2775.488	-0.970736
7	1959-09-30 23:59:59.999999999	infl	2.740	-1.307030
8	1959-09-30 23:59:59.999999999	unemp	5.300	0.286350
9	1959-12-31 23:59:59.999999999	realgdp	2785.204	0.377984

In [626]:

pivoted = ladata.pivot('date', 'item')

In [627]:

pivoted[:5]

Out[627]:

	date	value				value2		
		item	infl	realgdp	unemp	infl	realgdp	unemp
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8	-1.063512	0.119827	0.332883		
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1	-0.199543	-2.359419	-1.541996		
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3	-1.307030	-0.970736	0.286350		
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6	-0.753887	0.377984	0.331286		
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2	0.069877	1.349742	0.246674		

In [628]:

```
pivoted['value'][:5]
```

Out[628]:

	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	

In [629]:

```
unstacked = ldata.set_index(['date', 'item']).unstack('item')
```

In [630]:

```
unstacked[:7]
```

Out[630]:

	value				value2		
	item	infl	realgdp	unemp	infl	realgdp	unemp
	date						
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8	-1.063512	0.119827	0.332883	
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1	-0.199543	-2.359419	-1.541996	
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3	-1.307030	-0.970736	0.286350	
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6	-0.753887	0.377984	0.331286	
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2	0.069877	1.349742	0.246674	
1960-06-30 23:59:59.999999999	0.14	2834.390	5.2	1.004812	-0.011862	1.327195	
1960-09-30 23:59:59.999999999	2.70	2839.022	5.6	-1.549106	-0.919262	0.022185	

In [631]:

```
df = pd.DataFrame({'key': ['foo', 'bar', 'baz'],
                   'A': [1, 2, 3],
                   'B': [4, 5, 6],
                   'C': [7, 8, 9]})
```

In [632]:

```
df
```

Out[632]:

	key	A	B	C
0	foo	1	4	7
1	bar	2	5	8
2	baz	3	6	9

In [633]:

```
melted = pd.melt(df, ['key'])
```

In [634]:

```
melted
```

Out[634]:

	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

In [635]:

```
reshaped = melted.pivot('key', 'variable', 'value')
```

In [636]:

```
reshaped
```

Out[636]:

	variable	A	B	C
key				
bar	2	5	8	
baz	3	6	9	
foo	1	4	7	

In [637]:

```
reshaped.reset_index()
```

Out[637]:

variable	key	A	B	C
0	bar	2	5	8
1	baz	3	6	9
2	foo	1	4	7

In [638]:

```
pd.melt(df, id_vars=['key'], value_vars=['A', 'B'])
```

Out[638]:

	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

In [639]:

```
pd.melt(df, value_vars=['A', 'B', 'C'])
```

Out[639]:

	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 [640]:

```
pd.melt(df, value_vars=['key', 'A', 'B'])
```

Out[640]:

	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

9. PLOTTING AND VISUALIZATION

9.1 A Brief matplotlib API Primer

In [343]:

```
%matplotlib notebook
```

In [344]:

```
import matplotlib.pyplot as plt
```

In [345]:

```
import numpy as np
```

In [346]:

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

In [347]:

```
data
```

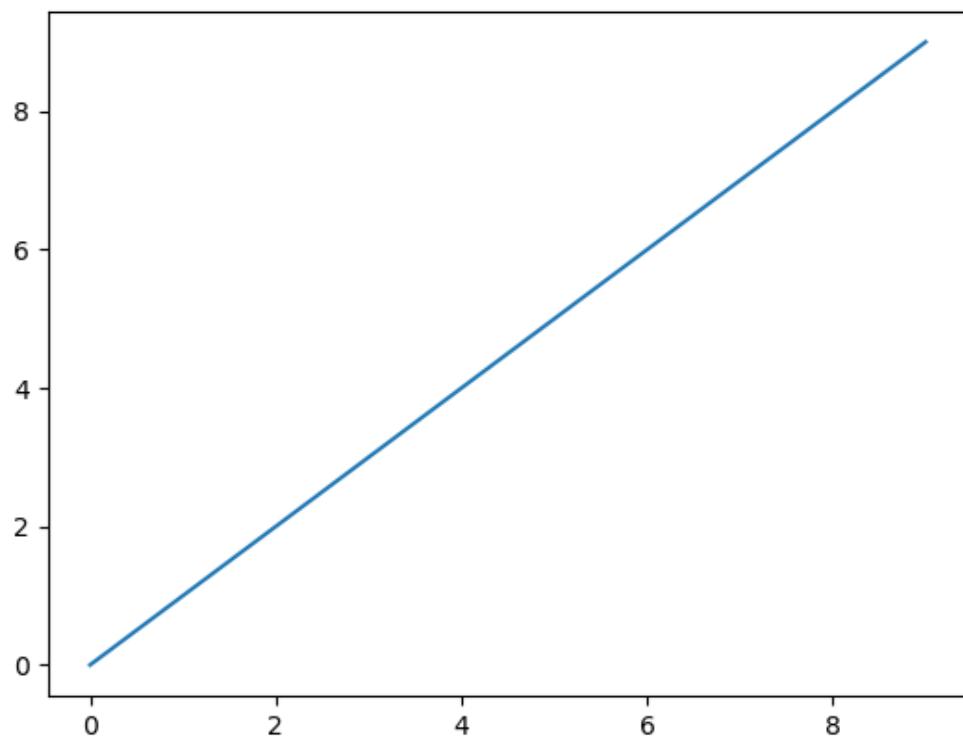
Out[347]:

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

In [348]:

```
plt.plot(data)
```

<IPython.core.display.Javascript object>



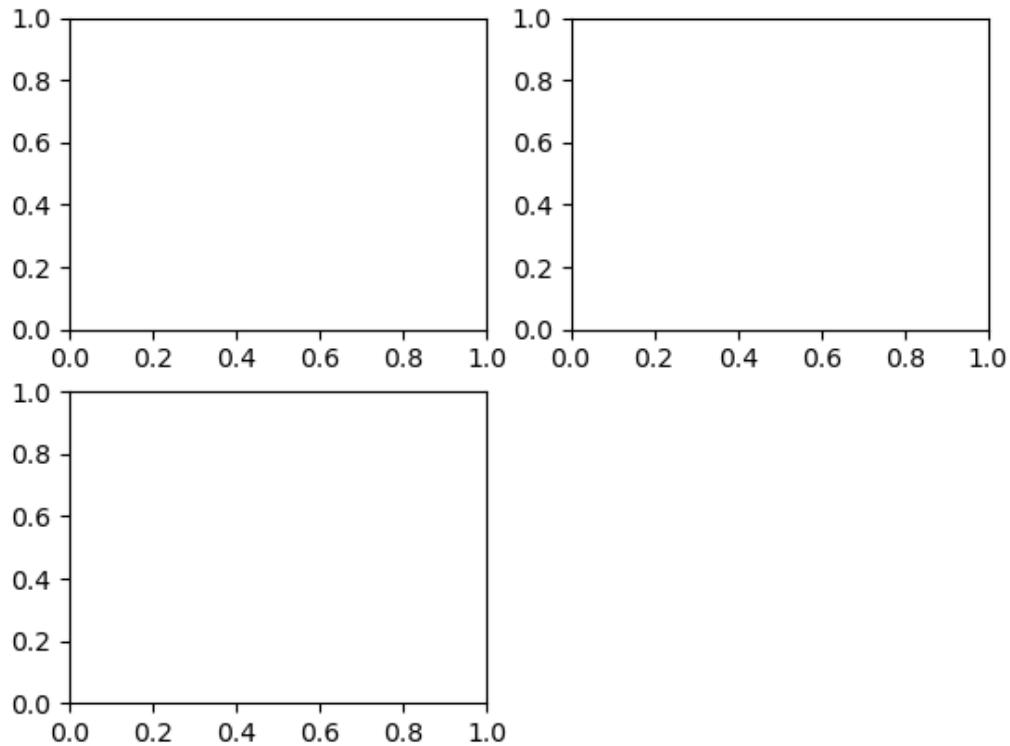
Out[348]:

```
[<matplotlib.lines.Line2D at 0x1b412b1d888>]
```

In [349]:

```
fig = plt.figure()
```

```
<IPython.core.display.Javascript object>
```



In [350]:

```
ax1 = fig.add_subplot(2, 2, 1)
```

In [351]:

```
ax2 = fig.add_subplot(2, 2, 2)
```

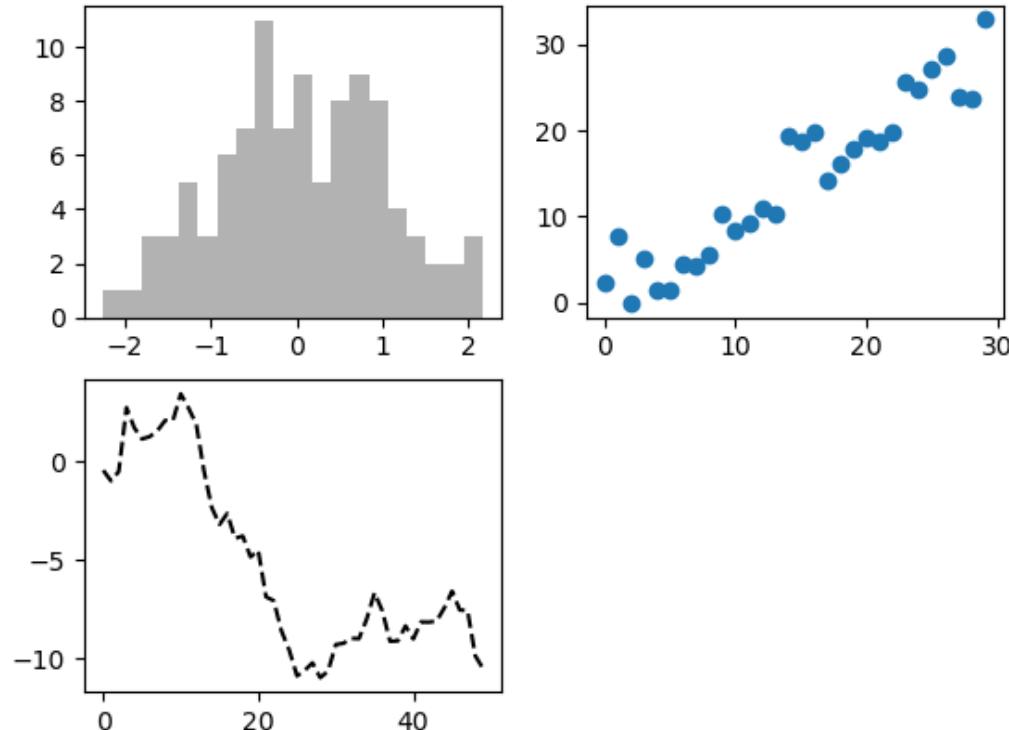
In [352]:

```
ax3 = fig.add_subplot(2, 2, 3)
```

In [353]:

```
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
```

<IPython.core.display.Javascript object>



In [354]:

```
plt.plot(np.random.randn(50).cumsum(), 'k--')
```

Out[354]:

[<matplotlib.lines.Line2D at 0x1b413d00888>]

In [355]:

```
fig.add_subplot
```

Out[355]:

<bound method Figure.add_subplot of <Figure size 640x480 with 3 Axes>>

In [356]:

```
ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)
```

Out[356]:

```
(array([ 1.,  1.,  3.,  3.,  5.,  3.,  6.,  7., 11.,  7.,  9.,  5.,  8.,
       9.,  8.,  4.,  3.,  2.,  2.,  3.]),
 array([-2.25279725, -2.03168431, -1.81057138, -1.58945844, -1.36834551,
        -1.14723258, -0.92611964, -0.70500671, -0.48389377, -0.26278084,
        -0.0416679 ,  0.17944503,  0.40055796,  0.6216709 ,  0.84278383,
       1.06389677,  1.2850097 ,  1.50612264,  1.72723557,  1.9483485 ,
      2.16946144]),  
<a list of 20 Patch objects>)
```

In [357]:

```
ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```

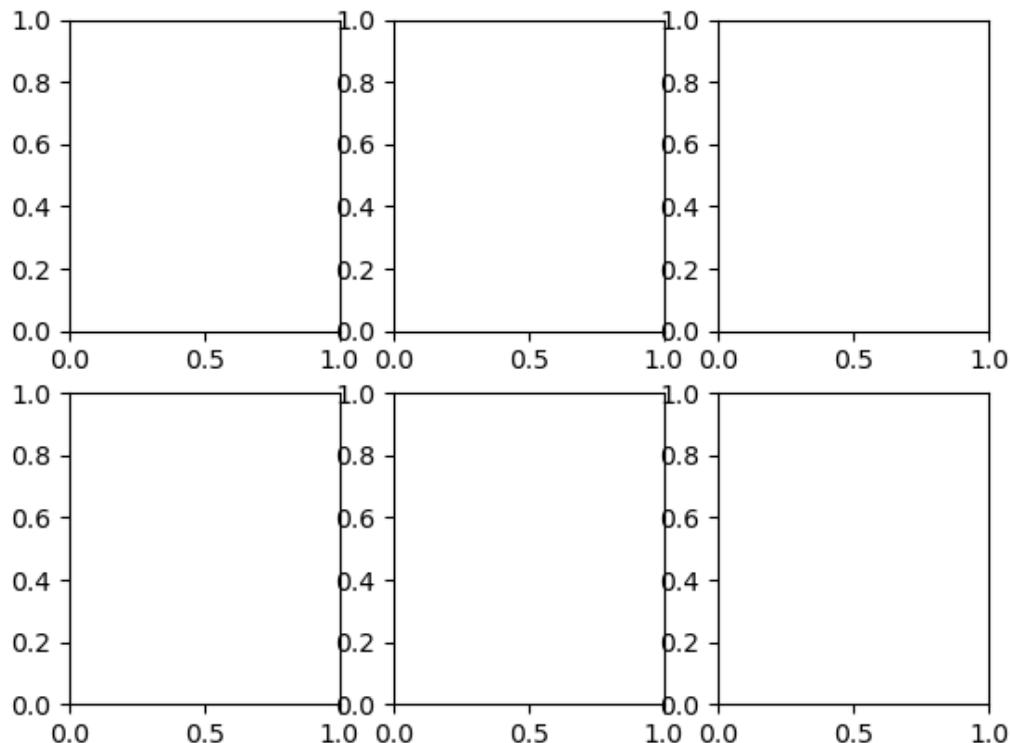
Out[357]:

```
<matplotlib.collections.PathCollection at 0x1b413d3a0c8>
```

In [358]:

```
fig, axes = plt.subplots(2, 3)
```

```
<IPython.core.display.Javascript object>
```



In [359]:

axes

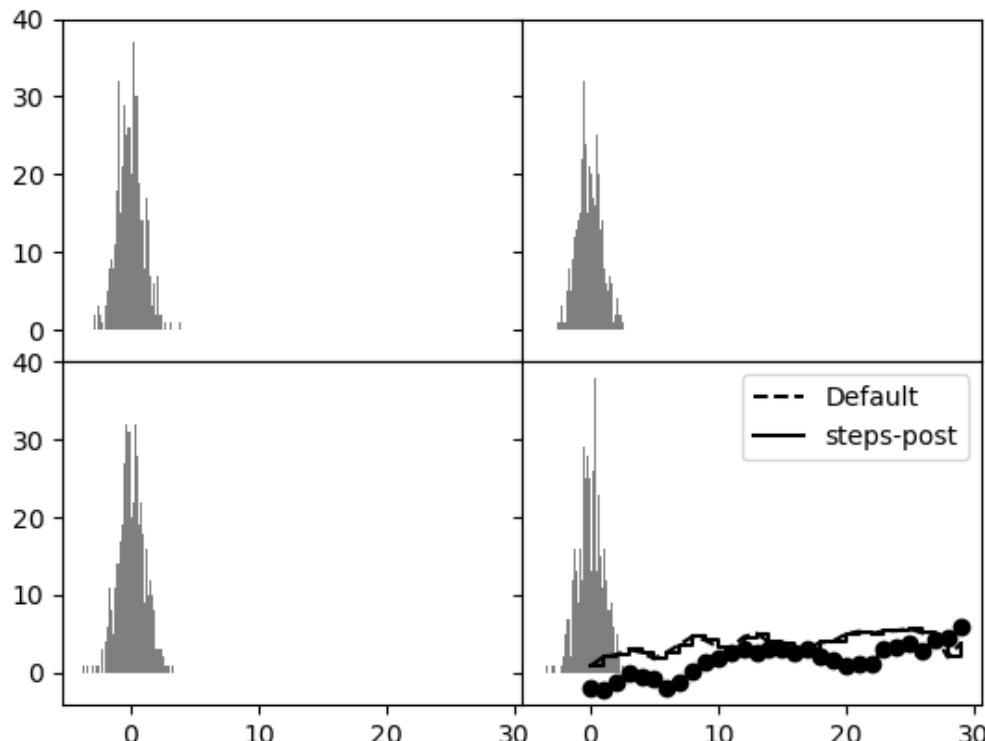
Out[359]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001B413D0B888
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001B413D730C8
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001B413DA7808
>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x000001B413DE2548
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001B413E1C448
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001B413E55388
>]],
      dtype=object)
```

In [361]:

```
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)
```

<IPython.core.display.Javascript object>



Matplotlib'in ana çizim işlevi, x ve y koordinatlarının dizilerini ve istege bağlı olarak renk ve çizgi stilini gösteren dize kısaltması aşağıdaki gibidir. Örneğin yeşil çizgilerle, x'e karşı y'yi çizmek için şunları yaparız:

```
ax.plot(x, y, 'g--')
ax.plot(x, y, linestyle='--', color='g')
```

In [362]:

```
from numpy.random import randn
```

In [363]:

```
plt.plot(randn(30).cumsum(), 'ko--')
```

Out[363]:

```
[<matplotlib.lines.Line2D at 0x1b41478f1c8>]
```

Yukarıdaki kodun daha açık hali:

```
plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
```

In [365]:

```
data = np.random.randn(30).cumsum()
```

In [366]:

```
plt.plot(data, 'k--', label='Default')
```

Out[366]:

```
[<matplotlib.lines.Line2D at 0x1b4147ae8c8>]
```

In [367]:

```
plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')
```

Out[367]:

```
[<matplotlib.lines.Line2D at 0x1b4147b12c8>]
```

In [368]:

```
plt.legend(loc='best')
```

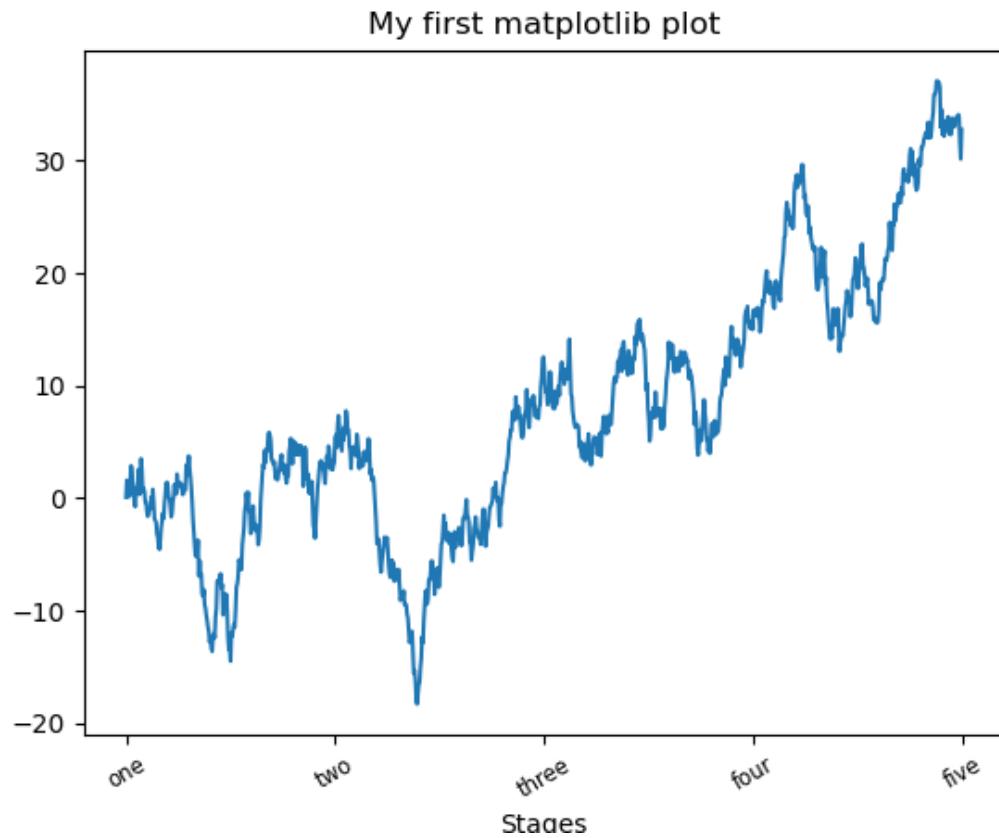
Out[368]:

```
<matplotlib.legend.Legend at 0x1b4147c1c48>
```

In [369]:

```
fig = plt.figure()
```

```
<IPython.core.display.Javascript object>
```



In [370]:

```
ax = fig.add_subplot(1, 1, 1)
```

In [371]:

```
ax.plot(np.random.randn(1000).cumsum())
```

Out[371]:

```
[<matplotlib.lines.Line2D at 0x1b414a84ac8>]
```

In [372]:

```
ticks = ax.set_xticks([0, 250, 500, 750, 1000])
```

In [373]:

```
labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'],
                           rotation=30, fontsize='small')
```

In [374]:

```
ax.set_title('My first matplotlib plot')
```

Out[374]:

```
Text(0.5, 1.0, 'My first matplotlib plot')
```

In [375]:

```
ax.set_xlabel('Stages')
```

Out[375]:

```
Text(0.5, 10.888891973024519, 'Stages')
```

In [376]:

```
props = {
    'title': 'My first matplotlib plot',
    'xlabel': 'Stages'
}
ax.set(**props)
```

Out[376]:

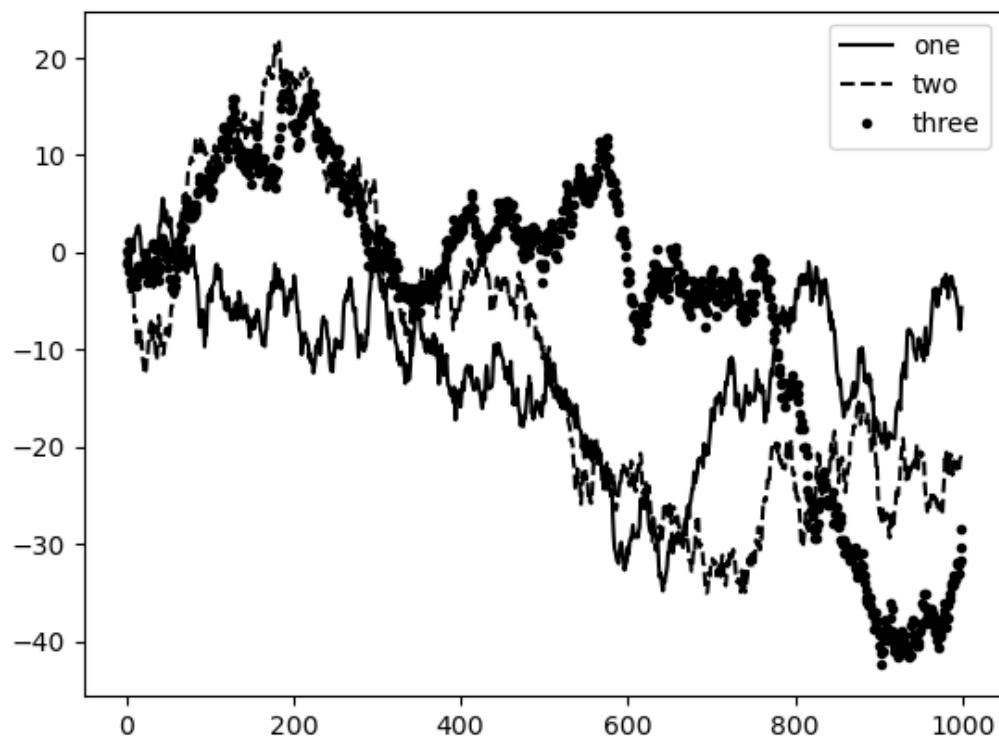
```
[Text(0.5, 10.888891973024519, 'Stages'),
 Text(0.5, 1.0, 'My first matplotlib plot')]
```

In [377]:

```
from numpy.random import randn
```

In [378]:

```
fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)  
<IPython.core.display.Javascript object>
```



In [379]:

```
ax.plot(randn(1000).cumsum(), 'k', label='one')
```

Out[379]:

```
[<matplotlib.lines.Line2D at 0x1b414d80f08>]
```

In [380]:

```
ax.plot(randn(1000).cumsum(), 'k--', label='two')
```

Out[380]:

```
[<matplotlib.lines.Line2D at 0x1b414d8dec8>]
```

In [381]:

```
ax.plot(randn(1000).cumsum(), 'k.', label='three')
```

Out[381]:

```
[<matplotlib.lines.Line2D at 0x1b414d8fb48>]
```

In [382]:

```
ax.legend(loc='best')
```

Out[382]:

```
<matplotlib.legend.Legend at 0x1b414d849c8>
```

Standart çizim türlerine ek olarak, kendi notlarınızı da çizmek isteyebilirsiniz. Bunlar metin, oklar veya diğer şekillerden oluşabilir. Metin, ok ve açıklama işlevlerini kullanarak ek açıklamalar ekleyebiliriz:

```
ax.text(x, y, 'Hello world!',  
        family='monospace', fontsize=10)
```

In [384]:

```
from datetime import datetime
```

In [385]:

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

<IPython.core.display.Javascript object>
```



In [386]:

```
data = pd.read_csv('examples/spx.csv', index_col=0, parse_dates=True)
spx = data['SPX']
```

In [387]:

```
spx.plot(ax=ax, style='k-')
```

Out[387]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b414aaef48>
```

In [388]:

```
crisis_data = [
    (datetime(2007, 10, 11), 'Peak of bull market'),
    (datetime(2008, 3, 12), 'Bear Stearns Fails'),
    (datetime(2008, 9, 15), 'Lehman Bankruptcy')
]
```

In [389]:

```
for date, label in crisis_data:
    ax.annotate(label, xy=(date, spx.asof(date) + 75),
               xytext=(date, spx.asof(date) + 225),
               arrowprops=dict(facecolor='black', headwidth=4, width=2,
                               headlength=4),
               horizontalalignment='left', verticalalignment='top')
```

In [390]:

```
ax.set_xlim(['1/1/2007', '1/1/2011'])
ax.set_ylim([600, 1800])
```

Out[390]:

```
(600.0, 1800.0)
```

In [391]:

```
ax.set_title('Important dates in the 2008-2009 financial crisis')
```

Out[391]:

```
Text(0.5, 1.0, 'Important dates in the 2008-2009 financial crisis')
```

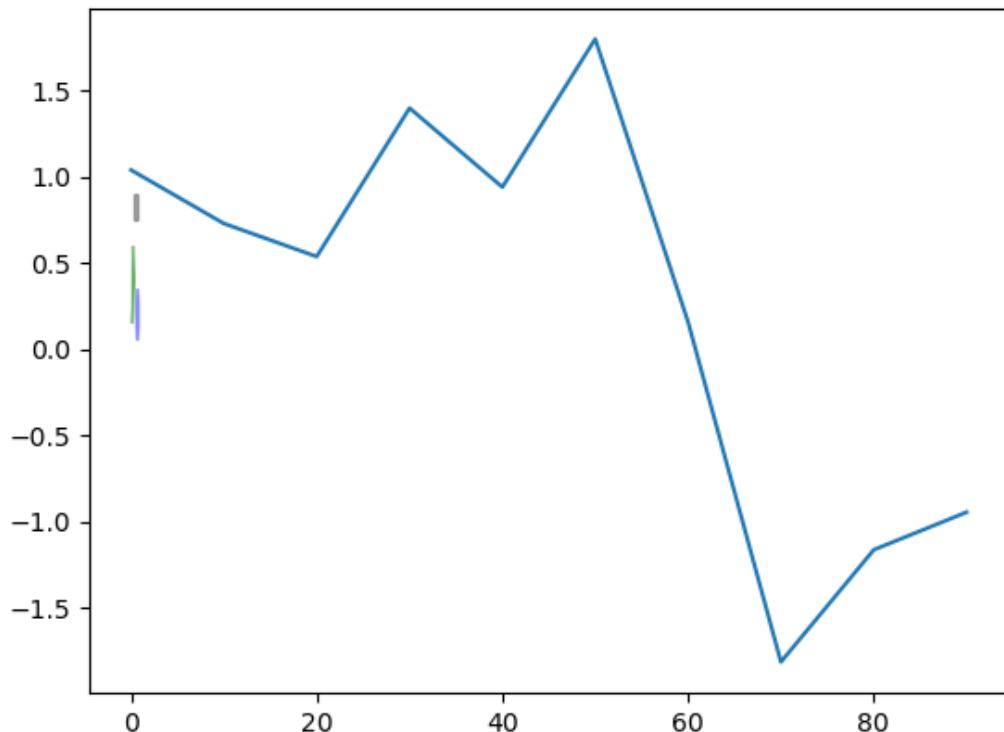
In [392]:

```
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)
```

<IPython.core.display.Javascript object>



Out[392]:

```
<matplotlib.patches.Polygon at 0x1b415165588>
```

In [393]:

```
plt.savefig('figpath.svg')
```

In [394]:

```
plt.savefig('figpath.png', dpi=400, bbox_inches='tight')
```

In [395]:

```
from io import BytesIO
buffer = BytesIO()
plt.savefig(buffer)
plot_data = buffer.getvalue()
```

In [396]:

```
plt.rc('figure', figsize=(10, 10))
```

Yukarıdaki kodu daha ayrıntılılarıyla özelleştirmek istersek:

```
font_options = {'family' : 'monospace',
                'weight' : 'bold',
                'size' : 'small'}
plt.rc('font', **font_options)
```

9.2 Plotting with pandas and seaborn

In [397]:

```
s = pd.Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10))
```

In [398]:

```
s.plot()
```

Out[398]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b415120b88>
```

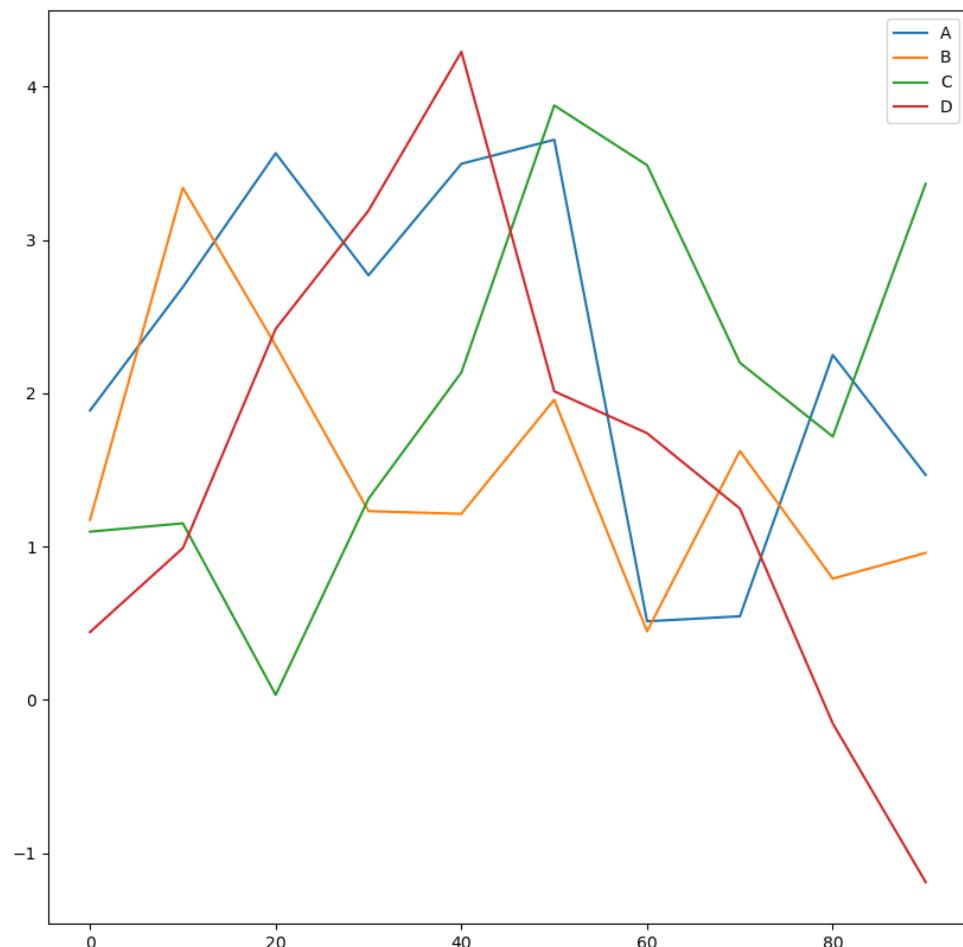
In [399]:

```
df = pd.DataFrame(np.random.randn(10, 4).cumsum(0),
                  columns=['A', 'B', 'C', 'D'],
                  index=np.arange(0, 100, 10))
```

In [400]:

```
df.plot()
```

<IPython.core.display.Javascript object>



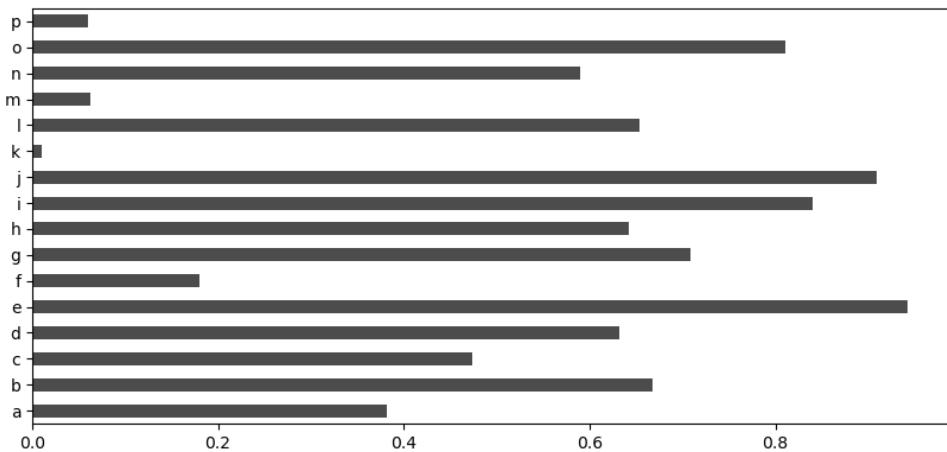
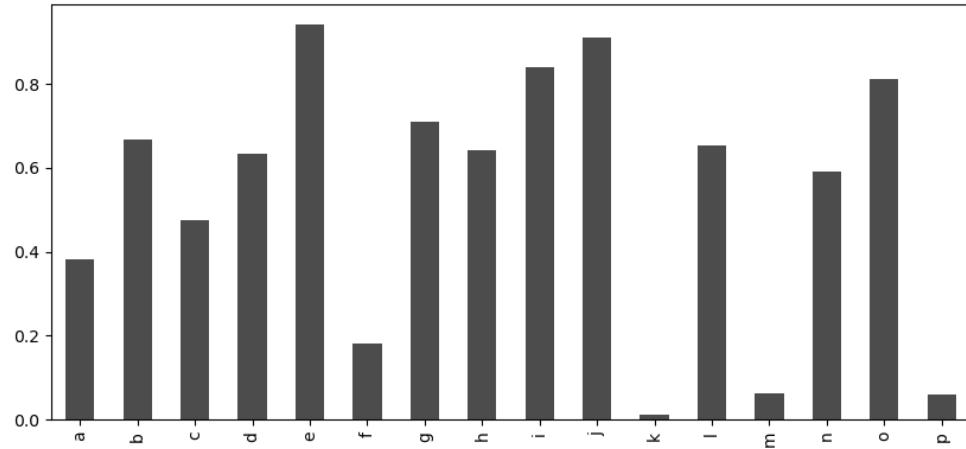
Out[400]:

<matplotlib.axes._subplots.AxesSubplot at 0x1b41542f588>

In [401]:

```
fig, axes = plt.subplots(2, 1)
```

<IPython.core.display.Javascript object>



In [402]:

```
data = pd.Series(np.random.rand(16), index=list('abcdefghijklmno'))
```

In [403]:

```
data.plot.bar(ax=axes[0], color='k', alpha=0.7)
```

Out[403]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b4154b9a08>
```

In [404]:

```
data.plot.barh(ax=axes[1], color='k', alpha=0.7)
```

Out[404]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b4151269c8>
```

In [405]:

```
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 [406]:

```
df
```

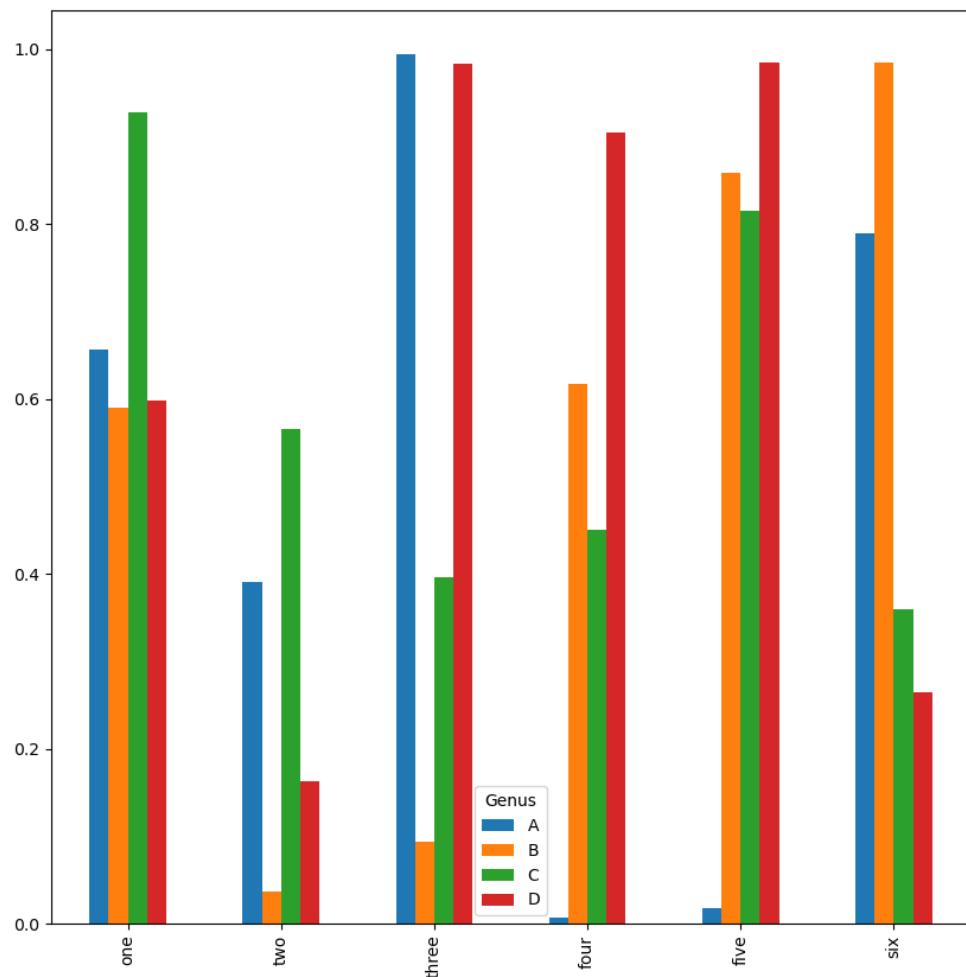
Out[406]:

Genus	A	B	C	D
one	0.656249	0.590225	0.927405	0.597770
two	0.390140	0.036328	0.565980	0.162559
three	0.994786	0.094264	0.395706	0.983019
four	0.006279	0.617671	0.449974	0.904357
five	0.017474	0.858471	0.815522	0.984996
six	0.789212	0.984949	0.359175	0.264507

In [407]:

df.plot.bar()

<IPython.core.display.Javascript object>



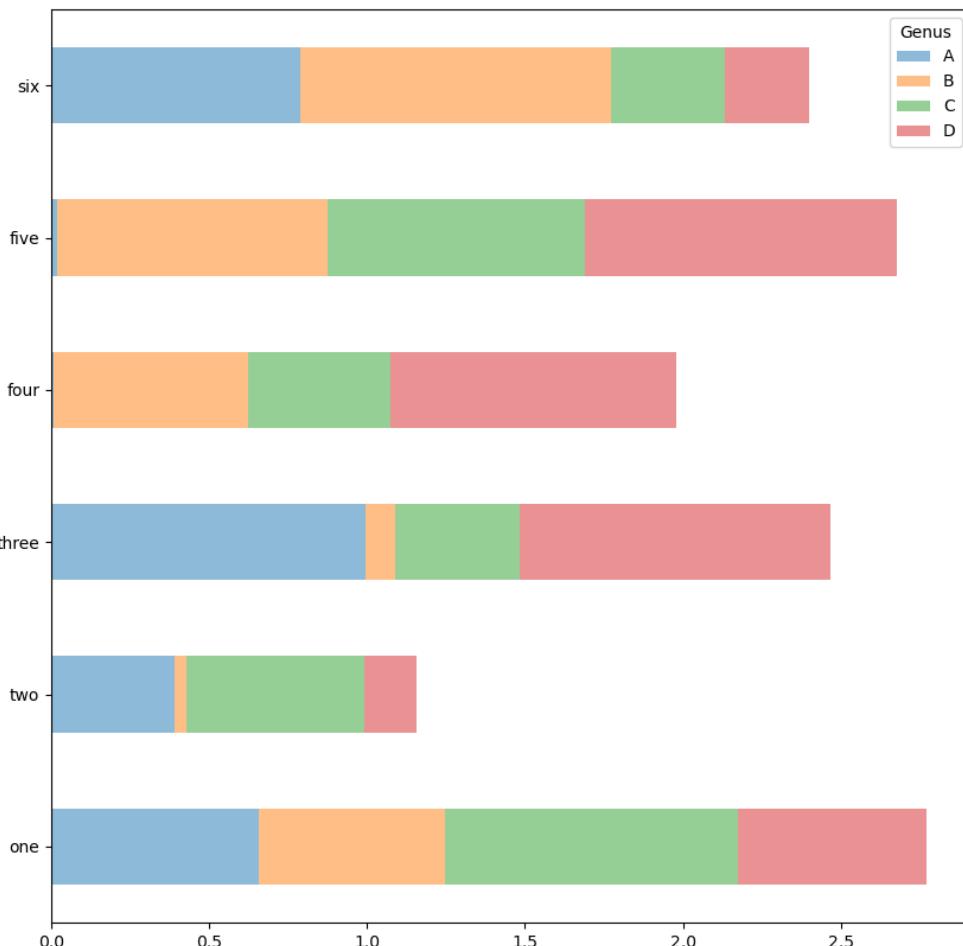
Out[407]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b4155abf48>
```

In [408]:

```
df.plot.barh(stacked=True, alpha=0.5)
```

```
<IPython.core.display.Javascript object>
```



Out[408]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b4156ae808>
```

In [409]:

```
tips = pd.read_csv('examples/tips.csv')
```

In [410]:

```
party_counts = pd.crosstab(tips['day'], tips['size'])
```

In [411]:

```
party_counts
```

Out[411]:

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 [412]:

```
party_counts = party_counts.loc[:, 2:5]
```

In [413]:

```
party_pcts = party_counts.div(party_counts.sum(1), axis=0)
```

In [414]:

```
party_pcts
```

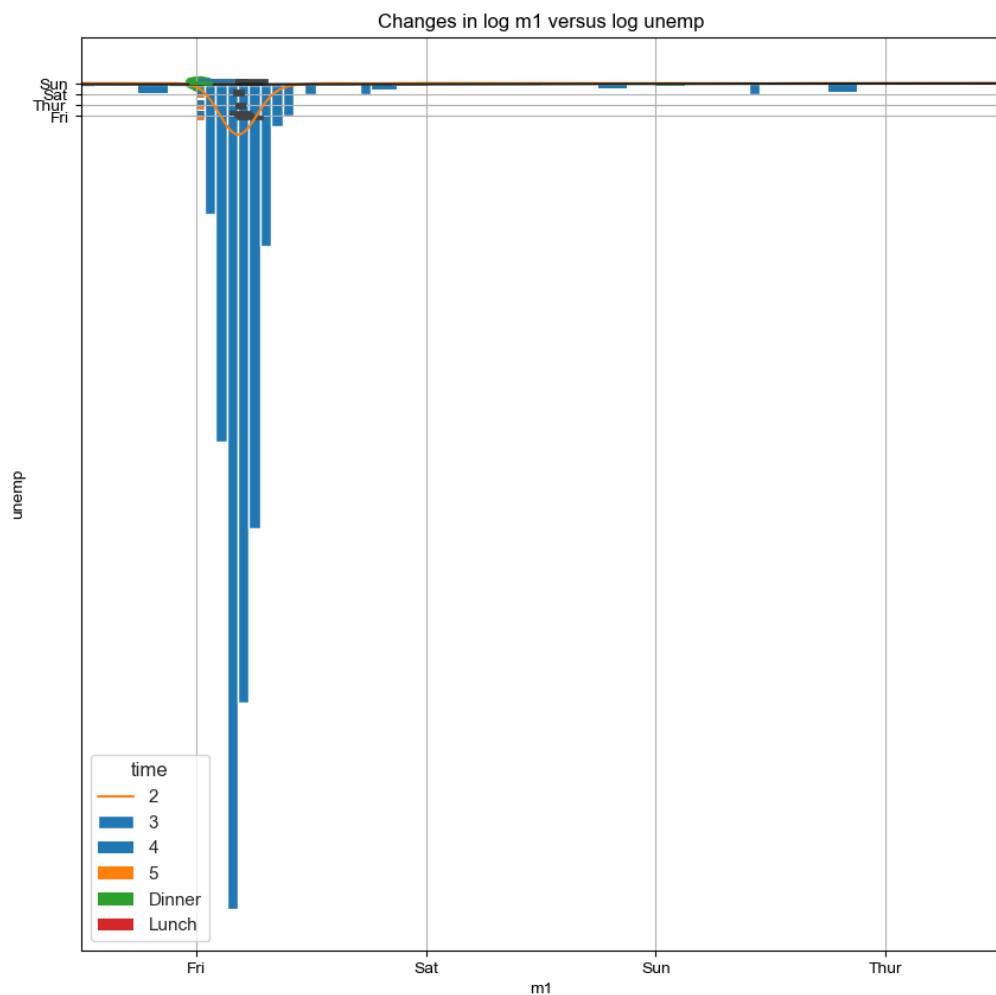
Out[414]:

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 [415]:

```
party_pcts.plot.bar()
```

<IPython.core.display.Javascript object>



Out[415]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b41869df48>
```

In [416]:

```
import seaborn as sns
```

In [417]:

```
tips['tip_pct'] = tips['tip'] / (tips['total_bill'] - tips['tip'])
```

In [418]:

```
tips.head()
```

Out[418]:

	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 [419]:

```
sns.barplot(x='tip_pct', y='day', data=tips, orient='h')
```

Out[419]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b41869df48>
```

In [420]:

```
sns.barplot(x='tip_pct', y='day', hue='time', data=tips, orient='h')
```

Out[420]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b41869df48>
```

In [421]:

```
sns.set(style="whitegrid")
```

In [422]:

```
tips['tip_pct'].plot.hist(bins=50)
```

Out[422]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b41869df48>
```

In [423]:

```
tips['tip_pct'].plot.density()
```

Out[423]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b41869df48>
```

In [424]:

```
comp1 = np.random.normal(0, 1, size=200)
```

In [425]:

```
comp2 = np.random.normal(10, 2, size=200)
```

In [426]:

```
values = pd.Series(np.concatenate([comp1, comp2]))
```

In [427]:

```
sns.distplot(values, bins=100, color='k')
```

Out[427]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b41869df48>
```

In [428]:

```
macro = pd.read_csv('examples/macrodata.csv')
```

In [429]:

```
data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]
```

In [430]:

```
trans_data = np.log(data).diff().dropna()
```

In [431]:

```
trans_data[-5:]
```

Out[431]:

	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 [432]:

```
sns.regplot('m1', 'unemp', data=trans_data)
```

Out[432]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b41869df48>
```

In [433]:

```
plt.title('Changes in log %s versus log %s' % ('m1', 'unemp'))
```

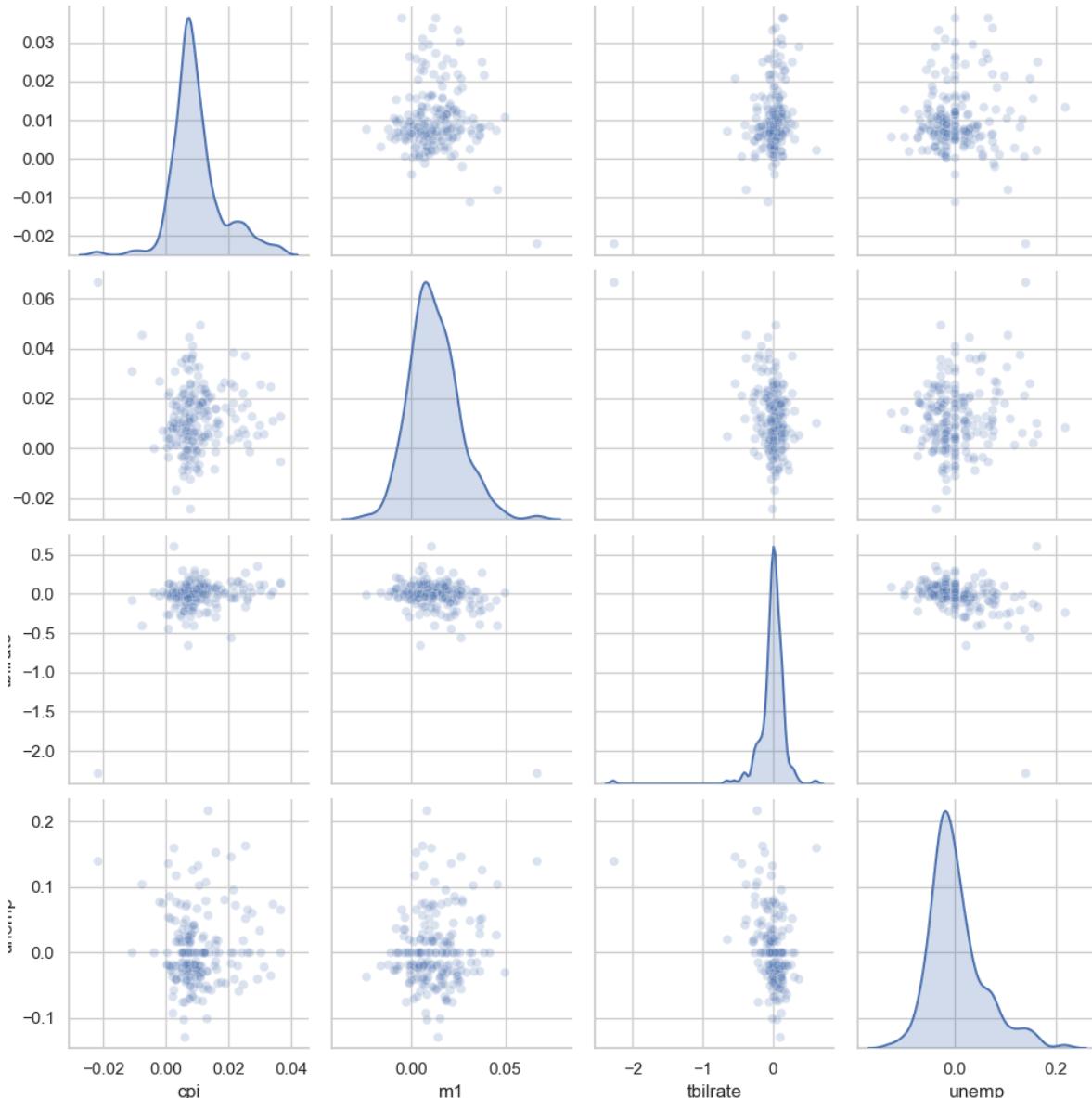
Out[433]:

```
Text(0.5, 1.0, 'Changes in log m1 versus log unemp')
```

In [434]:

```
sns.pairplot(trans_data, diag_kind='kde', plot_kws={'alpha': 0.2})
```

<IPython.core.display.Javascript object>



Out[434]:

```
<seaborn.axisgrid.PairGrid at 0x1b41a3a8048>
```

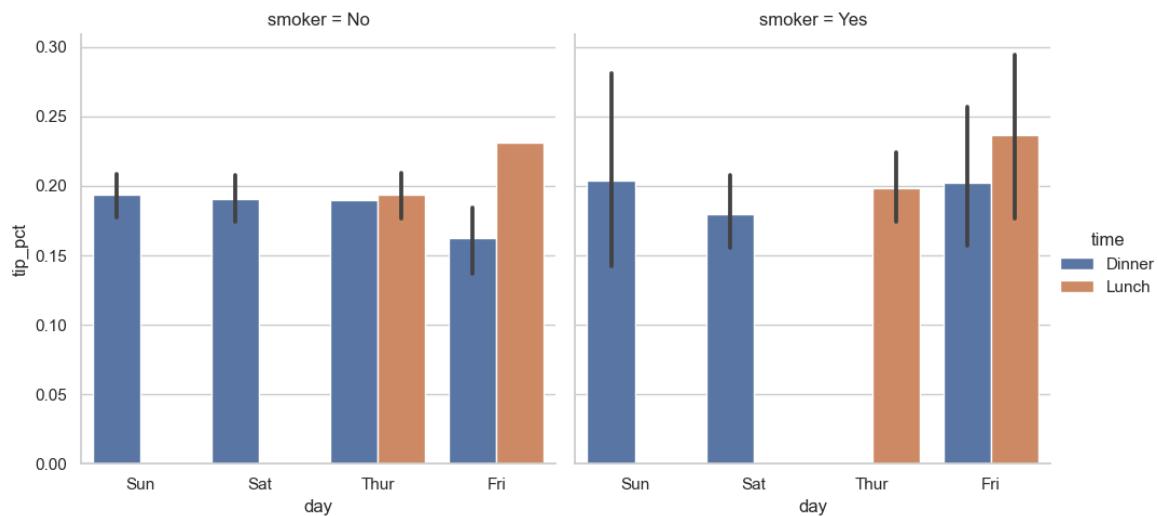
In [435]:

```
sns.factorplot(x='day', y='tip_pct', hue='time', col='smoker',
                 kind='bar', data=tips[tips.tip_pct < 1])
```

C:\Users\Kader\anaconda3\lib\site-packages\seaborn\categorical.py:3666: User Warning: 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 to `'strip'` in `catplot`.

```
warnings.warn(msg)
```

<IPython.core.display.Javascript object>



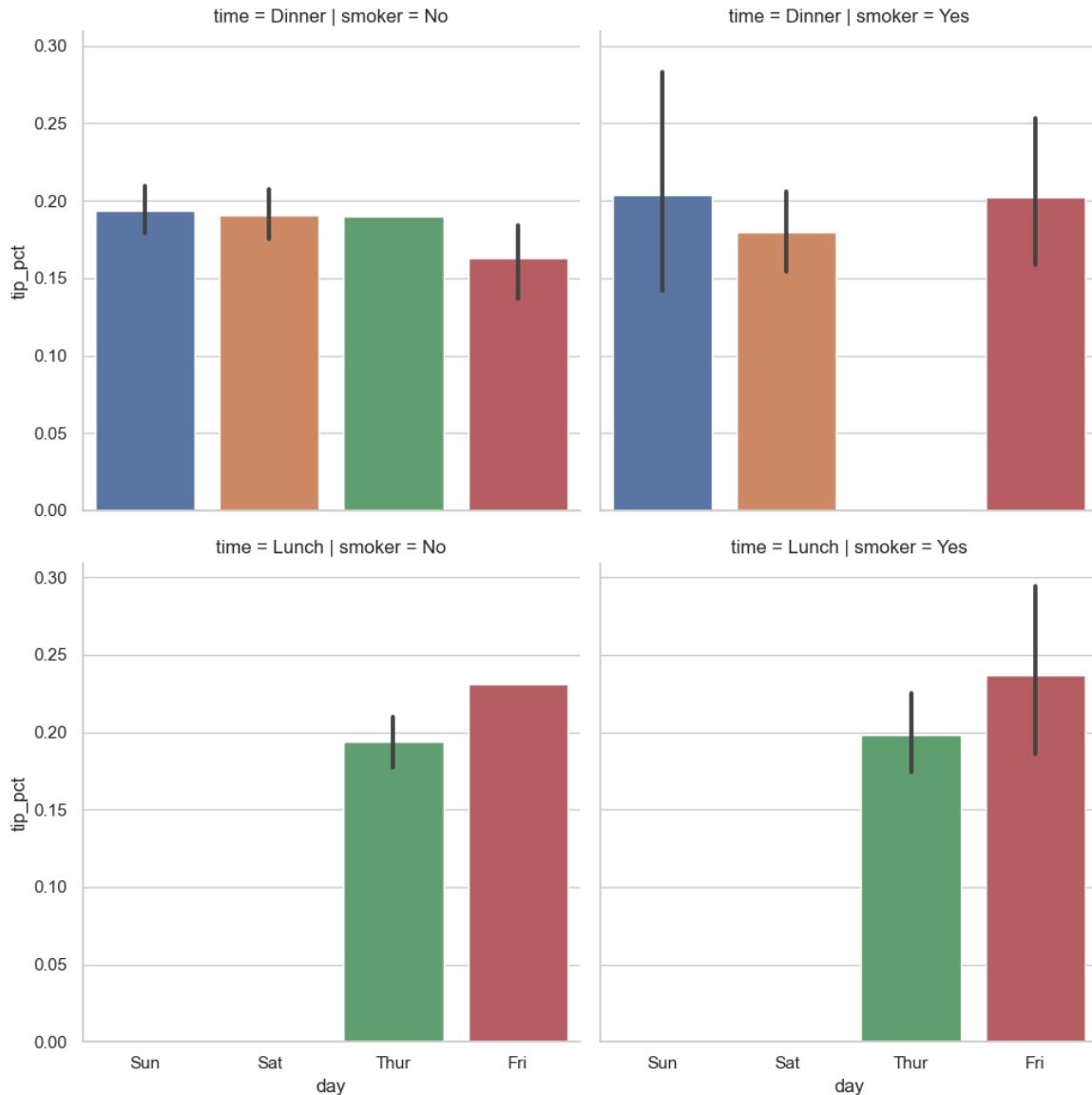
Out[435]:

<seaborn.axisgrid.FacetGrid at 0x1b41b1a8bc8>

In [436]:

```
sns.factorplot(x='day', y='tip_pct', row='time',
                 col='smoker',
                 kind='bar', data=tips[tips.tip_pct < 1])
```

<IPython.core.display.Javascript object>



Out[436]:

```
<seaborn.axisgrid.FacetGrid at 0x1b41b6a4b48>
```

In [437]:

```
sns.factorplot(x='tip_pct', y='day', kind='box',
                 data=tips[tips.tip_pct < 0.5])
```

...

10. DATA AGGREGATION AND GROUP OPERATIONS

10.1 GroupBy Mechanics

In [438]:

```
import numpy as np
```

In [439]:

```
import pandas as pd
```

In [440]:

```
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 [441]:

```
df
```

Out[441]:

	key1	key2	data1	data2
0	a	one	-0.597947	-0.871901
1	a	two	1.262680	-1.906540
2	b	one	1.086641	-0.331425
3	b	two	-0.150475	-0.889146
4	a	one	0.278976	-0.477823

In [442]:

```
grouped = df['data1'].groupby(df['key1'])
```

In [443]:

```
grouped
```

Out[443]:

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x000001B41B58E5C8>
```

In [444]:

```
grouped.mean()
```

Out[444]:

```
key1
a    0.314570
b    0.468083
Name: data1, dtype: float64
```

In [445]:

```
means = df['data1'].groupby([df['key1'], df['key2']]).mean()
```

In [446]:

```
means
```

Out[446]:

```
key1  key2
a    one    -0.159485
      two     1.262680
b    one     1.086641
      two    -0.150475
Name: data1, dtype: float64
```

In [447]:

```
means.unstack()
```

Out[447]:

key2	one	two
key1		
a	-0.159485	1.262680
b	1.086641	-0.150475

In [448]:

```
states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
```

In [449]:

```
years = np.array([2005, 2005, 2006, 2005, 2006])
```

In [450]:

```
df['data1'].groupby([states, years]).mean()
```

Out[450]:

```
California 2005    1.262680
              2006    1.086641
Ohio        2005    -0.374211
              2006    0.278976
Name: data1, dtype: float64
```

In [751]:

```
df.groupby('key1').mean()
```

Out[751]:

	data1	data2
--	-------	-------

key1

a	1.336344	-0.704733
b	-0.574513	-1.809366

In [752]:

```
df.groupby(['key1', 'key2']).mean()
```

Out[752]:

	data1	data2
--	-------	-------

key1 key2

a	one	1.170309	-0.483002
a	two	1.668416	-1.148195
b	one	-0.149717	-1.090747
b	two	-0.999308	-2.527985

In [753]:

```
df.groupby(['key1', 'key2']).size()
```

Out[753]:

key1 key2

a	one	2
a	two	1
b	one	1
b	two	1

dtype: int64

In [754]:

```
for name, group in df.groupby('key1'):
    print(name)
    print(group)
```

a

	key1	key2	data1	data2
0	a	one	1.030924	0.397888
1	a	two	1.668416	-1.148195
4	a	one	1.309694	-1.363892
b				
2	b	one	-0.149717	-1.090747
3	b	two	-0.999308	-2.527985

In [755]:

```
for (k1, k2), group in df.groupby(['key1', 'key2']):
    print((k1, k2))
    print(group)
```

```
('a', 'one')
   key1 key2      data1      data2
0     a  one  1.030924  0.397888
4     a  one  1.309694 -1.363892
('a', 'two')
   key1 key2      data1      data2
1     a  two  1.668416 -1.148195
('b', 'one')
   key1 key2      data1      data2
2     b  one -0.149717 -1.090747
('b', 'two')
   key1 key2      data1      data2
3     b  two -0.999308 -2.527985
```

In [756]:

```
pieces = dict(list(df.groupby('key1')))
```

In [757]:

```
pieces['b']
```

Out[757]:

	key1	key2	data1	data2
2	b	one	-0.149717	-1.090747
3	b	two	-0.999308	-2.527985

In [758]:

```
df.dtypes
```

Out[758]:

```
key1      object
key2      object
data1     float64
data2     float64
dtype: object
```

In [759]:

```
grouped = df.groupby(df.dtypes, axis=1)
```

In [760]:

```
for dtype, group in grouped:  
    print(dtype)  
    print(group)
```

```
float64  
      data1      data2  
0  1.030924  0.397888  
1  1.668416 -1.148195  
2 -0.149717 -1.090747  
3 -0.999308 -2.527985  
4  1.309694 -1.363892  
object  
  key1  key2  
0     a   one  
1     a   two  
2     b   one  
3     b   two  
4     a   one
```

In [761]:

```
df.groupby('key1')['data1']  
df.groupby('key1')[['data2']]
```

Out[761]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001670F8A4508>
```

In [762]:

```
df[['data1']].groupby(df['key1'])  
df[['data2']].groupby(df['key1'])
```

Out[762]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001670F8A5488>
```

In [763]:

```
df.groupby(['key1', 'key2'])[['data2']].mean()
```

Out[763]:

```
data2
```

key1	key2	
	one	-0.483002
a	two	-1.148195
	one	-1.090747
b	two	-2.527985

In [764]:

```
s_grouped = df.groupby(['key1', 'key2'])['data2']
```

In [765]:

```
s_grouped
```

Out[765]:

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x00000167122C33C8>
```

In [766]:

```
s_grouped.mean()
```

Out[766]:

```
key1  key2
a      one    -0.483002
       two    -1.148195
b      one    -1.090747
       two    -2.527985
Name: data2, dtype: float64
```

In [767]:

```
people = pd.DataFrame(np.random.randn(5, 5),
                      columns=['a', 'b', 'c', 'd', 'e'],
                      index=['Joe', 'Steve', 'Wes', 'Jim', 'Travis'])
```

In [768]:

```
people.iloc[2:3, [1, 2]] = np.nan
```

In [769]:

```
people
```

Out[769]:

	a	b	c	d	e
Joe	-2.452923	1.431814	0.786235	0.772756	-1.053437
Steve	-0.664470	-0.535794	-1.069377	0.335968	1.698140
Wes	0.097137	NaN	NaN	-0.000325	1.742777
Jim	-0.019753	0.223437	-0.318552	1.029619	-1.695942
Travis	-0.938932	-0.383253	-1.811586	0.739416	0.915601

In [770]:

```
mapping = {'a': 'red', 'b': 'red', 'c': 'blue',
           'd': 'blue', 'e': 'red', 'f' : 'orange'}
```

In [771]:

```
by_column = people.groupby(mapping, axis=1)
```

In [772]:

```
by_column.sum()
```

Out[772]:

	blue	red
Joe	1.558991	-2.074545
Steve	-0.733409	0.497877
Wes	-0.000325	1.839915
Jim	0.711068	-1.492258
Travis	-1.072170	-0.406584

In [773]:

```
map_series = pd.Series(mapping)
```

In [774]:

```
map_series
```

Out[774]:

```
a      red
b      red
c    blue
d    blue
e      red
f  orange
dtype: object
```

In [775]:

```
people.groupby(map_series, axis=1).count()
```

Out[775]:

	blue	red
Joe	2	3
Steve	2	3
Wes	1	2
Jim	2	3
Travis	2	3

In [776]:

```
people.groupby(len).sum()
```

Out[776]:

	a	b	c	d	e
3	-2.375538	1.655251	0.467683	1.802050	-1.006601
5	-0.664470	-0.535794	-1.069377	0.335968	1.698140
6	-0.938932	-0.383253	-1.811586	0.739416	0.915601

In [777]:

```
key_list = ['one', 'one', 'one', 'two', 'two']
```

In [778]:

```
people.groupby([len, key_list]).min()
```

Out[778]:

	a	b	c	d	e	
3	one	-2.452923	1.431814	0.786235	-0.000325	-1.053437
3	two	-0.019753	0.223437	-0.318552	1.029619	-1.695942
5	one	-0.664470	-0.535794	-1.069377	0.335968	1.698140
6	two	-0.938932	-0.383253	-1.811586	0.739416	0.915601

In [779]:

```
columns = pd.MultiIndex.from_arrays([[ 'US', 'US', 'US', 'JP', 'JP'],
                                     [1, 3, 5, 1, 3]],
                                     names=['cty', 'tenor'])
```

In [780]:

```
hier_df = pd.DataFrame(np.random.randn(4, 5), columns=columns)
```

In [781]:

```
hier_df
```

Out[781]:

cty	US			JP	
tenor	1	3	5	1	3
0	-0.459369	0.703083	0.075575	0.700022	1.416030
1	-0.700676	-0.301605	0.028612	-1.947259	2.033981
2	-0.138246	-0.719689	2.219406	-1.659319	0.877243
3	0.540589	0.045965	1.437115	0.280476	-0.820854

In [782]:

```
hier_df.groupby(level='cty', axis=1).count()
```

Out[782]:

cty	JP	US
0	2	3
1	2	3
2	2	3
3	2	3

10.2 Data Aggregation

In [783]:

```
df
```

Out[783]:

	key1	key2	data1	data2
0	a	one	1.030924	0.397888
1	a	two	1.668416	-1.148195
2	b	one	-0.149717	-1.090747
3	b	two	-0.999308	-2.527985
4	a	one	1.309694	-1.363892

In [784]:

```
grouped = df.groupby('key1')
```

In [785]:

```
grouped['data1'].quantile(0.9)
```

Out[785]:

```
key1
a    1.596672
b   -0.234676
Name: data1, dtype: float64
```

In [786]:

```
def peak_to_peak(arr):
    return arr.max() - arr.min()
```

In [787]:

```
grouped.agg(peak_to_peak)
```

Out[787]:

	data1	data2
--	-------	-------

key1

a	0.637492	1.761780
b	0.849592	1.437238

In [788]:

```
grouped.describe()
```

Out[788]:

	data1									
	count	mean	std	min	25%	50%	75%	max	count	
key1										
a	3.0	1.336344	0.319581	1.030924	1.170309	1.309694	1.489055	1.668416	3.0	
b	2.0	-0.574513	0.600752	-0.999308	-0.786910	-0.574513	-0.362115	-0.149717	2.0	

In [789]:

```
tips = pd.read_csv('examples/tips.csv')
```

In [790]:

```
tips['tip_pct'] = tips['tip'] / tips['total_bill']
```

In [791]:

```
tips[:6]
```

Out[791]:

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.059447
1	10.34	1.66	No	Sun	Dinner	3	0.160542
2	21.01	3.50	No	Sun	Dinner	3	0.166587
3	23.68	3.31	No	Sun	Dinner	2	0.139780
4	24.59	3.61	No	Sun	Dinner	4	0.146808
5	25.29	4.71	No	Sun	Dinner	4	0.186240

In [792]:

```
grouped = tips.groupby(['day', 'smoker'])
```

In [793]:

```
grouped_pct = grouped['tip_pct']
```

In [794]:

```
grouped_pct.agg('mean')
```

Out[794]:

```
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
```

In [795]:

```
grouped_pct.agg(['mean', 'std', 'peak_to_peak'])
```

Out[795]:

		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

In [796]:

```
grouped_pct.agg([('foo', 'mean'), ('bar', np.std)])
```

Out[796]:

		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

In [797]:

```
functions = ['count', 'mean', 'max']
```

In [798]:

```
result = grouped[['tip_pct', 'total_bill']].agg(functions)
```

...

In [799]:

```
result
```

Out[799]:

day	smoker	tip_pct			total_bill		
		count	mean	max	count	mean	max
Fri	No	4	0.151650	0.187735	4	18.420000	22.75
	Yes	15	0.174783	0.263480	15	16.813333	40.17
Sat	No	45	0.158048	0.291990	45	19.661778	48.33
	Yes	42	0.147906	0.325733	42	21.276667	50.81
Sun	No	57	0.160113	0.252672	57	20.506667	48.17
	Yes	19	0.187250	0.710345	19	24.120000	45.35
Thur	No	45	0.160298	0.266312	45	17.113111	41.19
	Yes	17	0.163863	0.241255	17	19.190588	43.11

In [800]:

```
result['tip_pct']
```

Out[800]:

		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

In [801]:

```
ftuples = [('Durchschnitt', 'mean'), ('Abweichung', np.var)]
```

In [802]:

```
grouped[['tip_pct', 'total_bill']].agg(ftuples)
```

C:\Users\Kader\anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.

"""Entry point for launching an IPython kernel.

Out[802]:

		tip_pct		total_bill	
day	smoker	Durchschnitt	Abweichung	Durchschnitt	Abweichung
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

In [803]:

```
grouped.agg({'tip' : np.max, 'size' : 'sum'})
```

Out[803]:

		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 [804]:

```
grouped.agg({'tip_pct' : ['min', 'max', 'mean', 'std'],
             'size' : 'sum'})
```

Out[804]:

		tip_pct	size			
day	smoker	min	max	mean	std	sum
Fri	No	0.120385	0.187735	0.151650	0.028123	9
	Yes	0.103555	0.263480	0.174783	0.051293	31
Sat	No	0.056797	0.291990	0.158048	0.039767	115
	Yes	0.035638	0.325733	0.147906	0.061375	104
Sun	No	0.059447	0.252672	0.160113	0.042347	167
	Yes	0.065660	0.710345	0.187250	0.154134	49
Thur	No	0.072961	0.266312	0.160298	0.038774	112
	Yes	0.090014	0.241255	0.163863	0.039389	40

In [805]:

```
tips.groupby(['day', 'smoker'], as_index=False).mean()
```

Out[805]:

	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

10.3 Apply: General split-apply-combine

In [806]:

```
def top(df, n=5, column='tip_pct'):
    return df.sort_values(by=column)[-n:]
```

In [807]:

```
top(tips, n=6)
```

Out[807]:

	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

In [808]:

```
tips.groupby('smoker').apply(top)
```

Out[808]:

		total_bill	tip	smoker	day	time	size	tip_pct
smoker								
	88	24.71	5.85	No	Thur	Lunch	2	0.236746
	185	20.69	5.00	No	Sun	Dinner	5	0.241663
No	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
Yes	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

In [809]:

```
tips.groupby(['smoker', 'day']).apply(top, n=1, column='total_bill')
```

Out[809]:

			total_bill	tip	smoker	day	time	size	tip_pct
smoker	day								
	Fri	94	22.75	3.25	No	Fri	Dinner	2	0.142857
	Sat	212	48.33	9.00	No	Sat	Dinner	4	0.186220
No	Sun	156	48.17	5.00	No	Sun	Dinner	6	0.103799
	Thur	142	41.19	5.00	No	Thur	Lunch	5	0.121389
	Fri	95	40.17	4.73	Yes	Fri	Dinner	4	0.117750
Yes	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 [810]:

```
result = tips.groupby('smoker')['tip_pct'].describe()
```

In [811]:

```
result
```

Out[811]:

	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 [812]:

```
result.unstack('smoker')
```

Out[812]:

```
smoker
count  No      151.000000
       Yes     93.000000
mean   No      0.159328
       Yes     0.163196
std    No      0.039910
          ...
50%    Yes     0.153846
75%    No      0.185014
       Yes     0.195059
max    No      0.291990
       Yes     0.710345
Length: 16, dtype: float64
```

In [813]:

```
f = lambda x: x.describe()
grouped.apply(f)
```

Out[813]:

			total_bill	tip	size	tip_pct
day	smoker					
		count	4.000000	4.000000	4.00	4.000000
		mean	18.420000	2.812500	2.25	0.151650
Fri	No	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
...
		min	10.340000	2.000000	2.00	0.090014
		25%	13.510000	2.000000	2.00	0.148038
Thur	Yes	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

In [814]:

```
tips.groupby('smoker', group_keys=False).apply(top)
```

Out[814]:

	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

In [815]:

```
frame = pd.DataFrame({'data1': np.random.randn(1000),
                      'data2': np.random.randn(1000)})
```

In [816]:

```
quartiles = pd.cut(frame.data1, 4)
```

In [817]:

```
quartiles[:10]
```

Out[817]:

```
0    (-2.993, -1.443]
1    (-1.443, 0.101]
2    (-2.993, -1.443]
3    (-1.443, 0.101]
4    (0.101, 1.646]
5    (-1.443, 0.101]
6    (-1.443, 0.101]
7    (-1.443, 0.101]
8    (0.101, 1.646]
9    (-2.993, -1.443]
Name: data1, dtype: category
Categories (4, interval[float64]): [(-2.993, -1.443] < (-1.443, 0.101] < (0.101, 1.646] < (1.646, 3.19]]
```

In [818]:

```
def get_stats(group):
    return {'min': group.min(), 'max': group.max(),
            'count': group.count(), 'mean': group.mean()}
```

In [819]:

```
grouped = frame.data2.groupby(quartiles)
```

In [820]:

```
grouped.apply(get_stats).unstack()
```

Out[820]:

	min	max	count	mean
data1				
(-2.993, -1.443]	-2.522638	2.086927	82.0	0.023174
(-1.443, 0.101]	-3.105636	2.916153	457.0	-0.006179
(0.101, 1.646]	-3.170292	2.766222	423.0	-0.040413
(1.646, 3.19]	-3.044612	1.499182	38.0	-0.270741

In [821]:

```
grouping = pd.qcut(frame.data1, 10, labels=False)
```

In [822]:

```
grouped = frame.data2.groupby(grouping)
```

In [823]:

```
grouped.apply(get_stats).unstack()
```

Out[823]:

	min	max	count	mean
0	-2.522638	2.522842	100.0	0.035884
1	-3.105636	2.916153	100.0	-0.166016
2	-1.989939	2.032291	100.0	-0.006380
3	-2.293376	2.381733	100.0	0.148442
4	-2.252000	2.477555	100.0	0.029979
5	-2.839491	2.231839	100.0	-0.189276
6	-3.170292	2.492410	100.0	-0.013040
7	-2.658241	1.864475	100.0	-0.132674
8	-2.998907	2.766222	100.0	0.072498
9	-3.044612	2.134639	100.0	-0.062478

In [824]:

```
s = pd.Series(np.random.randn(6))
```

In [825]:

```
s[::-2] = np.nan
```

In [826]:

```
s
```

Out[826]:

```
0      NaN
1    0.906168
2      NaN
3    0.011966
4      NaN
5   -0.336270
dtype: float64
```

In [827]:

```
s.fillna(s.mean())
```

Out[827]:

```
0    0.193955
1    0.906168
2    0.193955
3    0.011966
4    0.193955
5   -0.336270
dtype: float64
```

In [828]:

```
states = ['Ohio', 'New York', 'Vermont', 'Florida',
          'Oregon', 'Nevada', 'California', 'Idaho']
```

In [829]:

```
group_key = ['East'] * 4 + ['West'] * 4
```

In [830]:

```
data = pd.Series(np.random.randn(8), index=states)
```

In [831]:

```
data
```

Out[831]:

```
Ohio        1.590860
New York    2.153703
Vermont     -0.449880
Florida     -0.314014
Oregon      -0.327079
Nevada      -0.752324
California   0.316928
Idaho       0.523960
dtype: float64
```

In [832]:

```
data[['Vermont', 'Nevada', 'Idaho']] = np.nan
```

In [833]:

```
data
```

Out[833]:

```
Ohio      1.590860
New York  2.153703
Vermont    NaN
Florida   -0.314014
Oregon    -0.327079
Nevada     NaN
California 0.316928
Idaho      NaN
dtype: float64
```

In [834]:

```
data.groupby(group_key).mean()
```

Out[834]:

```
East     1.143516
West    -0.005076
dtype: float64
```

In [835]:

```
fill_mean = lambda g: g.fillna(g.mean())
```

In [836]:

```
data.groupby(group_key).apply(fill_mean)
```

Out[836]:

```
Ohio      1.590860
New York  2.153703
Vermont   1.143516
Florida   -0.314014
Oregon    -0.327079
Nevada    -0.005076
California 0.316928
Idaho     -0.005076
dtype: float64
```

In [837]:

```
fill_values = {'East': 0.5, 'West': -1}
```

In [838]:

```
fill_func = lambda g: g.fillna(fill_values[g.name])
```

In [839]:

```
data.groupby(group_key).apply(fill_func)
```

Out[839]:

```
Ohio      1.590860
New York  2.153703
Vermont   0.500000
Florida   -0.314014
Oregon    -0.327079
Nevada    -1.000000
California 0.316928
Idaho     -1.000000
dtype: float64
```

In [840]:

```
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)
```

In [841]:

```
deck[:13]
```

Out[841]:

```
AH      1
2H      2
3H      3
4H      4
5H      5
..
9H      9
10H    10
JH      10
KH      10
QH      10
Length: 13, dtype: int64
```

In [842]:

```
def draw(deck, n=5):
    return deck.sample(n)
```

In [843]:

```
draw(deck)
```

Out[843]:

```
2C      2  
7H      7  
6D      6  
9C      9  
JH      10  
dtype: int64
```

In [844]:

```
get_suit = lambda card: card[-1]
```

In [845]:

```
deck.groupby(get_suit).apply(draw, n=2)
```

Out[845]:

```
C  AC      1  
   QC     10  
D  2D      2  
   JD     10  
H  5H      5  
   KH     10  
S  4S      4  
   QS     10  
dtype: int64
```

In [846]:

```
deck.groupby(get_suit, group_keys=False).apply(draw, n=2)
```

Out[846]:

```
8C      8  
9C      9  
AD      1  
5D      5  
6H      6  
3H      3  
AS      1  
KS     10  
dtype: int64
```

In [847]:

```
df = pd.DataFrame({'category': ['a', 'a', 'a', 'a',  
                                'b', 'b', 'b', 'b'],  
                   'data': np.random.randn(8),  
                   'weights': np.random.rand(8)})
```

In [848]:

```
df
```

Out[848]:

	category	data	weights
0	a	-1.414637	0.149135
1	a	-0.573363	0.815760
2	a	1.802446	0.308881
3	a	-0.269092	0.384255
4	b	-0.133715	0.062249
5	b	-0.096225	0.934428
6	b	0.106579	0.776801
7	b	-1.040987	0.396084

In [849]:

```
grouped = df.groupby('category')
```

In [850]:

```
get_wavg = lambda g: np.average(g['data'], weights=g['weights'])
```

In [851]:

```
grouped.apply(get_wavg)
```

Out[851]:

```
category
a    -0.135918
b    -0.197167
dtype: float64
```

In [456]:

```
close_px = pd.read_csv('examples/stock_px_2.csv', parse_dates=True,
index_col=0)
```

In [457]:

```
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 [856]:

```
close_px[-4:]
```

Out[856]:

	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

In [857]:

```
spx_corr = lambda x: x.corrwith(x['SPX'])
```

In [858]:

```
rets = close_px.pct_change().dropna()
```

In [859]:

```
get_year = lambda x: x.year
```

In [860]:

```
by_year = rets.groupby(get_year)
```

In [861]:

```
by_year.apply(spx_corr)
```

Out[861]:

	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

In [862]:

```
by_year.apply(lambda g: g['AAPL'].corr(g['MSFT']))
```

Out[862]:

```
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
```

In [863]:

```
import statsmodels.api as sm
def regress(data, yvar, xvars):
    Y = data[yvar]
    X = data[xvars]
    X['intercept'] = 1.
    result = sm.OLS(Y, X).fit()
    return result.params
```

In [864]:

```
by_year.apply(regress, 'AAPL', ['SPX'])
```

Out[864]:

	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 Tables and Cross-Tabulation

In [865]:

```
tips.pivot_table(index=['day', 'smoker'])
```

Out[865]:

		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

In [866]:

```
tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
                 columns='smoker')
```

Out[866]:

		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	
Lunch	Thur	2.000000	NaN	0.159744	NaN	
	Fri	3.000000	1.833333	0.187735	0.188937	
	Thur	2.500000	2.352941	0.160311	0.163863	

In [867]:

```
tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
                 columns='smoker', margins=True)
```

Out[867]:

		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	
Lunch	Thur	2.000000	NaN	2.000000	0.159744	NaN	0.159744	
	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	

In [868]:

```
tips.pivot_table('tip_pct', index=['time', 'smoker'], columns='day',
                 aggfunc=len, margins=True)
```

Out[868]:

		day	Fri	Sat	Sun	Thur	All	
	time	smoker	No	3.0	45.0	57.0	1.0	106.0
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	

In [869]:

```
tips.pivot_table('tip_pct', index=['time', 'size', 'smoker'],
                 columns='day', aggfunc='mean', fill_value=0)
```

Out[869]:

		day	Fri	Sat	Sun	Thur		
	time	size	smoker	No	0.000000	0.137931	0.000000	0.000000
Dinner	1	No	0.000000	0.137931	0.000000	0.000000		
		Yes	0.000000	0.325733	0.000000	0.000000		
Dinner	2	No	0.139622	0.162705	0.168859	0.159744		
		Yes	0.171297	0.148668	0.207893	0.000000		
Dinner	3	No	0.000000	0.154661	0.152663	0.000000		
		Yes	0.000000	0.000000	0.000000	0.204952		
Lunch	4	No	0.000000	0.000000	0.000000	0.138919		
		Yes	0.000000	0.000000	0.000000	0.155410		
Lunch	5	No	0.000000	0.000000	0.000000	0.121389		
		Yes	0.000000	0.000000	0.000000	0.173706		

21 rows × 4 columns

In [870]:

```
data
```

Out[870]:

```
Ohio      1.590860
New York  2.153703
Vermont    NaN
Florida   -0.314014
Oregon    -0.327079
Nevada     NaN
California 0.316928
Idaho      NaN
dtype: float64
```

In [872]:

```
pd.crosstab([tips.time, tips.day], tips.smoker, margins=True)
```

Out[872]:

		smoker	No	Yes	All
	time	day			
		Fri	3	9	12
Dinner		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

11. TIME SERIES

11.1 Date and Time Data Types and Tools

In [873]:

```
from datetime import datetime
```

In [874]:

```
now = datetime.now()
```

In [875]:

```
now
```

Out[875]:

```
datetime.datetime(2020, 11, 26, 17, 59, 54, 441556)
```

In [876]:

```
now.year, now.month, now.day
```

Out[876]:

```
(2020, 11, 26)
```

In [877]:

```
delta = datetime(2011, 1, 7) - datetime(2008, 6, 24, 8, 15)
```

In [878]:

```
delta
```

Out[878]:

```
datetime.timedelta(days=926, seconds=56700)
```

In [879]:

```
delta.days
```

Out[879]:

```
926
```

In [880]:

```
delta.seconds
```

Out[880]:

```
56700
```

In [881]:

```
from datetime import timedelta
```

In [882]:

```
start = datetime(2011, 1, 7)
```

In [883]:

```
start + timedelta(12)
```

Out[883]:

```
datetime.datetime(2011, 1, 19, 0, 0)
```

In [884]:

```
start - 2 * timedelta(12)
```

Out[884]:

```
datetime.datetime(2010, 12, 14, 0, 0)
```

In [885]:

```
stamp = datetime(2011, 1, 3)
```

In [886]:

```
str(stamp)
```

Out[886]:

```
'2011-01-03 00:00:00'
```

In [887]:

```
stamp.strftime('%Y-%m-%d')
```

Out[887]:

```
'2011-01-03'
```

In [888]:

```
value = '2011-01-03'
```

In [889]:

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

Out[889]:

```
datetime.datetime(2011, 1, 3, 0, 0)
```

In [890]:

```
datestrs = ['7/6/2011', '8/6/2011']
```

In [891]:

```
[datetime.strptime(x, '%m/%d/%Y') for x in datestrs]
```

Out[891]:

```
[datetime.datetime(2011, 7, 6, 0, 0), datetime.datetime(2011, 8, 6, 0, 0)]
```

In [892]:

```
from dateutil.parser import parse
```

In [893]:

```
parse('2011-01-03')
```

Out[893]:

```
datetime.datetime(2011, 1, 3, 0, 0)
```

In [894]:

```
parse('Jan 31, 1997 10:45 PM')
```

Out[894]:

```
datetime.datetime(1997, 1, 31, 22, 45)
```

In [895]:

```
parse('6/12/2011', dayfirst=True)
```

Out[895]:

```
datetime.datetime(2011, 12, 6, 0, 0)
```

In [896]:

```
datestrs = ['2011-07-06 12:00:00', '2011-08-06 00:00:00']
```

In [897]:

```
pd.to_datetime(datestrs)
```

Out[897]:

```
DatetimeIndex(['2011-07-06 12:00:00', '2011-08-06 00:00:00'], dtype='datetime64[ns]', freq=None)
```

In [898]:

```
idx = pd.to_datetime(datestrs + [None])
```

In [899]:

```
idx
```

Out[899]:

```
DatetimeIndex(['2011-07-06 12:00:00', '2011-08-06 00:00:00', 'NaT'], dtype='datetime64[ns]', freq=None)
```

In [900]:

```
idx[2]
```

Out[900]:

```
NaT
```

In [901]:

```
pd.isnull(idx)
```

Out[901]:

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

11.2 Time Series Basics

In [902]:

```
from datetime import datetime
```

In [903]:

```
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 [904]:

```
ts = pd.Series(np.random.randn(6), index=dates)
```

In [905]:

```
ts
```

Out[905]:

```
2011-01-02    -1.279869
2011-01-05    -0.987003
2011-01-07     0.359551
2011-01-08     1.395030
2011-01-10     1.713062
2011-01-12     0.225472
dtype: float64
```

In [906]:

```
ts.index
```

Out[906]:

```
DatetimeIndex(['2011-01-02', '2011-01-05', '2011-01-07', '2011-01-08',
                 '2011-01-10', '2011-01-12'],
                dtype='datetime64[ns]', freq=None)
```

In [907]:

```
ts + ts[::-2]
```

Out[907]:

```
2011-01-02    -2.559738
2011-01-05      NaN
2011-01-07     0.719101
2011-01-08      NaN
2011-01-10     3.426124
2011-01-12      NaN
dtype: float64
```

In [908]:

```
ts.index.dtype
```

Out[908]:

```
dtype('M8[ns]')
```

In [909]:

```
stamp = ts.index[0]
```

In [910]:

```
stamp
```

Out[910]:

```
Timestamp('2011-01-02 00:00:00')
```

In [911]:

```
stamp = ts.index[2]
```

In [912]:

```
ts[stamp]
```

Out[912]:

```
0.3595507176606856
```

In [914]:

```
ts['1/10/2011']
```

Out[914]:

```
1.7130621701255242
```

In [915]:

```
ts['20110110']
```

Out[915]:

```
1.7130621701255242
```

In [916]:

```
longer_ts = pd.Series(np.random.randn(1000),
                      index=pd.date_range('1/1/2000', periods=1000))
```

In [917]:

```
longer_ts
```

Out[917]:

```
2000-01-01    1.418858
2000-01-02    1.096439
2000-01-03    1.169415
2000-01-04   -0.609634
2000-01-05   -0.721526
...
2002-09-22   -0.275711
2002-09-23    1.741427
2002-09-24    0.389016
2002-09-25    1.048313
2002-09-26    1.184063
Freq: D, Length: 1000, dtype: float64
```

In [918]:

```
longer_ts['2001']
```

Out[918]:

```
2001-01-01   -0.249067
2001-01-02    0.991027
2001-01-03   -0.122792
2001-01-04    0.016021
2001-01-05   -0.028394
...
2001-12-27    1.423234
2001-12-28   -1.301768
2001-12-29   -0.317198
2001-12-30    0.767908
2001-12-31   -0.802281
Freq: D, Length: 365, dtype: float64
```

In [919]:

```
longer_ts['2001-05']
```

Out[919]:

```
2001-05-01    0.307430
2001-05-02   -1.228146
2001-05-03    1.728526
2001-05-04   -0.721340
2001-05-05    0.998089
...
2001-05-27    0.604569
2001-05-28   -0.621221
2001-05-29    2.054181
2001-05-30   -1.211328
2001-05-31   -1.037526
Freq: D, Length: 31, dtype: float64
```

In [920]:

```
ts[datetime(2011, 1, 7):]
```

Out[920]:

```
2011-01-07    0.359551
2011-01-08    1.395030
2011-01-10    1.713062
2011-01-12    0.225472
dtype: float64
```

In [921]:

```
ts
```

Out[921]:

```
2011-01-02   -1.279869
2011-01-05   -0.987003
2011-01-07    0.359551
2011-01-08    1.395030
2011-01-10    1.713062
2011-01-12    0.225472
dtype: float64
```

In [922]:

```
ts['1/6/2011':'1/11/2011']
```

Out[922]:

```
2011-01-07    0.359551
2011-01-08    1.395030
2011-01-10    1.713062
dtype: float64
```

In [923]:

```
ts.truncate(after='1/9/2011')
```

Out[923]:

```
2011-01-02   -1.279869
2011-01-05   -0.987003
2011-01-07    0.359551
2011-01-08    1.395030
dtype: float64
```

In [924]:

```
dates = pd.date_range('1/1/2000', periods=100, freq='W-WED')
```

In [925]:

```
long_df = pd.DataFrame(np.random.randn(100, 4),
                       index=dates,
                       columns=['Colorado', 'Texas', 'New York', 'Ohio'])
```

In [926]:

```
long_df.loc['5-2001']
```

Out[926]:

	Colorado	Texas	New York	Ohio
2001-05-02	1.242534	0.661915	-0.112194	1.019570
2001-05-09	-0.625272	-0.225277	1.145207	-1.570418
2001-05-16	-1.649994	-1.073970	1.505576	0.362319
2001-05-23	0.421962	0.085521	0.994241	0.907999
2001-05-30	-0.632205	1.089561	-0.577815	-1.976222

In [927]:

```
dates = pd.DatetimeIndex(['1/1/2000', '1/2/2000', '1/2/2000', '1/2/2000', '1/3/2000'])
```

In [928]:

```
dup_ts = pd.Series(np.arange(5), index=dates)
```

In [929]:

```
dup_ts
```

Out[929]:

```
2000-01-01    0
2000-01-02    1
2000-01-02    2
2000-01-02    3
2000-01-03    4
dtype: int32
```

In [930]:

```
dup_ts.index.is_unique
```

Out[930]:

```
False
```

In [931]:

```
dup_ts.index.is_unique
```

Out[931]:

```
False
```

In [932]:

```
dup_ts['1/2/2000']
```

Out[932]:

```
2000-01-02    1
2000-01-02    2
2000-01-02    3
dtype: int32
```

In [933]:

```
grouped = dup_ts.groupby(level=0)
```

In [934]:

```
grouped.mean()
```

Out[934]:

```
2000-01-01    0
2000-01-02    2
2000-01-03    4
dtype: int32
```

In [935]:

```
grouped.count()
```

Out[935]:

```
2000-01-01    1
2000-01-02    3
2000-01-03    1
dtype: int64
```

11.3 Date Ranges, Frequencies, and Shifting

In [936]:

```
ts
```

Out[936]:

```
2011-01-02    -1.279869
2011-01-05    -0.987003
2011-01-07    0.359551
2011-01-08    1.395030
2011-01-10    1.713062
2011-01-12    0.225472
dtype: float64
```

In [937]:

```
resampler = ts.resample('D')
```

In [938]:

```
index = pd.date_range('2012-04-01', '2012-06-01')
```

In [939]:

```
index
```

Out[939]:

```
DatetimeIndex(['2012-04-01', '2012-04-02', '2012-04-03', '2012-04-04',
 '2012-04-05', '2012-04-06', '2012-04-07', '2012-04-08',
 '2012-04-09', '2012-04-10', '2012-04-11', '2012-04-12',
 '2012-04-13', '2012-04-14', '2012-04-15', '2012-04-16',
 '2012-04-17', '2012-04-18', '2012-04-19', '2012-04-20',
 '2012-04-21', '2012-04-22', '2012-04-23', '2012-04-24',
 '2012-04-25', '2012-04-26', '2012-04-27', '2012-04-28',
 '2012-04-29', '2012-04-30', '2012-05-01', '2012-05-02',
 '2012-05-03', '2012-05-04', '2012-05-05', '2012-05-06',
 '2012-05-07', '2012-05-08', '2012-05-09', '2012-05-10',
 '2012-05-11', '2012-05-12', '2012-05-13', '2012-05-14',
 '2012-05-15', '2012-05-16', '2012-05-17', '2012-05-18',
 '2012-05-19', '2012-05-20', '2012-05-21', '2012-05-22',
 '2012-05-23', '2012-05-24', '2012-05-25', '2012-05-26',
 '2012-05-27', '2012-05-28', '2012-05-29', '2012-05-30',
 '2012-05-31', '2012-06-01'],
 dtype='datetime64[ns]', freq='D')
```

In [940]:

```
pd.date_range(start='2012-04-01', periods=20)
```

Out[940]:

```
DatetimeIndex(['2012-04-01', '2012-04-02', '2012-04-03', '2012-04-04',
 '2012-04-05', '2012-04-06', '2012-04-07', '2012-04-08',
 '2012-04-09', '2012-04-10', '2012-04-11', '2012-04-12',
 '2012-04-13', '2012-04-14', '2012-04-15', '2012-04-16',
 '2012-04-17', '2012-04-18', '2012-04-19', '2012-04-20'],
 dtype='datetime64[ns]', freq='D')
```

In [941]:

```
pd.date_range(end='2012-06-01', periods=20)
```

Out[941]:

```
DatetimeIndex(['2012-05-13', '2012-05-14', '2012-05-15', '2012-05-16',
 '2012-05-17', '2012-05-18', '2012-05-19', '2012-05-20',
 '2012-05-21', '2012-05-22', '2012-05-23', '2012-05-24',
 '2012-05-25', '2012-05-26', '2012-05-27', '2012-05-28',
 '2012-05-29', '2012-05-30', '2012-05-31', '2012-06-01'],
 dtype='datetime64[ns]', freq='D')
```

In [942]:

```
pd.date_range('2000-01-01', '2000-12-01', freq='BM')
```

Out[942]:

```
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 [943]:

```
pd.date_range('2012-05-02 12:56:31', periods=5)
```

Out[943]:

```
DatetimeIndex(['2012-05-02 12:56:31', '2012-05-03 12:56:31',
                 '2012-05-04 12:56:31', '2012-05-05 12:56:31',
                 '2012-05-06 12:56:31'],
                dtype='datetime64[ns]', freq='D')
```

In [944]:

```
pd.date_range('2012-05-02 12:56:31', periods=5, normalize=True)
```

Out[944]:

```
DatetimeIndex(['2012-05-02', '2012-05-03', '2012-05-04', '2012-05-05',
                 '2012-05-06'],
                dtype='datetime64[ns]', freq='D')
```

In [945]:

```
from pandas.tseries.offsets import Hour, Minute
```

In [946]:

```
hour = Hour()
```

In [947]:

```
hour
```

Out[947]:

```
<Hour>
```

In [948]:

```
four_hours = Hour(4)
```

In [949]:

```
four_hours
```

Out[949]:

```
<4 * Hours>
```

In [950]:

```
pd.date_range('2000-01-01', '2000-01-03 23:59', freq='4h')
```

Out[950]:

```
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')
```

In [951]:

```
Hour(2) + Minute(30)
```

Out[951]:

```
<150 * Minutes>
```

In [952]:

```
pd.date_range('2000-01-01', periods=10, freq='1h30min')
```

Out[952]:

```
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')
```

In [953]:

```
rng = pd.date_range('2012-01-01', '2012-09-01', freq='WOM-3FRI')
```

In [954]:

```
list(rng)
```

Out[954]:

```
[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')]
```

In [955]:

```
ts = pd.Series(np.random.randn(4),
               index=pd.date_range('1/1/2000', periods=4, freq='M'))
```

In [956]:

```
ts
```

Out[956]:

```
2000-01-31    1.112505
2000-02-29   -0.573516
2000-03-31   -0.824013
2000-04-30   -0.122321
Freq: M, dtype: float64
```

In [957]:

```
ts.shift(2)
```

Out[957]:

```
2000-01-31      NaN
2000-02-29      NaN
2000-03-31    1.112505
2000-04-30   -0.573516
Freq: M, dtype: float64
```

In [958]:

```
ts.shift(-2)
```

Out[958]:

```
2000-01-31   -0.824013
2000-02-29   -0.122321
2000-03-31      NaN
2000-04-30      NaN
Freq: M, dtype: float64
```

In [959]:

```
ts.shift(2, freq='M')
```

Out[959]:

```
2000-03-31    1.112505
2000-04-30   -0.573516
2000-05-31   -0.824013
2000-06-30   -0.122321
Freq: M, dtype: float64
```

In [960]:

```
ts.shift(3, freq='D')
```

Out[960]:

```
2000-02-03    1.112505
2000-03-03   -0.573516
2000-04-03   -0.824013
2000-05-03   -0.122321
dtype: float64
```

In [961]:

```
ts.shift(1, freq='90T')
```

Out[961]:

```
2000-01-31 01:30:00    1.112505
2000-02-29 01:30:00   -0.573516
2000-03-31 01:30:00   -0.824013
2000-04-30 01:30:00   -0.122321
Freq: M, dtype: float64
```

In [962]:

```
from pandas.tseries.offsets import Day, MonthEnd
```

In [963]:

```
now = datetime(2011, 11, 17)
```

In [964]:

```
now + 3 * Day()
```

Out[964]:

```
Timestamp('2011-11-20 00:00:00')
```

In [965]:

```
now + MonthEnd()
```

Out[965]:

```
Timestamp('2011-11-30 00:00:00')
```

In [966]:

```
now + MonthEnd(2)
```

Out[966]:

```
Timestamp('2011-12-31 00:00:00')
```

In [967]:

```
offset = MonthEnd()
```

In [968]:

```
offset.rollforward(now)
```

Out[968]:

```
Timestamp('2011-11-30 00:00:00')
```

In [969]:

```
offset.rollback(now)
```

Out[969]:

```
Timestamp('2011-10-31 00:00:00')
```

In [970]:

```
ts = pd.Series(np.random.randn(20),  
               index=pd.date_range('1/15/2000', periods=20, freq='4d'))
```

In [971]:

```
ts
```

Out[971]:

```
2000-01-15    -0.520201  
2000-01-19     0.159705  
2000-01-23    -0.964586  
2000-01-27    -1.681549  
2000-01-31    -0.875571  
...  
2000-03-15    -0.176414  
2000-03-19    -0.608558  
2000-03-23     0.755249  
2000-03-27     0.090615  
2000-03-31    -0.741421  
Freq: 4D, Length: 20, dtype: float64
```

In [972]:

```
ts.groupby(offset.rollforward).mean()
```

Out[972]:

```
2000-01-31    -0.776440  
2000-02-29    -0.095467  
2000-03-31    -0.268375  
dtype: float64
```

In [973]:

```
ts.resample('M').mean()
```

Out[973]:

```
2000-01-31    -0.776440  
2000-02-29    -0.095467  
2000-03-31    -0.268375  
Freq: M, dtype: float64
```

11.4 Time Zone Handling

In [974]:

```
import pytz
```

In [975]:

```
pytz.common_timezones[-5:]
```

Out[975]:

```
['US/Eastern', 'US/Hawaii', 'US/Mountain', 'US/Pacific', 'UTC']
```

In [976]:

```
tz = pytz.timezone('America/New_York')
```

In [977]:

```
tz
```

Out[977]:

```
<DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>
```

In [978]:

```
rng = pd.date_range('3/9/2012 9:30', periods=6, freq='D')
```

In [979]:

```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
```

In [980]:

```
ts
```

Out[980]:

```
2012-03-09 09:30:00    0.000716
2012-03-10 09:30:00   -0.408387
2012-03-11 09:30:00    1.036041
2012-03-12 09:30:00    0.431407
2012-03-13 09:30:00    0.922326
2012-03-14 09:30:00   -0.285790
Freq: D, dtype: float64
```

In [981]:

```
print(ts.index.tz)
```

None

In [982]:

```
pd.date_range('3/9/2012 9:30', periods=10, freq='D', tz='UTC')
```

Out[982]:

```
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 [983]:

```
ts
```

Out[983]:

```
2012-03-09 09:30:00    0.000716
2012-03-10 09:30:00   -0.408387
2012-03-11 09:30:00    1.036041
2012-03-12 09:30:00    0.431407
2012-03-13 09:30:00    0.922326
2012-03-14 09:30:00   -0.285790
Freq: D, dtype: float64
```

In [984]:

```
ts_utc = ts.tz_localize('UTC')
```

In [985]:

```
ts_utc
```

Out[985]:

```
2012-03-09 09:30:00+00:00    0.000716
2012-03-10 09:30:00+00:00   -0.408387
2012-03-11 09:30:00+00:00    1.036041
2012-03-12 09:30:00+00:00    0.431407
2012-03-13 09:30:00+00:00    0.922326
2012-03-14 09:30:00+00:00   -0.285790
Freq: D, dtype: float64
```

In [986]:

```
ts_utc.index
```

Out[986]:

```
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')
```

In [987]:

```
ts_utc.tz_convert('America/New_York')
```

Out[987]:

```
2012-03-09 04:30:00-05:00    0.000716
2012-03-10 04:30:00-05:00   -0.408387
2012-03-11 05:30:00-04:00    1.036041
2012-03-12 05:30:00-04:00    0.431407
2012-03-13 05:30:00-04:00    0.922326
2012-03-14 05:30:00-04:00   -0.285790
Freq: D, dtype: float64
```

In [988]:

```
ts_eastern = ts.tz_localize('America/New_York')
```

In [989]:

```
ts_eastern.tz_convert('UTC')
```

Out[989]:

```
2012-03-09 14:30:00+00:00    0.000716
2012-03-10 14:30:00+00:00   -0.408387
2012-03-11 13:30:00+00:00    1.036041
2012-03-12 13:30:00+00:00    0.431407
2012-03-13 13:30:00+00:00    0.922326
2012-03-14 13:30:00+00:00   -0.285790
Freq: D, dtype: float64
```

In [990]:

```
ts_eastern.tz_convert('Europe/Berlin')
```

Out[990]:

```
2012-03-09 15:30:00+01:00    0.000716
2012-03-10 15:30:00+01:00   -0.408387
2012-03-11 14:30:00+01:00    1.036041
2012-03-12 14:30:00+01:00    0.431407
2012-03-13 14:30:00+01:00    0.922326
2012-03-14 14:30:00+01:00   -0.285790
Freq: D, dtype: float64
```

In [991]:

```
ts.index.tz_localize('Asia/Shanghai')
```

Out[991]:

```
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')
```

In [992]:

```
stamp = pd.Timestamp('2011-03-12 04:00')
```

In [993]:

```
stamp_utc = stamp.tz_localize('utc')
```

In [994]:

```
stamp_utc.tz_convert('America/New_York')
```

Out[994]:

```
Timestamp('2011-03-11 23:00:00-0500', tz='America/New_York')
```

In [995]:

```
stamp_moscow = pd.Timestamp('2011-03-12 04:00', tz='Europe/Moscow')
```

In [996]:

```
stamp_moscow
```

Out[996]:

```
Timestamp('2011-03-12 04:00:00+0300', tz='Europe/Moscow')
```

In [997]:

```
stamp_utc.value
```

Out[997]:

```
12999024000000000000
```

In [998]:

```
stamp_utc.tz_convert('America/New_York').value
```

Out[998]:

```
12999024000000000000
```

In [999]:

```
from pandas.tseries.offsets import Hour
```

In [1000]:

```
stamp = pd.Timestamp('2012-03-12 01:30', tz='US/Eastern')
```

In [1001]:

```
stamp
```

Out[1001]:

```
Timestamp('2012-03-12 01:30:00-0400', tz='US/Eastern')
```

In [1002]:

```
stamp + Hour()
```

Out[1002]:

```
Timestamp('2012-03-12 02:30:00-0400', tz='US/Eastern')
```

In [1003]:

```
stamp = pd.Timestamp('2012-11-04 00:30', tz='US/Eastern')
```

In [1004]:

```
stamp
```

Out[1004]:

```
Timestamp('2012-11-04 00:30:00-0400', tz='US/Eastern')
```

In [1005]:

```
stamp + 2 * Hour()
```

Out[1005]:

```
Timestamp('2012-11-04 01:30:00-0500', tz='US/Eastern')
```

In [1006]:

```
rng = pd.date_range('3/7/2012 9:30', periods=10, freq='B')
```

In [1007]:

```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
```

In [1008]:

```
ts
```

Out[1008]:

```
2012-03-07 09:30:00    -0.919795
2012-03-08 09:30:00     0.142986
2012-03-09 09:30:00    -0.905176
2012-03-12 09:30:00    -1.008509
2012-03-13 09:30:00    -1.128155
2012-03-14 09:30:00    -0.310488
2012-03-15 09:30:00     2.311796
2012-03-16 09:30:00     0.961643
2012-03-19 09:30:00     1.077483
2012-03-20 09:30:00     1.251228
Freq: B, dtype: float64
```

In [1009]:

```
ts1 = ts[:7].tz_localize('Europe/London')
```

In [1010]:

```
ts2 = ts1[2:].tz_convert('Europe/Moscow')
```

In [1011]:

```
result = ts1 + ts2
```

In [1012]:

```
result.index
```

Out[1012]:

```
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 Periods and Period Arithmetic

In [1013]:

```
p = pd.Period(2007, freq='A-DEC')
```

In [1014]:

```
p
```

Out[1014]:

```
Period('2007', 'A-DEC')
```

In [1017]:

```
p+5
```

Out[1017]:

```
Period('2012', 'A-DEC')
```

In [1016]:

```
p-2
```

Out[1016]:

```
Period('2005', 'A-DEC')
```

In [1018]:

```
pd.Period('2014', freq='A-DEC')-p
```

Out[1018]:

```
<7 * YearEnds: month=12>
```

In [1019]:

```
rng = pd.period_range('2000-01-01', '2000-06-30', freq='M')
```

In [1020]:

```
rng
```

Out[1020]:

```
PeriodIndex(['2000-01', '2000-02', '2000-03', '2000-04', '2000-05', '2000-06'], dtype='period[M]', freq='M')
```

In [1021]:

```
pd.Series(np.random.randn(6), index=rng)
```

Out[1021]:

```
2000-01    -1.066997
2000-02     0.452461
2000-03    -0.065868
2000-04     1.443342
2000-05    -0.422706
2000-06    -0.113887
Freq: M, dtype: float64
```

In [1022]:

```
values = ['2001Q3', '2002Q2', '2003Q1']
```

In [1023]:

```
index = pd.PeriodIndex(values, freq='Q-DEC')
```

In [1024]:

```
index
```

Out[1024]:

```
PeriodIndex(['2001Q3', '2002Q2', '2003Q1'], dtype='period[Q-DEC]', freq='Q-DEC')
```

In [1025]:

```
p = pd.Period('2007', freq='A-DEC')
```

In [1026]:

```
p
```

Out[1026]:

```
Period('2007', 'A-DEC')
```

In [1027]:

```
p.asfreq('M', how='start')
```

Out[1027]:

```
Period('2007-01', 'M')
```

In [1028]:

```
p.asfreq('M', how='end')
```

Out[1028]:

```
Period('2007-12', 'M')
```

In [1029]:

```
p = pd.Period('2007', freq='A-JUN')
```

In [1030]:

```
p
```

Out[1030]:

```
Period('2007', 'A-JUN')
```

In [1031]:

```
p.asfreq('M', 'start')
```

Out[1031]:

```
Period('2006-07', 'M')
```

In [1032]:

```
p.asfreq('M', 'end')
```

Out[1032]:

```
Period('2007-06', 'M')
```

In [1033]:

```
p = pd.Period('Aug-2007', 'M')
```

In [1034]:

```
p.asfreq('A-JUN')
```

Out[1034]:

```
Period('2008', 'A-JUN')
```

In [1035]:

```
rng = pd.period_range('2006', '2009', freq='A-DEC')
```

In [1036]:

```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
```

In [1037]:

```
ts
```

Out[1037]:

```
2006    -1.908969
2007    -0.022584
2008     1.085709
2009    -1.269468
Freq: A-DEC, dtype: float64
```

In [1038]:

```
ts.asfreq('M', how='start')
```

Out[1038]:

```
2006-01    -1.908969
2007-01    -0.022584
2008-01     1.085709
2009-01    -1.269468
Freq: M, dtype: float64
```

In [1039]:

```
ts.asfreq('B', how='end')
```

Out[1039]:

```
2006-12-29   -1.908969
2007-12-31   -0.022584
2008-12-31    1.085709
2009-12-31   -1.269468
Freq: B, dtype: float64
```

In [1040]:

```
p = pd.Period('2012Q4', freq='Q-JAN')
```

In [1041]:

```
p
```

Out[1041]:

```
Period('2012Q4', 'Q-JAN')
```

In [1042]:

```
p.asfreq('D', 'start')
```

Out[1042]:

```
Period('2011-11-01', 'D')
```

In [1043]:

```
p.asfreq('D', 'end')
```

Out[1043]:

```
Period('2012-01-31', 'D')
```

In [1044]:

```
p4pm = (p.asfreq('B', 'e') - 1).asfreq('T', 's') + 16 * 60
```

In [1045]:

```
p4pm
```

Out[1045]:

```
Period('2012-01-30 16:00', 'T')
```

In [1046]:

```
p4pm.to_timestamp()
```

Out[1046]:

```
Timestamp('2012-01-30 16:00:00')
```

In [1047]:

```
rng = pd.period_range('2011Q3', '2012Q4', freq='Q-JAN')
```

In [1048]:

```
ts = pd.Series(np.arange(len(rng)), index=rng)
```

In [1049]:

```
ts
```

Out[1049]:

```
2011Q3    0
2011Q4    1
2012Q1    2
2012Q2    3
2012Q3    4
2012Q4    5
Freq: Q-JAN, dtype: int32
```

In [1050]:

```
new_rng = (rng.asfreq('B', 'e') - 1).asfreq('T', 's') + 16 * 60
```

In [1051]:

```
ts.index = new_rng.to_timestamp()
```

In [1052]:

```
ts
```

Out[1052]:

```
2010-10-28 16:00:00    0
2011-01-28 16:00:00    1
2011-04-28 16:00:00    2
2011-07-28 16:00:00    3
2011-10-28 16:00:00    4
2012-01-30 16:00:00    5
dtype: int32
```

In [1053]:

```
rng = pd.date_range('2000-01-01', periods=3, freq='M')
```

In [1054]:

```
ts = pd.Series(np.random.randn(3), index=rng)
```

In [1055]:

```
ts
```

Out[1055]:

```
2000-01-31    1.923861  
2000-02-29    1.420495  
2000-03-31   -0.338339  
Freq: M, dtype: float64
```

In [1056]:

```
pts = ts.to_period()
```

In [1057]:

```
pts
```

Out[1057]:

```
2000-01    1.923861  
2000-02    1.420495  
2000-03   -0.338339  
Freq: M, dtype: float64
```

In [1058]:

```
rng = pd.date_range('1/29/2000', periods=6, freq='D')
```

In [1059]:

```
ts2 = pd.Series(np.random.randn(6), index=rng)
```

In [1060]:

```
ts2
```

Out[1060]:

```
2000-01-29    0.783651  
2000-01-30   -0.225799  
2000-01-31   -2.533983  
2000-02-01   -0.216427  
2000-02-02   -1.077682  
2000-02-03   -1.204535  
Freq: D, dtype: float64
```

In [1061]:

```
ts2.to_period('M')
```

Out[1061]:

```
2000-01    0.783651
2000-01   -0.225799
2000-01   -2.533983
2000-02   -0.216427
2000-02   -1.077682
2000-02   -1.204535
Freq: M, dtype: float64
```

In [1062]:

```
pts = ts2.to_period()
```

In [1063]:

```
pts
```

Out[1063]:

```
2000-01-29    0.783651
2000-01-30   -0.225799
2000-01-31   -2.533983
2000-02-01   -0.216427
2000-02-02   -1.077682
2000-02-03   -1.204535
Freq: D, dtype: float64
```

In [1064]:

```
pts.to_timestamp(how='end')
```

Out[1064]:

```
2000-01-29 23:59:59.999999999    0.783651
2000-01-30 23:59:59.999999999   -0.225799
2000-01-31 23:59:59.999999999   -2.533983
2000-02-01 23:59:59.999999999   -0.216427
2000-02-02 23:59:59.999999999   -1.077682
2000-02-03 23:59:59.999999999   -1.204535
Freq: D, dtype: float64
```

In [1065]:

```
data = pd.read_csv('examples/macrodata.csv')
```

In [1066]:

```
data.head(5)
```

Out[1066]:

	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbilrate	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 [1067]:

```
data.year
```

Out[1067]:

```
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 [1068]:

```
data.quarter
```

Out[1068]:

```
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
```

In [1069]:

```
index = pd.PeriodIndex(year=data.year, quarter=data.quarter, freq='Q-DEC')
```

In [1070]:

```
index
```

Out[1070]:

```
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 [1071]:

```
data.index = index
```

In [1072]:

```
data.infl
```

Out[1072]:

```
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 Resampling and Frequency Conversion

In [1073]:

```
rng = pd.date_range('2000-01-01', periods=100, freq='D')
```

In [1074]:

```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
```

In [1075]:

```
ts
```

Out[1075]:

```
2000-01-01    -0.329527
2000-01-02     0.576066
2000-01-03    -0.295345
2000-01-04     1.308053
2000-01-05    -0.417799
...
2000-04-05     0.394722
2000-04-06    -0.230306
2000-04-07     0.965007
2000-04-08     0.587713
2000-04-09     0.094304
Freq: D, Length: 100, dtype: float64
```

In [1076]:

```
ts.resample('M').mean()
```

Out[1076]:

```
2000-01-31    0.012566
2000-02-29   -0.133542
2000-03-31   -0.109629
2000-04-30    0.435354
Freq: M, dtype: float64
```

In [1077]:

```
ts.resample('M', kind='period').mean()
```

Out[1077]:

```
2000-01    0.012566
2000-02   -0.133542
2000-03   -0.109629
2000-04    0.435354
Freq: M, dtype: float64
```

In [1078]:

```
rng = pd.date_range('2000-01-01', periods=12, freq='T')
```

In [1079]:

```
ts = pd.Series(np.arange(12), index=rng)
```

In [1080]:

```
ts
```

Out[1080]:

```
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 00:07:00      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, Length: 12, dtype: int32
```

In [1081]:

```
ts.resample('5min', closed='right').sum()
```

Out[1081]:

```
1999-12-31 23:55:00      0
2000-01-01 00:00:00     15
2000-01-01 00:05:00     40
2000-01-01 00:10:00     11
Freq: 5T, dtype: int32
```

In [1082]:

```
ts.resample('5min', closed='right').sum()
```

Out[1082]:

```
1999-12-31 23:55:00      0
2000-01-01 00:00:00     15
2000-01-01 00:05:00     40
2000-01-01 00:10:00     11
Freq: 5T, dtype: int32
```

In [1083]:

```
ts.resample('5min', closed='right', label='right').sum()
```

Out[1083]:

```
2000-01-01 00:00:00      0
2000-01-01 00:05:00     15
2000-01-01 00:10:00     40
2000-01-01 00:15:00     11
Freq: 5T, dtype: int32
```

In [1084]:

```
ts.resample('5min', closed='right', label='right', loffset='-1s').sum()
```

Out[1084]:

```
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: int32
```

In [1085]:

```
ts.resample('5min').ohlc()
```

Out[1085]:

	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

In [1086]:

```
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 [1087]:

```
frame
```

Out[1087]:

	Colorado	Texas	New York	Ohio
2000-01-05	-2.041716	-0.766231	-0.764284	0.574667
2000-01-12	-1.548400	0.790279	0.026939	-1.207864

In [1088]:

```
df_daily = frame.resample('D').asfreq()
```

In [1089]:

```
df_daily
```

Out[1089]:

	Colorado	Texas	New York	Ohio
2000-01-05	-2.041716	-0.766231	-0.764284	0.574667
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	-1.548400	0.790279	0.026939	-1.207864

In [1090]:

```
frame.resample('D').ffill()
```

Out[1090]:

	Colorado	Texas	New York	Ohio
2000-01-05	-2.041716	-0.766231	-0.764284	0.574667
2000-01-06	-2.041716	-0.766231	-0.764284	0.574667
2000-01-07	-2.041716	-0.766231	-0.764284	0.574667
2000-01-08	-2.041716	-0.766231	-0.764284	0.574667
2000-01-09	-2.041716	-0.766231	-0.764284	0.574667
2000-01-10	-2.041716	-0.766231	-0.764284	0.574667
2000-01-11	-2.041716	-0.766231	-0.764284	0.574667
2000-01-12	-1.548400	0.790279	0.026939	-1.207864

In [1091]:

```
frame.resample('D').ffill(limit=2)
```

Out[1091]:

	Colorado	Texas	New York	Ohio
2000-01-05	-2.041716	-0.766231	-0.764284	0.574667
2000-01-06	-2.041716	-0.766231	-0.764284	0.574667
2000-01-07	-2.041716	-0.766231	-0.764284	0.574667
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	-1.548400	0.790279	0.026939	-1.207864

In [1092]:

```
frame.resample('W-THU').ffill()
```

Out[1092]:

	Colorado	Texas	New York	Ohio
2000-01-06	-2.041716	-0.766231	-0.764284	0.574667
2000-01-13	-1.548400	0.790279	0.026939	-1.207864

In [1093]:

```
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 [1094]:

```
frame[:5]
```

Out[1094]:

	Colorado	Texas	New York	Ohio
2000-01	-0.079418	-0.067526	0.820389	0.039555
2000-02	0.346481	-1.950188	-0.400798	-0.269067
2000-03	0.997121	-0.401784	-0.317724	0.752058
2000-04	0.663380	0.871852	0.746692	-1.607917
2000-05	-1.455519	2.209239	-1.409472	0.605738

In [1095]:

```
annual_frame = frame.resample('A-DEC').mean()
```

In [1096]:

```
annual_frame
```

Out[1096]:

	Colorado	Texas	New York	Ohio
2000	-0.161468	0.310869	0.059121	-0.202676
2001	-0.089176	-0.615332	-0.111393	0.438581

In [1097]:

```
annual_frame.resample('Q-DEC').ffill()
```

Out[1097]:

	Colorado	Texas	New York	Ohio
2000Q1	-0.161468	0.310869	0.059121	-0.202676
2000Q2	-0.161468	0.310869	0.059121	-0.202676
2000Q3	-0.161468	0.310869	0.059121	-0.202676
2000Q4	-0.161468	0.310869	0.059121	-0.202676
2001Q1	-0.089176	-0.615332	-0.111393	0.438581
2001Q2	-0.089176	-0.615332	-0.111393	0.438581
2001Q3	-0.089176	-0.615332	-0.111393	0.438581
2001Q4	-0.089176	-0.615332	-0.111393	0.438581

In [1098]:

```
annual_frame.resample('Q-DEC', convention='end').ffill()
```

Out[1098]:

	Colorado	Texas	New York	Ohio
2000Q4	-0.161468	0.310869	0.059121	-0.202676
2001Q1	-0.161468	0.310869	0.059121	-0.202676
2001Q2	-0.161468	0.310869	0.059121	-0.202676
2001Q3	-0.161468	0.310869	0.059121	-0.202676
2001Q4	-0.089176	-0.615332	-0.111393	0.438581

In [1099]:

```
annual_frame.resample('Q-MAR').ffill()
```

Out[1099]:

	Colorado	Texas	New York	Ohio
2000Q4	-0.161468	0.310869	0.059121	-0.202676
2001Q1	-0.161468	0.310869	0.059121	-0.202676
2001Q2	-0.161468	0.310869	0.059121	-0.202676
2001Q3	-0.161468	0.310869	0.059121	-0.202676
2001Q4	-0.089176	-0.615332	-0.111393	0.438581
2002Q1	-0.089176	-0.615332	-0.111393	0.438581
2002Q2	-0.089176	-0.615332	-0.111393	0.438581
2002Q3	-0.089176	-0.615332	-0.111393	0.438581

11.7 Moving Window Functions

In [1100]:

```
close_px_all = pd.read_csv('examples/stock_px_2.csv', parse_dates=True, index_col=0)
```

In [1101]:

```
close_px = close_px_all[['AAPL', 'MSFT', 'XOM']]
```

In [1102]:

```
close_px = close_px.resample('B').ffill()
```

In [1105]:

```
appl_std250 = close_px.AAPL.rolling(250, min_periods=10).std()
```

In [1106]:

```
appl_std250[5:12]
```

Out[1106]:

```
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
```

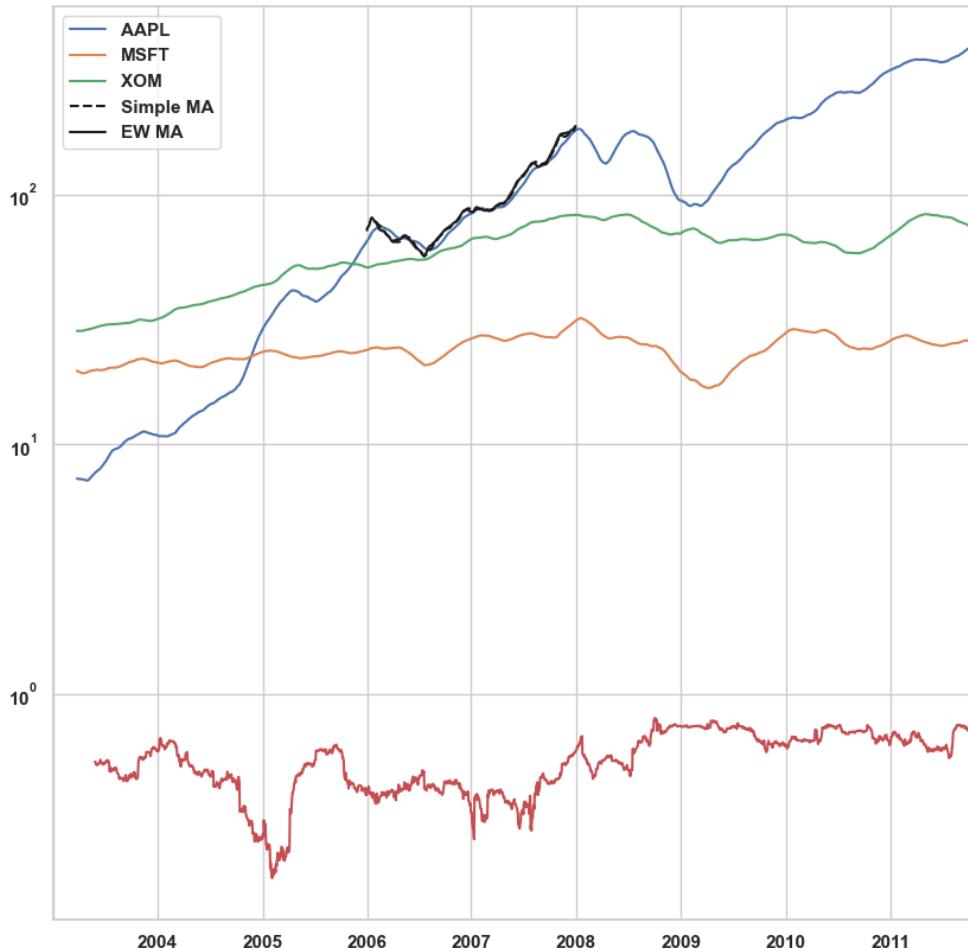
In [1108]:

```
expanding_mean = appl_std250.expanding().mean()
```

In [1109]:

```
close_px.rolling(60).mean().plot(logy=True)
```

<IPython.core.display.Javascript object>



Out[1109]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1670fb4cc88>
```

In [1110]:

```
close_px.rolling('20D').mean()
```

Out[1110]:

	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

In [1111]:

```
aapl_px = close_px.AAPL['2006':'2007']
```

In [1112]:

```
ma60 = aapl_px.rolling(30, min_periods=20).mean()
```

In [1113]:

```
ewma60 = aapl_px.ewm(span=30).mean()
```

In [1114]:

```
ma60.plot(style='k--', label='Simple MA')
```

Out[1114]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1670fb4cc88>
```

In [1115]:

```
ewma60.plot(style='k-', label='EW MA')
```

Out[1115]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1670fb4cc88>
```

In [1116]:

```
plt.legend()
```

Out[1116]:

```
<matplotlib.legend.Legend at 0x16715b85108>
```

In [1117]:

```
spx_px = close_px_all['SPX']
```

In [1118]:

```
spx_rets = spx_px.pct_change()
```

In [458]:

```
returns = close_px.pct_change()
```

In [1120]:

```
corr = returns.AAPL.rolling(125, min_periods=100).corr(spx_rets)
```

In [1121]:

```
corr.plot()
```

Out[1121]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1670fb4cc88>
```

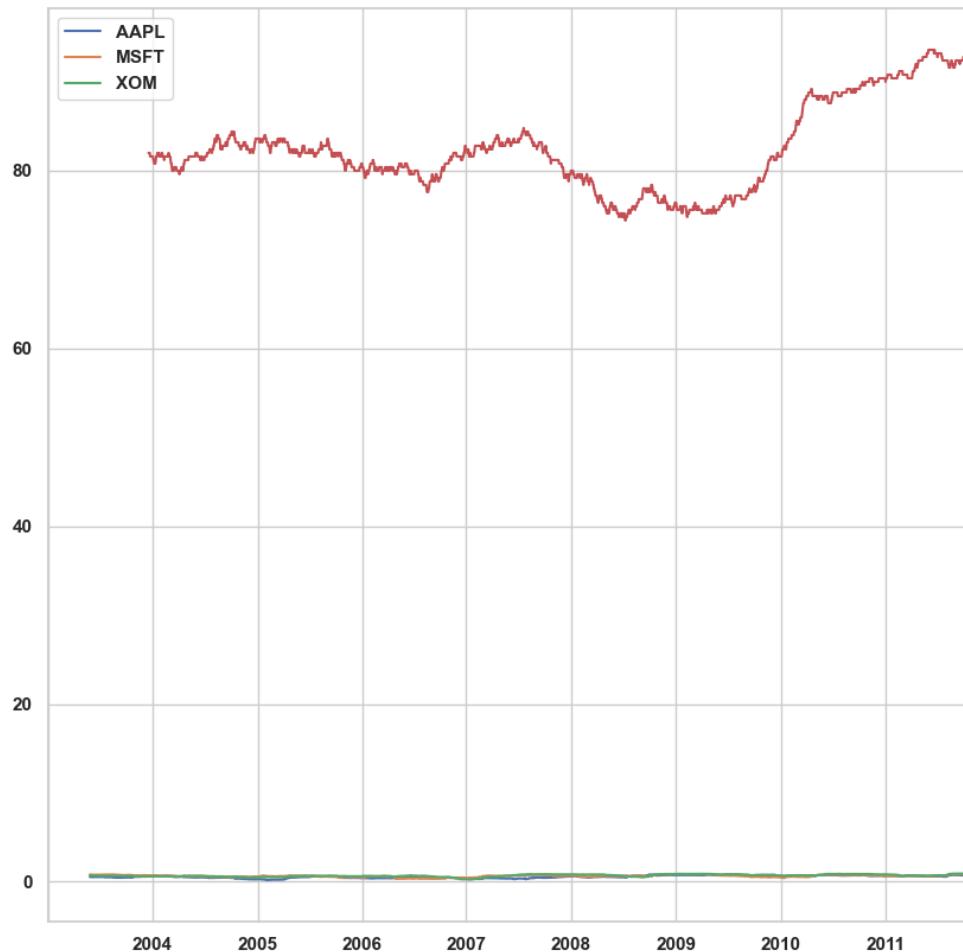
In [1122]:

```
corr = returns.rolling(125, min_periods=100).corr(spx_rets)
```

In [1123]:

```
corr.plot()
```

<IPython.core.display.Javascript object>



Out[1123]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16715b43d88>
```

In [451]:

```
from scipy.stats import percentileofscore
```

In [452]:

```
score_at_2percent = lambda x: percentileofscore(x, 0.02)
```

In [464]:

```
result = returns.AAPL.rolling(250).apply(score_at_2percent)
```

In [463]:

```
result.plot()
```

...

12. ADVANCED PANDAS

12.1 Categorical Data

In [1]:

```
import numpy as np
```

In [2]:

```
import pandas as pd
```

In [3]:

```
values = pd.Series(['apple', 'orange', 'apple', 'apple'] * 2)
```

In [4]:

```
values
```

Out[4]:

```
0    apple
1    orange
2    apple
3    apple
4    apple
5    orange
6    apple
7    apple
dtype: object
```

In [5]:

```
pd.unique(values)
```

Out[5]:

```
array(['apple', 'orange'], dtype=object)
```

In [6]:

```
pd.value_counts(values)
```

Out[6]:

```
apple      6
orange     2
dtype: int64
```

In [7]:

```
values = pd.Series([0, 1, 0, 0] * 2)
```

In [8]:

```
dim = pd.Series(['apple', 'orange'])
```

In [9]:

```
values
```

Out[9]:

```
0      0
1      1
2      0
3      0
4      0
5      1
6      0
7      0
dtype: int64
```

In [10]:

```
dim
```

Out[10]:

```
0    apple
1    orange
dtype: object
```

In [11]:

```
dim.take(values)
```

Out[11]:

```
0    apple
1    orange
0    apple
0    apple
0    apple
1    orange
0    apple
0    apple
dtype: object
```

In [12]:

```
fruits = ['apple', 'orange', 'apple', 'apple'] * 2
```

In [13]:

```
N = len(fruits)
```

In [14]:

```
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 [15]:

```
df
```

Out[15]:

	basket_id	fruit	count	weight
0	0	apple	13	2.341809
1	1	orange	14	0.411976
2	2	apple	5	3.562734
3	3	apple	12	3.384596
4	4	apple	9	3.281791
5	5	orange	3	3.696218
6	6	apple	10	0.375855
7	7	apple	4	0.823114

In [16]:

```
fruit_cat = df['fruit'].astype('category')
```

In [17]:

```
fruit_cat
```

Out[17]:

```
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]
```

In [18]:

```
c = fruit_cat.values
```

In [19]:

```
type(c)
```

Out[19]:

```
pandas.core.arrays.categorical.Categorical
```

In [20]:

```
c.categories
```

Out[20]:

```
Index(['apple', 'orange'], dtype='object')
```

In [21]:

```
c.codes
```

Out[21]:

```
array([0, 1, 0, 0, 0, 1, 0, 0], dtype=int8)
```

In [22]:

```
df['fruit'] = df['fruit'].astype('category')
```

In [23]:

```
df.fruit
```

Out[23]:

```
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]
```

In [24]:

```
my_categories = pd.Categorical(['foo', 'bar', 'baz', 'foo', 'bar'])
```

In [26]:

```
my_categories
```

Out[26]:

```
[foo, bar, baz, foo, bar]
Categories (3, object): [bar, baz, foo]
```

In [27]:

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

In [28]:

```
codes = [0, 1, 2, 0, 0, 1]
```

In [29]:

```
my_cats_2 = pd.Categorical.from_codes(codes, categories)
```

In [30]:

```
my_cats_2
```

Out[30]:

```
[foo, bar, baz, foo, foo, bar]
Categories (3, object): [foo, bar, baz]
```

In [31]:

```
ordered_cat = pd.Categorical.from_codes(codes, categories, ordered=True)
```

In [32]:

```
ordered_cat
```

Out[32]:

```
[foo, bar, baz, foo, foo, bar]
Categories (3, object): [foo < bar < baz]
```

In [33]:

```
my_cats_2.as_ordered()
```

Out[33]:

```
[foo, bar, baz, foo, foo, bar]
Categories (3, object): [foo < bar < baz]
```

In [34]:

```
np.random.seed(12345)
```

In [35]:

```
draws = np.random.randn(1000)
```

In [36]:

```
draws[:5]
```

Out[36]:

```
array([-0.20470766,  0.47894334, -0.51943872, -0.5557303 ,  1.96578057])
```

In [37]:

```
bins = pd.qcut(draws, 4)
```

In [38]:

```
bins
```

Out[38]:

```
[(-0.684, -0.0101], (-0.0101, 0.63], (-0.684, -0.0101], (-0.684, -0.0101],
(0.63, 3.928], ..., (-0.0101, 0.63], (-0.684, -0.0101], (-2.949999999999999
7, -0.684], (-0.0101, 0.63], (0.63, 3.928])
Length: 1000
Categories (4, interval[float64]): [(-2.949999999999997, -0.684] < (-0.684,
-0.0101] < (-0.0101, 0.63] < (0.63, 3.928)]
```

In [39]:

```
bins = pd.qcut(draws, 4, labels=['Q1', 'Q2', 'Q3', 'Q4'])
```

In [40]:

```
bins
```

Out[40]:

```
[Q2, Q3, Q2, Q2, Q4, ..., Q3, Q2, Q1, Q3, Q4]
```

```
Length: 1000
```

```
Categories (4, object): [Q1 < Q2 < Q3 < Q4]
```

In [41]:

```
bins.codes[:10]
```

Out[41]:

```
array([1, 2, 1, 1, 3, 3, 2, 2, 3, 3], dtype=int8)
```

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	-2.949343	-0.685484
1	Q2	250	-0.683066	-0.010115
2	Q3	250	-0.010032	0.628894
3	Q4	250	0.634238	3.927528

In [45]:

```
results['quartile']
```

Out[45]:

```
0    Q1
1    Q2
2    Q3
3    Q4
```

```
Name: quartile, dtype: category
```

```
Categories (4, object): [Q1 < Q2 < Q3 < Q4]
```

In [46]:

```
N = 10000000
```

In [47]:

```
draws = pd.Series(np.random.randn(N))
```

In [48]:

```
labels = pd.Series(['foo', 'bar', 'baz', 'qux'] * (N // 4))
```

In [49]:

```
categories = labels.astype('category')
```

In [50]:

```
labels.memory_usage()
```

Out[50]:

80000128

In [51]:

```
categories.memory_usage()
```

Out[51]:

10000320

In [52]:

```
%time _ = labels.astype('category')
```

Wall time: 771 ms

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]
```

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')
```

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]
```

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
```

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]:

```
cat_s = pd.Series(['a', 'b', 'c', 'd'] * 2, dtype='category')
```

In [67]:

```
pd.get_dummies(cat_s)
```

Out[67]:

	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 Advanced GroupBy Use

In [13]:

```
import numpy as np
```

In [14]:

```
df = pd.DataFrame({'key': ['a', 'b', 'c'] * 4, 'value': np.arange(12.)})
```

In [15]:

```
df
```

Out[15]:

	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 [16]:

```
g = df.groupby('key').value
```

In [17]:

```
g.mean()
```

Out[17]:

```
key
a    4.5
b    5.5
c    6.5
Name: value, dtype: float64
```

In [18]:

```
g.transform(lambda x: x.mean())
```

Out[18]:

```
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 [19]:

```
g.transform('mean')
```

Out[19]:

```
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 [20]:

```
g.transform(lambda x: x * 2)
```

Out[20]:

```
0    0.0
1    2.0
2    4.0
3    6.0
4    8.0
5   10.0
6   12.0
7   14.0
8   16.0
9   18.0
10  20.0
11  22.0
Name: value, dtype: float64
```

In [21]:

```
g.transform(lambda x: x.rank(ascending=False))
```

Out[21]:

```
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
```

In [22]:

```
def normalize(x):
    return (x - x.mean()) / x.std()
```

In [23]:

```
g.transform(normalize)
```

Out[23]:

```
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 [24]:

```
g.apply(normalize)
```

Out[24]:

```
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 [25]:

```
g.transform('mean')
```

Out[25]:

```
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 [26]:

```
normalized = (df['value'] - g.transform('mean')) / g.transform('std')
```

In [27]:

```
normalized
```

Out[27]:

```
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 [28]:

```
N = 15
```

In [29]:

```
times = pd.date_range('2017-05-20 00:00', freq='1min', periods=N)
```

In [30]:

```
df = pd.DataFrame({'time': times, 'value': np.arange(N)})
```

In [31]:

```
df
```

Out[31]:

	time	value
0	2017-05-20 00:00:00	0
1	2017-05-20 00:01:00	1
2	2017-05-20 00:02:00	2
3	2017-05-20 00:03:00	3
4	2017-05-20 00:04:00	4
5	2017-05-20 00:05:00	5
6	2017-05-20 00:06:00	6
7	2017-05-20 00:07:00	7
8	2017-05-20 00:08:00	8
9	2017-05-20 00:09:00	9
10	2017-05-20 00:10:00	10
11	2017-05-20 00:11:00	11
12	2017-05-20 00:12:00	12
13	2017-05-20 00:13:00	13
14	2017-05-20 00:14:00	14

In [32]:

```
df.set_index('time').resample('5min').count()
```

Out[32]:

	value
time	
2017-05-20 00:00:00	5
2017-05-20 00:05:00	5
2017-05-20 00:10:00	5

In [33]:

```
df2 = pd.DataFrame({'time': times.repeat(3),
                    'key': np.tile(['a', 'b', 'c'], N),
                    'value': np.arange(N * 3.)})
```

In [34]:

df2[:7]

Out[34]:

	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

In [35]:

import pandas as pd

Zaman Gruplayıcı Nesnesi

Her bir anahtar değeri için yeniden örneklendirmeyi yapmak için Pandas'a tanıtıyoruz:

time_key = pd.TimeGrouper('5min')

Daha sonra indexini ayarlayıp gruplandırdık:

```
resampled = (df2.set_index('time')
             .groupby(['key', time_key])
             .sum())
resampled.reset_index()
```

12.3 Techniques for Method Chaining

Şimdi ise bir yöntem zincirleme teknüğine bakacağız:

```
df = load_data()
df2 = df[df['col2'] < 0]
df2['col1_demeaned'] = df2['col1'] - df2['col1'].mean()
result = df2.groupby('key').col1_demeaned.std()
```

```
df2 = df.copy()
df2['k'] = v
df2 = df.assign(k=v)
```

```
result = (df2.assign(col1_demeaned=df2.col1 - df2.col2.mean())
          .groupby('key')
          .col1_demeaned.std())
```

```
df = load_data()
```

```
df2 = df[df['col2'] < 0]
df = (load_data()
      [lambda x: x['col2'] < 0].)
```

```
result = (load_data()
          [lambda x: x.col2 < 0].
          .assign(col1_demeaned=lambda x: x.col1 - x.col1.mean())
          .groupby('key')
          .col1_demeaned.std())
```

The Pipe methodu ise şu şekilde uygulanır:

```
a = f(df, arg1=v1)
b = g(a, v2, arg3=v3)
c = h(b, arg4=v4)
```

```
result = (df.pipe(f, arg1=v1)
          .pipe(g, v2, arg3=v3)
          .pipe(h, arg4=v4))
```

```
g = df.groupby(['key1', 'key2'])
df['col1'] = df['col1'] - g.transform('mean')
```

```
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']))
```

13. INTRODUCTION TO MODELING LIBRARIES IN PYTHON

13.1 Interfacing Between pandas and Model Code

In [1]:

```
import pandas as pd
```

In [2]:

```
import numpy as np
```

In [3]:

```
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 [4]:

```
data
```

Out[4]:

	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 [5]:

```
data.columns
```

Out[5]:

```
Index(['x0', 'x1', 'y'], dtype='object')
```

In [6]:

```
data.values
```

Out[6]:

```
array([[ 1. ,  0.01, -1.5 ],
       [ 2. , -0.01,  0. ],
       [ 3. ,  0.25,  3.6 ],
       [ 4. , -4.1 ,  1.3 ],
       [ 5. ,  0. , -2. ]])
```

In [9]:

```
df2 = pd.DataFrame(data.values, columns=['one', 'two', 'three'])
```

In [10]:

```
df2
```

Out[10]:

	one	two	three
0	1.0	0.01	-1.5
1	2.0	-0.01	0.0
2	3.0	0.25	3.6
3	4.0	-4.10	1.3
4	5.0	0.00	-2.0

In [11]:

```
df3 = data.copy()
```

In [12]:

```
df3['strings'] = ['a', 'b', 'c', 'd', 'e']
```

In [13]:

```
df3
```

Out[13]:

	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 [14]:

```
df3.values
```

Out[14]:

```
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)
```

In [15]:

```
model_cols = ['x0', 'x1']
```

In [17]:

```
data.loc[:, model_cols].values
```

Out[17]:

```
array([[ 1. ,  0.01],
       [ 2. , -0.01],
       [ 3. ,  0.25],
       [ 4. , -4.1 ],
       [ 5. ,  0. ]])
```

In [18]:

```
data['category'] = pd.Categorical(['a', 'b', 'a', 'a', 'b'], categories=['a', 'b'])
```

In [19]:

```
data
```

Out[19]:

	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

In [20]:

```
dummies = pd.get_dummies(data.category, prefix='category')
```

In [21]:

```
data_with_dummies = data.drop('category', axis=1).join(dummies)
```

In [22]:

```
data_with_dummies
```

Out[22]:

	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

13.2 Creating Model Descriptions with Patsy

In [23]:

```
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 [24]:

```
data
```

Out[24]:

	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 [25]:

```
import patsy
```

In [26]:

```
y, X = patsy.dmatrices('y ~ x0 + x1', data)
```

In [27]:

```
y
```

Out[27]:

DesignMatrix with shape (5, 1)
y
-1.5
0.0
3.6
1.3
-2.0
Terms:
'y' (column 0)

In [28]:

```
X
```

Out[28]:

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)

In [29]:

```
np.asarray(y)
```

Out[29]:

```
array([[-1.5],  
       [ 0. ],  
       [ 3.6],  
       [ 1.3],  
       [-2. ]])
```

In [30]:

```
np.asarray(X)
```

Out[30]:

```
array([[ 1. ,  1. ,  0.01],  
       [ 1. ,  2. , -0.01],  
       [ 1. ,  3. ,  0.25],  
       [ 1. ,  4. , -4.1 ],  
       [ 1. ,  5. ,  0. ]])
```

In [31]:

```
patsy.dmatrices('y ~ x0 + x1 + 0', data)[1]
```

Out[31]:

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)

In [32]:

```
coef, resid, _, _ = np.linalg.lstsq(X, y)
```

...

In [33]:

```
coef
```

Out[33]:

```
array([[ 0.31290976],  
       [-0.07910564],  
       [-0.26546384]])
```

In [34]:

```
coef = pd.Series(coef.squeeze(), index=X.design_info.column_names)
```

In [35]:

coef

Out[35]:

```
Intercept    0.312910
x0          -0.079106
x1          -0.265464
dtype: float64
```

In [36]:

```
y, X = patsy.dmatrices('y ~ x0 + np.log(np.abs(x1) + 1)', data)
```

In [37]:

X

Out[37]:

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)

In [38]:

```
y, X = patsy.dmatrices('y ~ standardize(x0) + center(x1)', data)
```

In [39]:

X

Out[39]:

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)

In [40]:

```
new_data = pd.DataFrame({ 'x0': [6, 7, 8, 9],
                           'x1': [3.1, -0.5, 0, 2.3],
                           'y': [1, 2, 3, 4]})
```

In [41]:

```
new_X = patsy.build_design_matrices([X.design_info], new_data)
```

In [42]:

```
new_X
```

Out[42]:

[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)]

In [43]:

```
y, X = patsy.dmatrices('y ~ I(x0 + x1)', data)
```

In [44]:

```
X
```

Out[44]:

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)

In [45]:

```
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 [46]:

```
y, X = patsy.dmatrices('v2 ~ key1', data)
```

In [47]:

```
X
```

Out[47]:

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)

In [48]:

```
y, X = patsy.dmatrices('v2 ~ key1 + 0', data)
```

In [49]:

```
X
```

Out[49]:

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)

In [50]:

```
y, X = patsy.dmatrices('v2 ~ C(key2)', data)
```

In [51]:

```
X
```

Out[51]:

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)

In [52]:

```
data['key2'] = data['key2'].map({0: 'zero', 1: 'one'})
```

In [53]:

```
data
```

Out[53]:

	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 [54]:

```
y, X = patsy.dmatrices('v2 ~ key1 + key2', data)
```

In [55]:

X

Out[55]:

```
DesignMatrix with shape (8, 3)
 Intercept key1[T.b] key2[T.zero]
 1          0          1
 1          0          0
 1          1          1
 1          1          0
 1          0          1
 1          1          0
 1          0          1
 1          1          1
```

Terms:

- 'Intercept' (column 0)
- 'key1' (column 1)
- 'key2' (column 2)

In [56]:

```
y, X = patsy.dmatrices('v2 ~ key1 + key2 + key1:key2', data)
```

In [57]:

X

Out[57]:

```
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 Introduction to statsmodels

In [58]:

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

In [59]:

```
def dnorm(mean, variance, size=1):
    if isinstance(size, int):
        size = size,
    return mean + np.sqrt(variance) * np.random.randn(*size)
```

In [60]:

```
np.random.seed(12345)
```

In [61]:

```
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]
```

In [62]:

```
y = np.dot(X, beta) + eps
```

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, -0.12894109])
```

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]])
```

In [67]:

```
model = sm.OLS(y, X)
```

In [68]:

```
results = model.fit()
```

In [69]:

```
results.params
```

Out[69]:

```
array([0.17826108, 0.22303962, 0.50095093])
```

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, 13 Dec 2020 Prob (F-statistic): 7.44e-12
Time: 12:59:34 Log-Likelihood: -34.305
No. Observations: 100 AIC: 74.61
Df Residuals: 97 BIC: 82.42
Df Model: 3
Covariance Type: nonrobust
=====
== coef std err t P>|t| [0.025] 0.97
5]
-----
-- x1 0.1783 0.053 3.364 0.001 0.073 0.2
83
x2 0.2230 0.046 4.818 0.000 0.131 0.3
15
x3 0.5010 0.080 6.237 0.000 0.342 0.6
60
=====
== Omnibus: 4.662 Durbin-Watson: 2.2
01
Prob(Omnibus): 0.097 Jarque-Bera (JB): 4.0
98
Skew: 0.481 Prob(JB): 0.1
29
Kurtosis: 3.243 Cond. No. 1.
74
=====
==

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is corre-
ctly specified.
```

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

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
```

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
```

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 = dnorm(0, 0.1, N)
for i in range(N):
    new_x = values[-1] * b0 + values[-2] * b1 + noise[i]
    values.append(new_x)
```

In [81]:

```
MAXLAGS = 5
```

In [82]:

```
model = sm.tsa.AR(values)
```

...

In [83]:

```
results = model.fit(MAXLAGS)
```

In [84]:

```
results.params
```

Out[84]:

```
array([-0.00616093,  0.78446347, -0.40847891, -0.01364148,  0.01496872,
       0.01429462])
```

13.4 Introduction to scikit-learn

In [39]:

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

In [40]:

```
train = pd.read_csv('datasets/titanic/train.csv')
```

In [41]:

```
test = pd.read_csv('datasets/titanic/test.csv')
```

In [42]:

```
train[:4]
```

Out[42]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	(
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
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	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	

In [43]:

```
train.isnull().sum()
```

Out[43]:

```
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 [44]:

```
test.isnull().sum()
```

Out[44]:

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [45]:

```
impute_value = train['Age'].median()
```

In [46]:

```
train['Age'] = train['Age'].fillna(impute_value)
```

In [47]:

```
test['Age'] = test['Age'].fillna(impute_value)
```

In [48]:

```
train['IsFemale'] = (train['Sex'] == 'female').astype(int)
```

In [49]:

```
test['IsFemale'] = (test['Sex'] == 'female').astype(int)
```

In [50]:

```
predictors = ['Pclass', 'IsFemale', 'Age']
```

In [51]:

```
X_train = train[predictors].values
```

In [52]:

```
X_test = test[predictors].values
```

In [53]:

```
y_train = train['Survived'].values
```

In [54]:

```
X_train[:5]
```

Out[54]:

```
array([[ 3.,  0., 22.],
       [ 1.,  1., 38.],
       [ 3.,  1., 26.],
       [ 1.,  1., 35.],
       [ 3.,  0., 35.]])
```

In [55]:

```
y_train[:5]
```

Out[55]:

```
array([0, 1, 1, 1, 0], dtype=int64)
```

In [56]:

```
from sklearn.linear_model import LogisticRegression
```

In [57]:

```
model = LogisticRegression()
```

In [58]:

```
model.fit(X_train, y_train)
```

Out[58]:

```
LogisticRegression()
```

In [59]:

```
y_predict = model.predict(X_test)
```

In [60]:

```
y_predict[:10]
```

Out[60]:

```
array([0, 0, 0, 0, 1, 0, 1, 0, 1, 0], dtype=int64)
```

In [63]:

```
from sklearn.linear_model import LogisticRegressionCV
```

In [64]:

```
model_cv = LogisticRegressionCV(10)
```

...

In [65]:

```
model_cv.fit(X_train, y_train)
```

Out[65]:

```
LogisticRegressionCV()
```

In [66]:

```
from sklearn.model_selection import cross_val_score
```

In [67]:

```
model = LogisticRegression(C=10)
```

In [68]:

```
scores = cross_val_score(model, X_train, y_train, cv=4)
```

In [69]:

```
scores
```

Out[69]:

```
array([0.77578475, 0.79820628, 0.77578475, 0.78828829])
```

14. DATA ANALYSIS EXAMPLES

14.1 1.USA.gov Data from Bitly

In [1]:

```
path = 'datasets/bitly_usagov/example.txt'
```

In [2]:

```
open(path).readline()
```

Out[2]:

```
{ "a": "Mozilla\\/5.0 (Windows NT 6.1; WOW64) AppleWebKit\\/535.11 (KHTML, like Gecko) Chrome\\/17.0.963.78 Safari\\/535.11", "c": "US", "nk": 1, "tz": "America\\/New_York", "gr": "MA", "g": "A6qOVH", "h": "wfLQtf", "l": "orofrog", "al": "en-US,en;q=0.8", "hh": "1.usa.gov", "r": "http:\\\\www.facebook.com\\/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 [3]:

```
import json
path = 'datasets/bitly_usagov/example.txt'
records = [json.loads(line) for line in open(path)]
```

In [4]:

```
records[0]
```

Out[4]:

```
{'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]}
```

In [5]:

```
time_zones = [rec['tz'] for rec in records]
```

```
-----  
KeyError                                                 Traceback (most recent call last)  
<ipython-input-5-f3fbcb37f129> in <module>  
----> 1 time_zones = [rec['tz'] for rec in records]  
  
<ipython-input-5-f3fbcb37f129> in <listcomp>(.0)  
----> 1 time_zones = [rec['tz'] for rec in records]
```

KeyError: 'tz'

In [6]:

```
time_zones = [rec['tz'] for rec in records if 'tz' in rec]
```

In [7]:

```
time_zones[:10]
```

Out[7]:

```
['America/New_York',
 'America/Denver',
 'America/New_York',
 'America/Sao_Paulo',
 'America/New_York',
 'America/New_York',
 'Europe/Warsaw',
 '',
 '',
 '']
```

In [8]:

```
def get_counts(sequence):
    counts = {}
    for x in sequence:
        if x in counts:
            counts[x] += 1
        else:
            counts[x] = 1
    return counts
```

In [9]:

```
from collections import defaultdict
```

In [10]:

```
def get_counts2(sequence):
    counts = defaultdict(int)
    for x in sequence:
        counts[x] += 1
    return counts
```

In [11]:

```
counts = get_counts(time_zones)
```

In [12]:

```
counts['America/New_York']
```

Out[12]:

```
1251
```

In [13]:

```
len(time_zones)
```

Out[13]:

```
3440
```

In [14]:

```
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 [15]:

```
top_counts(counts)
```

Out[15]:

```
[(33, 'America/Sao_Paulo'),  
(35, 'Europe/Madrid'),  
(36, 'Pacific/Honolulu'),  
(37, 'Asia/Tokyo'),  
(74, 'Europe/London'),  
(191, 'America/Denver'),  
(382, 'America/Los_Angeles'),  
(400, 'America/Chicago'),  
(521, ''),  
(1251, 'America/New_York')]
```

In [16]:

```
from collections import Counter
```

In [17]:

```
counts = Counter(time_zones)
```

In [18]:

```
counts.most_common(10)
```

Out[18]:

```
[('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)]
```

In [19]:

```
import pandas as pd
```

In [20]:

```
frame = pd.DataFrame(records)
```

In [21]:

```
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 [22]:

```
frame['tz'][::10]
```

Out[22]:

```
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
```

In [23]:

```
tz_counts = frame['tz'].value_counts()
```

In [24]:

```
tz_counts[:10]
```

Out[24]:

```
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
```

In [25]:

```
clean_tz = frame['tz'].fillna('Missing')
```

In [26]:

```
clean_tz[clean_tz == ''] = 'Unknown'
```

In [27]:

```
tz_counts = clean_tz.value_counts()
```

In [28]:

```
tz_counts[:10]
```

Out[28]:

```
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
```

In [29]:

```
import seaborn as sns
```

In [30]:

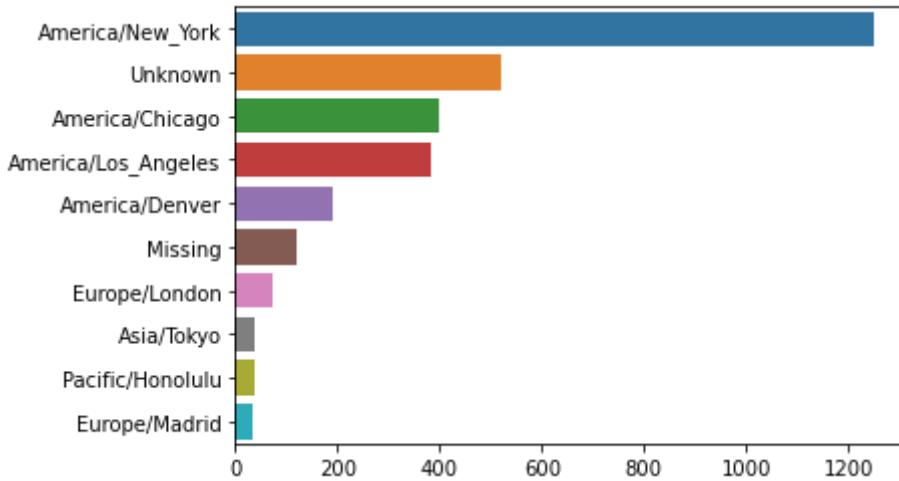
```
subset = tz_counts[:10]
```

In [31]:

```
sns.barplot(y=subset.index, x=subset.values)
```

Out[31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c9e9f2db48>
```



In [32]:

```
frame['a'][1]
```

Out[32]:

```
'GoogleMaps/RochesterNY'
```

In [33]:

```
frame['a'][50]
```

Out[33]:

```
'Mozilla/5.0 (Windows NT 5.1; rv:10.0.2) Gecko/20100101 Firefox/10.0.2'
```

In [34]:

```
frame['a'][51][:50]
```

Out[34]:

```
'Mozilla/5.0 (Linux; U; Android 2.2.2; en-us; LG-P9'
```

In [35]:

```
results = pd.Series([x.split()[0] for x in frame.a.dropna()])
```

In [36]:

```
results[:5]
```

Out[36]:

```
0           Mozilla/5.0
1   GoogleMaps/RochesterNY
2           Mozilla/4.0
3           Mozilla/5.0
4           Mozilla/5.0
dtype: object
```

In [37]:

```
results.value_counts()[:8]
```

Out[37]:

```
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
```

In [38]:

```
cframe = frame[frame.a.notnull()]
```

In [39]:

```
import numpy as np
```

In [40]:

```
cframe['os'] = np.where(cframe['a'].str.contains('Windows'), 'Windows', 'Not Windows')
```

...

In [41]:

```
cframe['os'][:5]
```

Out[41]:

```
0      Windows
1  Not Windows
2      Windows
3  Not Windows
4      Windows
Name: os, dtype: object
```

In [42]:

```
by_tz_os = cframe.groupby(['tz', 'os'])
```

In [43]:

```
agg_counts = by_tz_os.size().unstack().fillna(0)
```

In [44]:

```
agg_counts[:10]
```

Out[44]:

	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	

In [45]:

```
indexer = agg_counts.sum(1).argsort()
```

In [46]:

```
indexer[:10]
```

Out[46]:

```
tz
24
Africa/Cairo      20
Africa/Casablanca 21
Africa/Ceuta       92
Africa/Johannesburg 87
Africa/Lusaka      53
America/Anchorage  54
America/Argentina/Buenos_Aires 57
America/Argentina/Cordoba   26
America/Argentina/Mendoza   55
dtype: int64
```

In [47]:

```
count_subset = agg_counts.take(indexer[-10:])
```

In [48]:

```
count_subset
```

Out[48]:

tz	os	Not Windows	Windows
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	

In [49]:

```
agg_counts.sum(1).nlargest(10)
```

Out[49]:

```
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
```

In [50]:

```
count_subset = count_subset.stack()
```

In [51]:

```
count_subset.name = 'total'
```

In [52]:

```
count_subset = count_subset.reset_index()
```

In [53]:

count_subset[:10]

Out[53]:

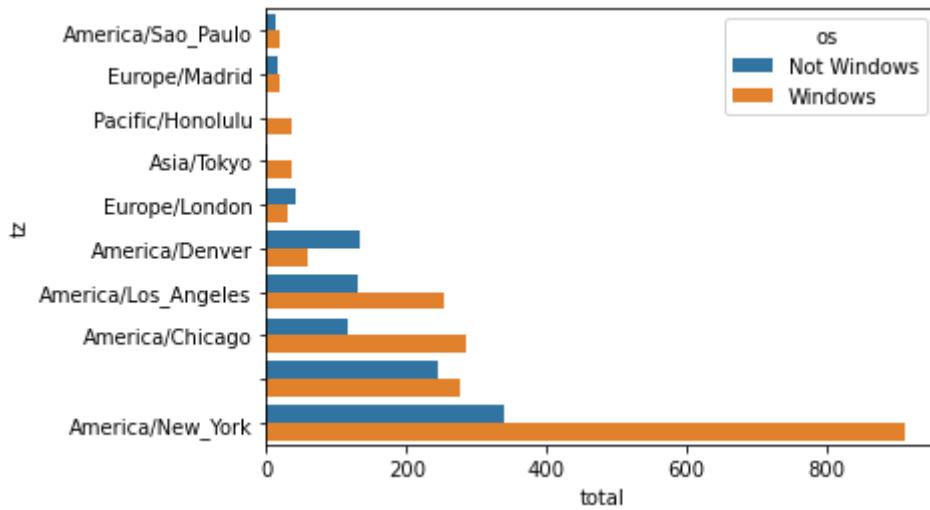
	tz	os	total
0	America/Sao_Paulo	Not Windows	13.0
1	America/Sao_Paulo	Windows	20.0
2	Europe/Madrid	Not Windows	16.0
3	Europe/Madrid	Windows	19.0
4	Pacific/Honolulu	Not Windows	0.0
5	Pacific/Honolulu	Windows	36.0
6	Asia/Tokyo	Not Windows	2.0
7	Asia/Tokyo	Windows	35.0
8	Europe/London	Not Windows	43.0
9	Europe/London	Windows	31.0

In [54]:

sns.barplot(x='total', y='tz', hue='os', data=count_subset)

Out[54]:

<matplotlib.axes._subplots.AxesSubplot at 0x1c9ed612a88>



In [55]:

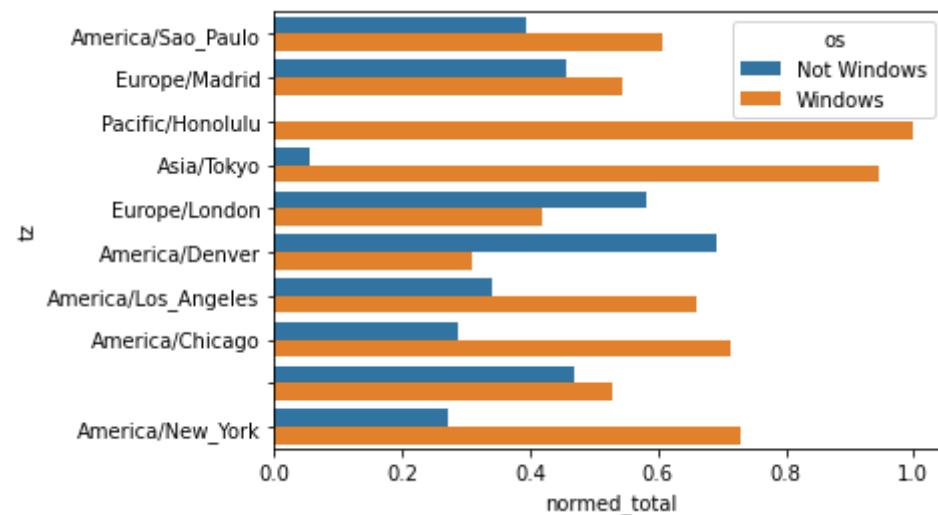
```
def norm_total(group):
    group['normed_total'] = group.total / group.total.sum()
    return group
results = count_subset.groupby('tz').apply(norm_total)
```

In [56]:

```
sns.barplot(x='normed_total', y='tz', hue='os', data=results)
```

Out[56]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c9ed7b6d88>
```



In [57]:

```
g = count_subset.groupby('tz')
```

In [58]:

```
results2 = count_subset.total / g.total.transform('sum')
```

14.2 MovieLens 1M Dataset

In [59]:

```
import pandas as pd
```

In [60]:

```
pd.options.display.max_rows = 10
```

In [61]:

```
unames = ['user_id', 'gender', 'age', 'occupation', 'zip']
users = pd.read_table('datasets/movielens/users.dat', sep='::',
                      header=None, names=unames)
```

...

In [62]:

```
rnames = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_table('datasets/movielens/ratings.dat', sep='::',
                        header=None, names=rnames)
```

...

In [63]:

```
mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('datasets/movielens/movies.dat', sep='::',
                       header=None, names=mnames)
```

...

In [64]:

```
users[:5]
```

Out[64]:

	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 [65]:

```
ratings[:5]
```

Out[65]:

	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 [66]:

```
movies[:5]
```

Out[66]:

	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 [67]:

```
ratings
```

Out[67]:

	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

In [68]:

```
data = pd.merge(pd.merge(ratings, users), movies)
```

In [69]:

data

Out[69]:

	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) C
1000208	5938	2909	4	957273353	M	25		1	35401 Five Wives, Three Secretaries and Me (1998)

1000209 rows × 10 columns



In [70]:

```
data.iloc[0]
```

Out[70]:

```
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
```

In [71]:

```
mean_ratings = data.pivot_table('rating', index='title', columns='gender', aggfunc='mean')
```

In [72]:

```
mean_ratings[:5]
```

Out[72]:

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

In [73]:

```
ratings_by_title = data.groupby('title').size()
```

In [74]:

```
ratings_by_title[:10]
```

Out[74]:

```
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 [75]:

```
active_titles = ratings_by_title.index[ratings_by_title >= 250]
```

In [76]:

```
active_titles
```

Out[76]:

```
Index([''burbs, The (1989)', '10 Things I Hate About You (1999)',
       '101 Dalmatians (1961)', '101 Dalmatians (1996)', '12 Angry Men (195
    7)',
       '13th Warrior, The (1999)', '2 Days in the Valley (1996)',
       '20,000 Leagues Under the Sea (1954)', '2001: A Space Odyssey (196
    8)',
       '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)
```

In [77]:

```
mean_ratings = mean_ratings.loc[active_titles]
```

In [78]:

```
mean_ratings
```

Out[78]:

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

In [79]:

```
top_female_ratings = mean_ratings.sort_values(by='F', ascending=False)
```

In [80]:

top_female_ratings[:10]

Out[80]:

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

In [81]:

mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']

In [82]:

sorted_by_diff = mean_ratings.sort_values(by='diff')

In [83]:

sorted_by_diff[:10]

Out[83]:

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

In [84]:

```
sorted_by_diff[::-1][:10]
```

Out[84]:

	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

In [85]:

```
rating_std_by_title = data.groupby('title')['rating'].std()
```

In [86]:

```
rating_std_by_title = rating_std_by_title.loc[active_titles]
```

In [87]:

```
rating_std_by_title.sort_values(ascending=False)[:10]
```

Out[87]:

title	
Dumb & Dumber (1994)	1.321333
Blair Witch Project, The (1999)	1.316368
Natural Born Killers (1994)	1.307198
Tank Girl (1995)	1.277695
Rocky Horror Picture Show, The (1975)	1.260177
Eyes Wide Shut (1999)	1.259624
Evita (1996)	1.253631
Billy Madison (1995)	1.249970
Fear and Loathing in Las Vegas (1998)	1.246408
Bicentennial Man (1999)	1.245533
Name: rating, dtype: float64	

14.3 US Baby Names 1880–2010

In [89]:

```
!head -n 10 datasets/babynames/yob1880.txt
```

...

In [90]:

```
import pandas as pd
```

In [91]:

```
names1880 = pd.read_csv('datasets/babynames/yob1880.txt', names=['name', 'sex', 'births'])
```

In [92]:

```
names1880
```

Out[92]:

	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

In [93]:

```
names1880.groupby('sex').births.sum()
```

Out[93]:

```
sex
F    90993
M   110493
Name: births, dtype: int64
```

In [94]:

```
years = range(1880, 2011)
```

In [95]:

```
pieces = []
columns = ['name', 'sex', 'births']
```

In [96]:

```
for year in years:
    path = 'datasets/babynames/yob%d.txt' % year
    frame = pd.read_csv(path, names=columns)
    frame['year'] = year
    pieces.append(frame)
```

In [97]:

```
names = pd.concat(pieces, ignore_index=True)
```

In [98]:

```
names
```

Out[98]:

	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

In [99]:

```
total_births = names.pivot_table('births', index='year', columns='sex', aggfunc=sum)
```

In [100]:

```
total_births.tail()
```

Out[100]:

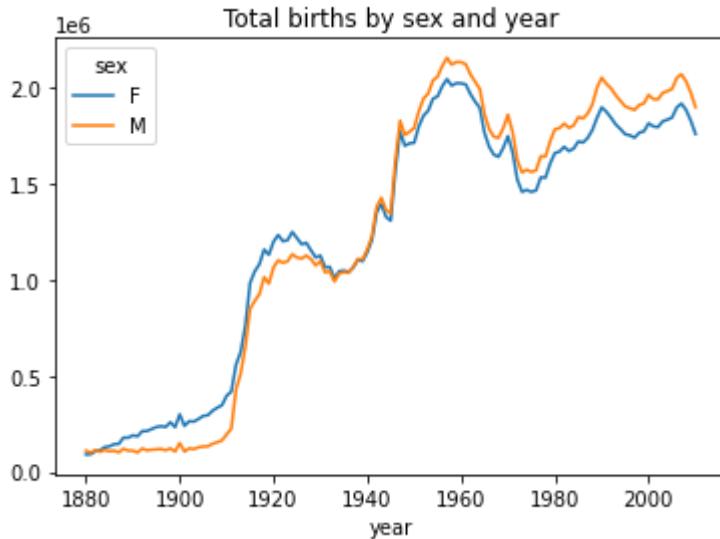
sex	F	M
year		
2006	1896468	2050234
2007	1916888	2069242
2008	1883645	2032310
2009	1827643	1973359
2010	1759010	1898382

In [101]:

```
total_births.plot(title='Total births by sex and year')
```

Out[101]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c98354c788>
```



In [102]:

```
def add_prop(group):
    group['prop'] = group.births / group.births.sum()
    return group
names = names.groupby(['year', 'sex']).apply(add_prop)
```

In [103]:

names

Out[103]:

		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

In [104]:

```
names.groupby(['year', 'sex']).prop.sum()
```

Out[104]:

```
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
```

In [105]:

```
def get_top1000(group):
    return group.sort_values(by='births', ascending=False)[:1000]
grouped = names.groupby(['year', 'sex'])
top1000 = grouped.apply(get_top1000)
top1000.reset_index(inplace=True, drop=True)
```

In [106]:

```
pieces = []
for year, group in names.groupby(['year', 'sex']):
    pieces.append(group.sort_values(by='births', ascending=False)[:1000])
top1000 = pd.concat(pieces, ignore_index=True)
```

In [107]:

```
top1000
```

Out[107]:

	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

In [108]:

```
boys = top1000[top1000.sex == 'M']
```

In [109]:

```
girls = top1000[top1000.sex == 'F']
```

In [110]:

```
total_births = top1000.pivot_table('births', index='year', columns='name', aggfunc=sum)
```

In [111]:

```
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 [112]:

```
subset = total_births[['John', 'Harry', 'Mary', 'Marilyn']]
```

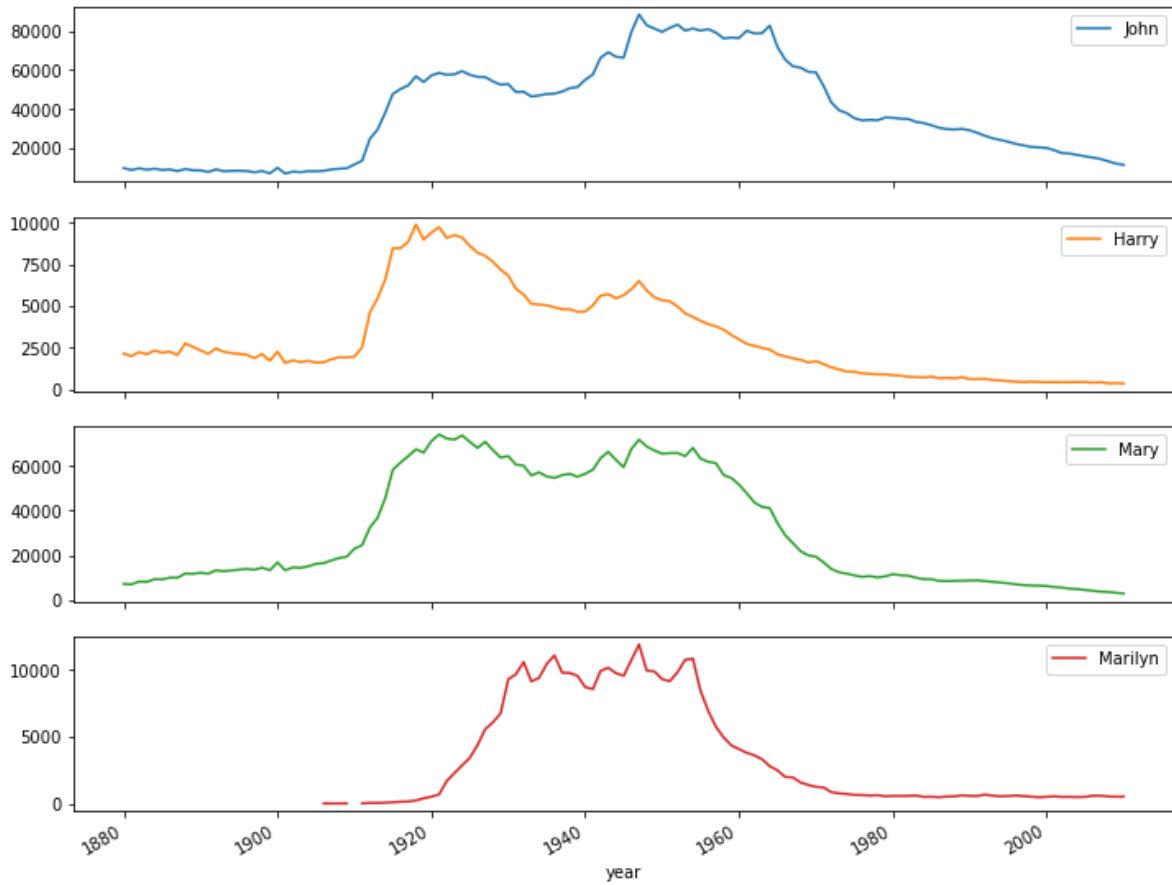
In [113]:

```
subset.plot(subplots=True, figsize=(12, 10), grid=False, title="Number of births per year")
```

Out[113]:

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x000001C9892D17C8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001C999F2F208>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001C999F64808>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001C999F99CC8
    >],
      dtype=object)
```

Number of births per year



In [114]:

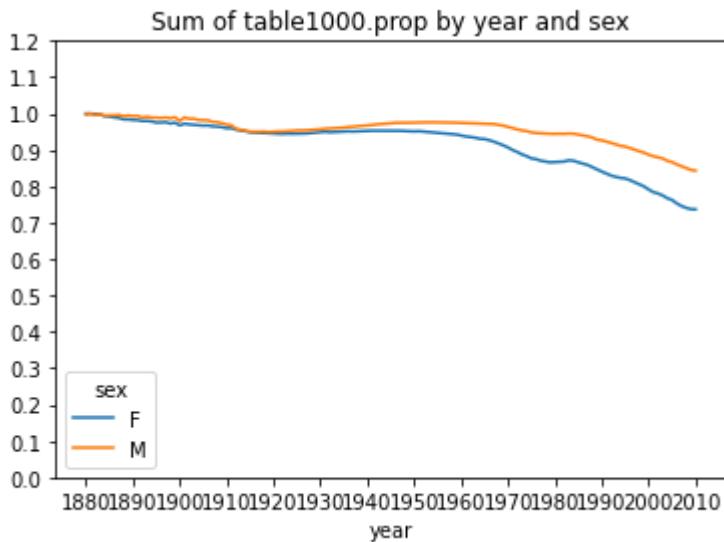
```
table = top1000.pivot_table('prop', index='year', columns='sex', aggfunc=sum)
```

In [115]:

```
table.plot(title='Sum of table1000.prop by year and sex', yticks=np.linspace(0, 1.2, 13), x
```

Out[115]:

<matplotlib.axes._subplots.AxesSubplot at 0x1c99a05e608>



In [116]:

```
df = boys[boys.year == 2010]
```

In [117]:

```
df
```

Out[117]:

	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

In [118]:

```
prop_cumsum = df.sort_values(by='prop', ascending=False).prop.cumsum()
```

In [119]:

```
prop_cumsum[:10]
```

Out[119]:

```
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 [120]:

```
prop_cumsum.values.searchsorted(0.5)
```

Out[120]:

```
116
```

In [121]:

```
df = boys[boys.year == 1900]
```

In [122]:

```
in1900 = df.sort_values(by='prop', ascending=False).prop.cumsum()
```

In [123]:

```
in1900.values.searchsorted(0.5) + 1
```

Out[123]:

```
25
```

In [124]:

```
def get_quantile_count(group, q=0.5):
    group = group.sort_values(by='prop', ascending=False)
    return group.prop.cumsum().values.searchsorted(q) + 1
```

In [125]:

```
diversity = top1000.groupby(['year', 'sex']).apply(get_quantile_count)
diversity = diversity.unstack('sex')
```

In [126]:

```
diversity.head()
```

Out[126]:

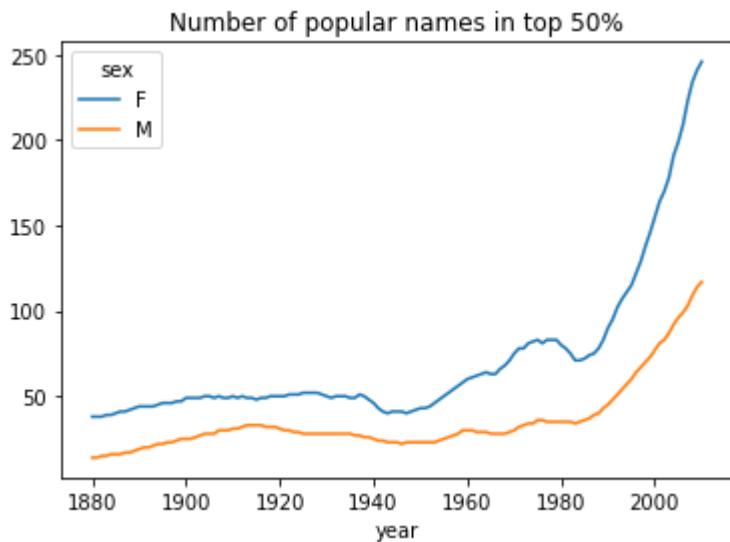
sex	F	M
year		
1880	38	14
1881	38	14
1882	38	15
1883	39	15
1884	39	16

In [127]:

```
diversity.plot(title="Number of popular names in top 50%)")
```

Out[127]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c99a579488>
```



In [128]:

```
get_last_letter = lambda x: x[-1]
last_letters = names.name.map(get_last_letter)
last_letters.name = 'last_letter'
```

In [129]:

```
table = names.pivot_table('births', index=last_letters, columns=['sex', 'year'], aggfunc=sum)
```

In [130]:

```
subtable = table.reindex(columns=[1910, 1960, 2010], level='year')
```

In [131]:

```
subtable.head()
```

Out[131]:

sex	F			M		
	year	1910	1960	2010	1910	1960
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

In [132]:

```
subtable.sum()
```

Out[132]:

```
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 [133]:

```
letter_prop = subtable / subtable.sum()
```

In [134]:

```
letter_prop
```

Out[134]:

sex	F			M		
	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

In [135]:

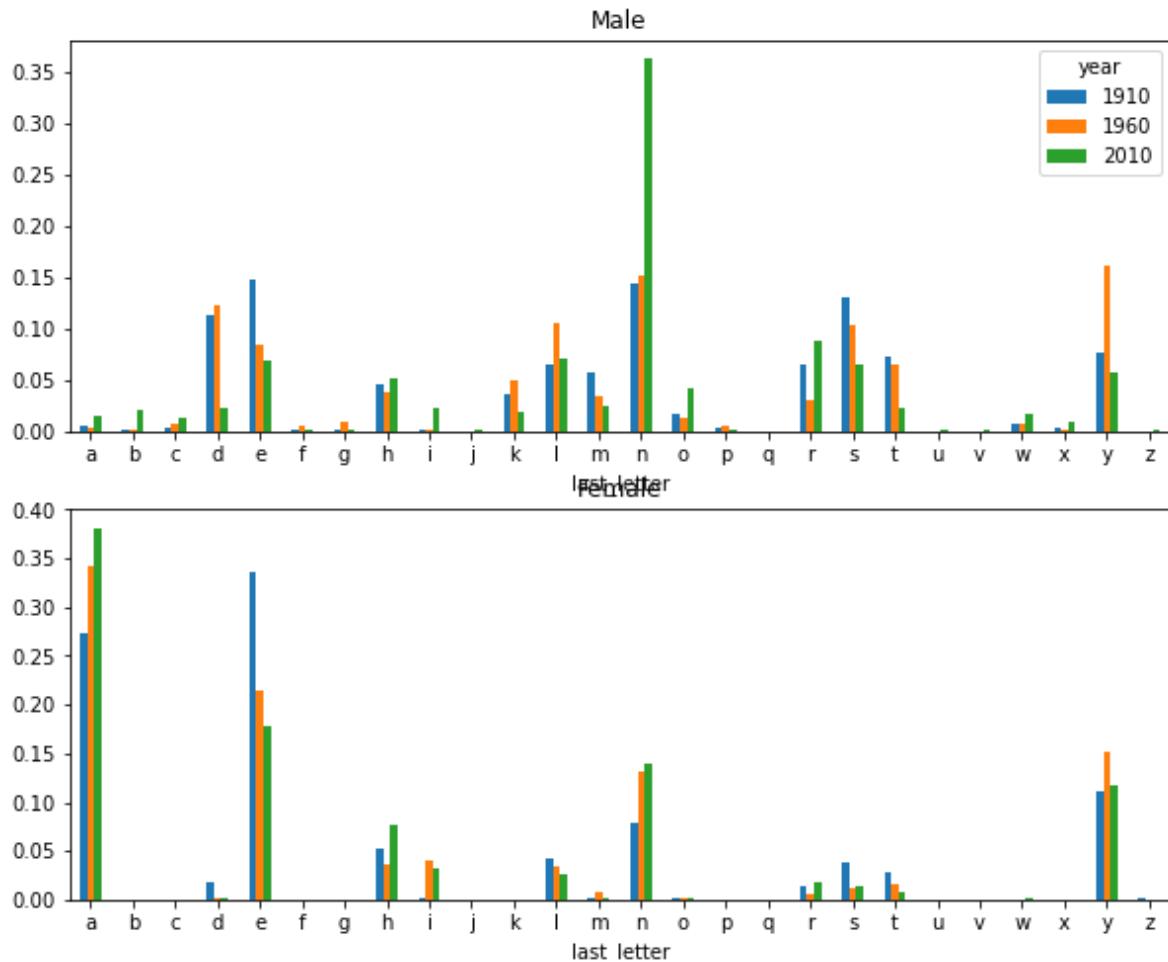
```
import matplotlib.pyplot as plt
```

In [136]:

```
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[136]:

<matplotlib.axes._subplots.AxesSubplot at 0x1c99a637d48>



In [137]:

```
letter_prop = table / table.sum()
```

In [138]:

```
dny_ts = letter_prop.loc[['d', 'n', 'y'], 'M'].T
```

In [139]:

```
dny_ts.head()
```

Out[139]:

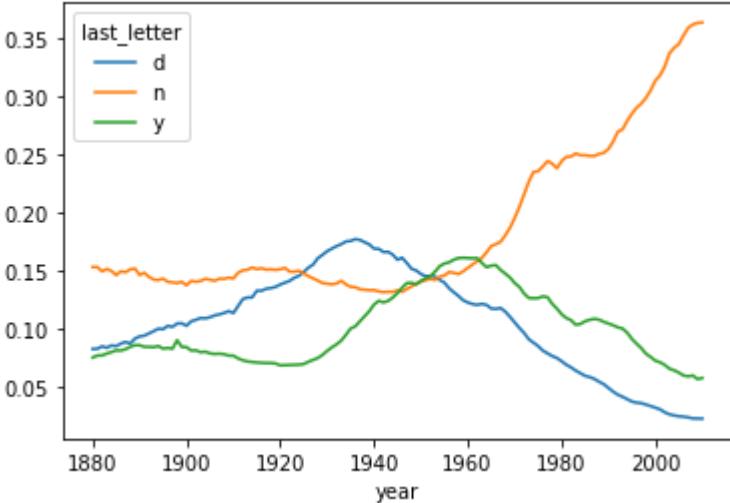
last_letter	d	n	y
year			
1880	0.083055	0.153213	0.075760
1881	0.083247	0.153214	0.077451
1882	0.085340	0.149560	0.077537
1883	0.084066	0.151646	0.079144
1884	0.086120	0.149915	0.080405

In [140]:

```
dny_ts.plot()
```

Out[140]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c99a8be4c8>
```



In [141]:

```
all_names = pd.Series(top1000.name.unique())
```

In [142]:

```
lesley_like = all_names[all_names.str.lower().str.contains('lesl')]
```

In [143]:

```
lesley_like
```

Out[143]:

```
632      Leslie
2294     Lesley
4262     Leslee
4728     Lesli
6103     Lesly
dtype: object
```

In [144]:

```
filtered = top1000[top1000.name.isin(lesley_like)]
```

In [145]:

```
filtered.groupby('name').births.sum()
```

Out[145]:

```
name
Leslee      1082
Lesley     35022
Lesli       929
Leslie    370429
Lesly      10067
Name: births, dtype: int64
```

In [146]:

```
table = filtered.pivot_table('births', index='year', columns='sex', aggfunc='sum')
```

In [147]:

```
table = table.div(table.sum(1), axis=0)
```

In [148]:

```
table.tail()
```

Out[148]:

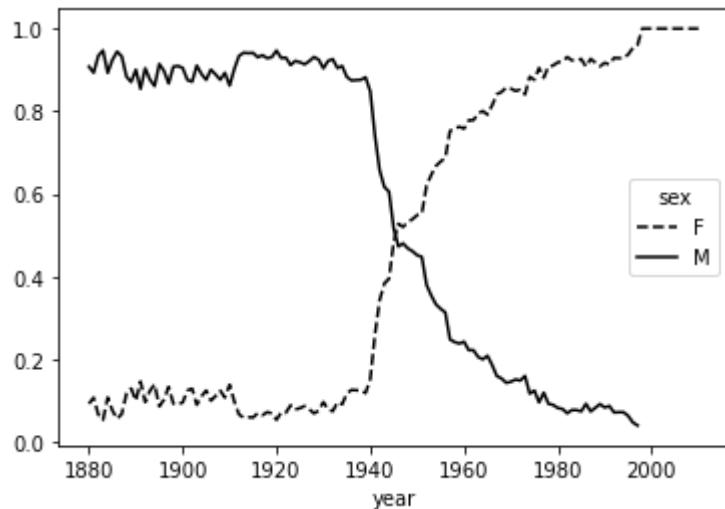
sex	F	M
year		
2006	1.0	NaN
2007	1.0	NaN
2008	1.0	NaN
2009	1.0	NaN
2010	1.0	NaN

In [149]:

```
table.plot(style={'M': 'k-', 'F': 'k--'})
```

Out[149]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c99a98ad48>
```



14.4 USDA Food Database

In [150]:

```
import json
```

In [151]:

```
db = json.load(open('datasets/usda_food/database.json'))
```

In [152]:

```
len(db)
```

Out[152]:

```
6636
```

In [153]:

```
db[0].keys()
```

Out[153]:

```
dict_keys(['id', 'description', 'tags', 'manufacturer', 'group', 'portions',  
'nutrients'])
```

In [154]:

```
db[0]['nutrients'][0]
```

Out[154]:

```
{'value': 25.18,
 'units': 'g',
 'description': 'Protein',
 'group': 'Composition'}
```

In [155]:

```
nutrients = pd.DataFrame(db[0]['nutrients'])
```

In [156]:

```
nutrients[:7]
```

Out[156]:

	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

In [157]:

```
info_keys = ['description', 'group', 'id', 'manufacturer']
```

In [158]:

```
info = pd.DataFrame(db, columns=info_keys)
```

In [159]:

```
info[:5]
```

Out[159]:

	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 [160]:

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
```

In [161]:

pd.value_counts(info.group)[:10]

Out[161]:

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
Fruits and Fruit Juices	328
Pork Products	328

Name: group, dtype: int64

In [162]:

nutrients

Out[162]:

	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

In [163]:

```
nutrients.duplicated().sum()
```

Out[163]:

108

In [164]:

```
nutrients = nutrients.drop_duplicates()
```

In [165]:

```
col_mapping = {'description' : 'food', 'group' : 'fgroup'}
```

In [166]:

```
info = info.rename(columns=col_mapping, copy=False)
```

In [167]:

```
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 [168]:

```
col_mapping = {'description' : 'nutrient', 'group' : 'nutgroup'}
```

In [169]:

```
nutrients = nutrients.rename(columns=col_mapping, copy=False)
```

In [170]:

nutrients

Out[170]:

	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

Json dosyasındaki veriler kitaba göre kesik geldi bu yüzden sütun birleştirme yapamadım:

```
ndata = pd.merge(nutrients, info, on='id', how='outer')
ndata.iloc[30000]
```

```
result = ndata.groupby(['nutrient', 'fgroup'])['value'].quantile(0.5)
result['Zinc', 'Zn'].sort_values().plot(kind='barh')
```

```
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']]
max_foods.food = max_foods.food.str[:50]
max_foods.loc['Amino Acids']['food']
```

14.5 2012 Federal Election Commission Database

In [172]:

```
fec = pd.read_csv('datasets/fec/P00000001-ALL.csv')
```

...

In [173]:

```
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
```

In [174]:

```
fec.iloc[123456]
```

Out[174]:

```
cmte_id          C00431445
cand_id          P80003338
cand_nm          Obama, Barack
contbr_nm        ELLMAN, IRA
contbr_city      TEMPE
...
receipt_desc     NaN
memo_cd          NaN
memo_text         NaN
form_tp          SA17A
file_num         772372
Name: 123456, Length: 16, dtype: object
```

In [175]:

```
unique_cands = fec.cand_nm.unique()
```

In [176]:

```
unique_cands
```

Out[176]:

```
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 [177]:

```
unique_cands[2]
```

Out[177]:

```
'Obama, Barack'
```

In [178]:

```
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'}
```

In [179]:

```
fec.cand_nm[123456:123461]
```

Out[179]:

```
123456    Obama, Barack
123457    Obama, Barack
123458    Obama, Barack
123459    Obama, Barack
123460    Obama, Barack
Name: cand_nm, dtype: object
```

In [180]:

```
fec.cand_nm[123456:123461].map(parties)
```

Out[180]:

```
123456    Democrat
123457    Democrat
123458    Democrat
123459    Democrat
123460    Democrat
Name: cand_nm, dtype: object
```

In [181]:

```
fec['party'] = fec.cand_nm.map(parties)
```

In [182]:

```
fec['party'].value_counts()
```

Out[182]:

```
Democrat      593746  
Republican    407985  
Name: party, dtype: int64
```

In [183]:

```
(fec.contb_receipt_amt > 0).value_counts()
```

Out[183]:

```
True       991475  
False      10256  
Name: contb_receipt_amt, dtype: int64
```

In [185]:

```
fec = fec[fec.contb_receipt_amt > 0]
```

In [186]:

```
fec_mrbo = fec[fec.cand_nm.isin(['Obama, Barack', 'Romney, Mitt'])]
```

In [187]:

```
fec.contbr_occupation.value_counts()[:10]
```

Out[187]:

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

In [188]:

```
occ_mapping = {  
    'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',  
    'INFORMATION REQUESTED' : 'NOT PROVIDED',  
    'INFORMATION REQUESTED (BEST EFFORTS)' : 'NOT PROVIDED',  
    'C.E.O.' : 'CEO'  
}
```

In [189]:

```
f = lambda x: occ_mapping.get(x, x)
fec.contbr_occupation = fec.contbr_occupation.map(f)
```

In [190]:

```
emp_mapping = {
    'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
    'INFORMATION REQUESTED' : 'NOT PROVIDED',
    'SELF' : 'SELF-EMPLOYED',
    'SELF EMPLOYED' : 'SELF-EMPLOYED',
}
```

In [191]:

```
f = lambda x: emp_mapping.get(x, x)
fec.contbr_employer = fec.contbr_employer.map(f)
```

In [192]:

```
by_occupation = fec.pivot_table('contb_receipt_amt', index='contbr_occupation', columns='pa
```

In [193]:

```
over_2mm = by_occupation[by_occupation.sum(1) > 2000000]
```

In [194]:

```
over_2mm
```

Out[194]:

party	Democrat	Republican
contbr_occupation		
ATTORNEY	11141982.97	7.477194e+06
CEO	2074974.79	4.211041e+06
CONSULTANT	2459912.71	2.544725e+06
ENGINEER	951525.55	1.818374e+06
EXECUTIVE	1355161.05	4.138850e+06
...
PRESIDENT	1878509.95	4.720924e+06
PROFESSOR	2165071.08	2.967027e+05
REAL ESTATE	528902.09	1.625902e+06
RETIRED	25305116.38	2.356124e+07
SELF-EMPLOYED	672393.40	1.640253e+06

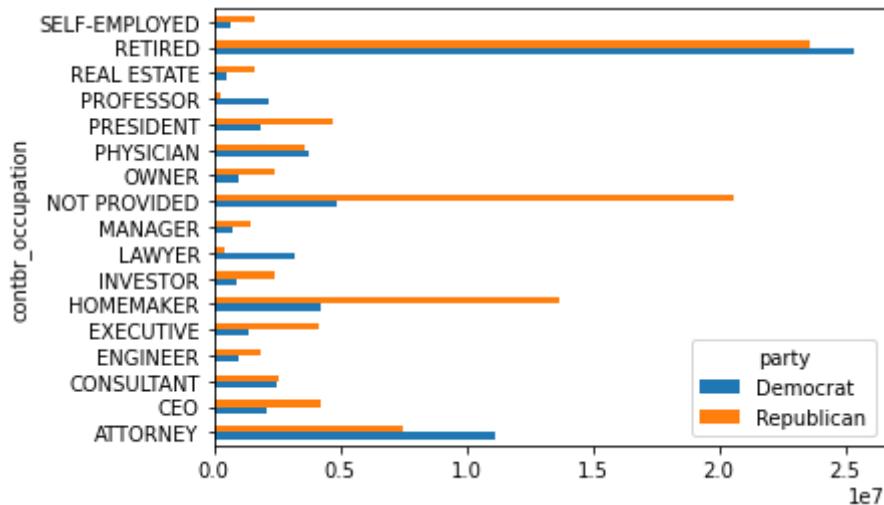
17 rows × 2 columns

In [195]:

```
over_2mm.plot(kind='barh')
```

Out[195]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c9ab332fc8>
```



In [196]:

```
def get_top_amounts(group, key, n=5):
    totals = group.groupby(key)[ 'contb_receipt_amt' ].sum()
    return totals.nlargest(n)
```

In [197]:

```
grouped = fec_mrbo.groupby( 'cand_nm' )
```

In [198]:

```
grouped.apply(get_top_amounts, 'contbr_occupation', n=7)
```

Out[198]:

cand_nm	contbr_occupation	contb_receipt_amt
Obama, Barack	RETIRED	25305116.38
	ATTORNEY	11141982.97
	INFORMATION REQUESTED	4866973.96
	HOMEMAKER	4248875.80
	PHYSICIAN	3735124.94
	...	
Romney, Mitt	HOMEMAKER	8147446.22
	ATTORNEY	5364718.82
	PRESIDENT	2491244.89
	EXECUTIVE	2300947.03
	C.E.O.	1968386.11

Name: contb_receipt_amt, Length: 14, dtype: float64

In [199]:

```
grouped.apply(get_top_amounts, 'contbr_employer', n=10)
```

Out[199]:

```
cand_nm      contbr_employer
Obama, Barack RETIRED           22694358.85
                  SELF-EMPLOYED    17080985.96
                  NOT EMPLOYED     8586308.70
                  INFORMATION REQUESTED 5053480.37
                  HOMEMAKER        2605408.54
                               ...
Romney, Mitt   CREDIT SUISSE      281150.00
                  MORGAN STANLEY    267266.00
                  GOLDMAN SACH & CO. 238250.00
                  BARCLAYS CAPITAL   162750.00
                  H.I.G. CAPITAL     139500.00
Name: contb_receipt_amt, Length: 20, dtype: float64
```

In [200]:

```
bins = np.array([0, 1, 10, 100, 1000, 10000, 100000, 1000000, 10000000])
```

In [201]:

```
labels = pd.cut(fec_mrbo.contb_receipt_amt, bins)
```

In [202]:

```
labels
```

Out[202]:

```
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] < (1000000, 10000000]
```

In [203]:

```
grouped = fec_mrbo.groupby(['cand_nm', labels])
```

In [204]:

```
grouped.size().unstack(0)
```

Out[204]:

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

In [205]:

```
bucket_sums = grouped.contb_receipt_amt.sum().unstack(0)
```

In [206]:

```
normed_sums = bucket_sums.div(bucket_sums.sum(axis=1), axis=0)
```

In [207]:

```
normed_sums
```

Out[207]:

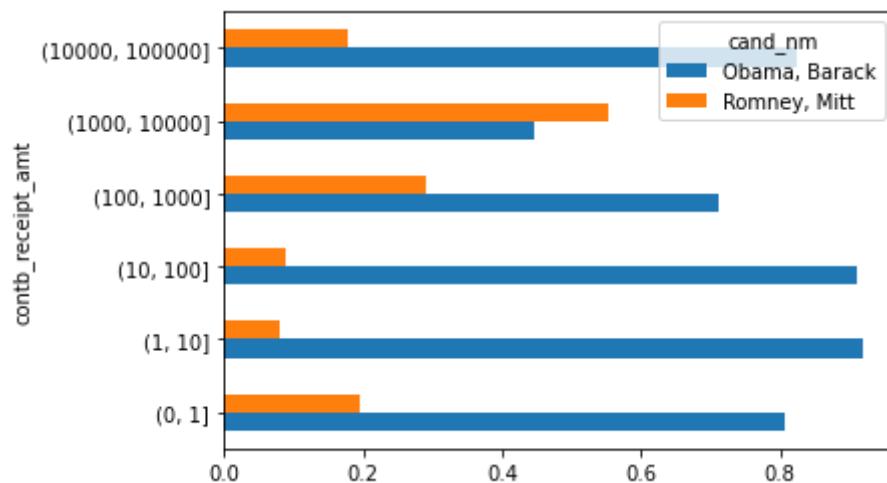
cand_nm	Obama, Barack	Romney, Mitt
contb_receipt_amt		
(0, 1]	0.805182	0.194818
(1, 10]	0.918767	0.081233
(10, 100]	0.910769	0.089231
(100, 1000]	0.710176	0.289824
(1000, 10000]	0.447326	0.552674
(10000, 100000]	0.823120	0.176880
(100000, 1000000]	1.000000	NaN
(1000000, 10000000]	1.000000	NaN

In [208]:

```
normed_sums[:-2].plot(kind='barh')
```

Out[208]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c9abe0b788>
```



In [209]:

```
grouped = fec_mrbo.groupby(['cand_nm', 'contbr_st'])
```

In [210]:

```
totals = grouped.contb_receipt_amt.sum().unstack(0).fillna(0)
```

In [211]:

```
totals = totals[totals.sum(1) > 100000]
```

In [212]:

```
totals[:10]
```

Out[212]:

```
cand_nm  Obama, Barack  Romney, Mitt
```

```
contbr_st
```

AK	281840.15	86204.24
AL	543123.48	527303.51
AR	359247.28	105556.00
AZ	1506476.98	1888436.23
CA	23824984.24	11237636.60
CO	2132429.49	1506714.12
CT	2068291.26	3499475.45
DC	4373538.80	1025137.50
DE	336669.14	82712.00
FL	7318178.58	8338458.81

In [213]:

```
percent = totals.div(totals.sum(1), axis=0)
```

In [214]:

```
percent[:10]
```

Out[214]:

```
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

A. ADVANCED NUMPY

A.1 ndarray Object Internals

In [215]:

```
np.ones((10, 5)).shape
```

Out[215]:

```
(10, 5)
```

In [216]:

```
np.ones((3, 4, 5), dtype=np.float64).strides
```

Out[216]:

```
(160, 40, 8)
```

In [217]:

```
ints = np.ones(10, dtype=np.uint16)
```

In [218]:

```
floats = np.ones(10, dtype=np.float32)
```

In [219]:

```
np.issubdtype(ints.dtype, np.integer)
```

Out[219]:

```
True
```

In [220]:

```
p.issubdtype(floats.dtype, np.floating)
```

Out[220]:

```
True
```

In [221]:

```
np.float64.mro()
```

Out[221]:

```
[numpy.float64,
 numpy.floating,
 numpy.inexact,
 numpy.number,
 numpy.generic,
 float,
 object]
```

In [222]:

```
np.issubdtype(ints.dtype, np.number)
```

Out[222]:

```
True
```

A.2 Advanced Array Manipulation

In [223]:

```
arr = np.arange(8)
```

In [224]:

```
arr
```

Out[224]:

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

In [225]:

```
arr.reshape((4, 2))
```

Out[225]:

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

In [226]:

```
arr.reshape((4, 2)).reshape((2, 4))
```

Out[226]:

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

In [227]:

```
arr = np.arange(15)
```

In [228]:

```
arr.reshape((5, -1))
```

Out[228]:

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

In [229]:

```
other_arr = np.ones((3, 5))
```

In [230]:

```
other_arr.shape
```

Out[230]:

```
(3, 5)
```

In [231]:

```
arr.reshape(other_arr.shape)
```

Out[231]:

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

In [232]:

```
arr = np.arange(15).reshape((5, 3))
```

In [233]:

```
arr
```

Out[233]:

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

In [234]:

```
arr.ravel()
```

Out[234]:

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

In [235]:

```
arr.flatten()
```

Out[235]:

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

In [236]:

```
arr = np.arange(12).reshape((3, 4))
```

In [237]:

```
arr
```

Out[237]:

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

In [238]:

```
arr.ravel()
```

Out[238]:

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

In [239]:

```
arr.ravel('F')
```

Out[239]:

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

In [240]:

```
arr1 = np.array([[1, 2, 3], [4, 5, 6]])
```

In [241]:

```
arr2 = np.array([[7, 8, 9], [10, 11, 12]])
```

In [242]:

```
np.concatenate([arr1, arr2], axis=0)
```

Out[242]:

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

In [243]:

```
np.concatenate([arr1, arr2], axis=1)
```

Out[243]:

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

In [244]:

```
np.vstack((arr1, arr2))
```

Out[244]:

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

In [245]:

```
np.hstack((arr1, arr2))
```

Out[245]:

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

In [246]:

```
arr = np.random.randn(5, 2)
```

In [247]:

```
arr
```

Out[247]:

```
array([[ 3.06697775, -1.61717946],  
       [ 0.58352709,  0.90912396],  
       [-0.01475339, -0.2022391 ],  
       [-0.20794753, -0.70554815],  
       [-1.57742344,  0.57600379]])
```

In [248]:

```
first, second, third = np.split(arr, [1, 3])
```

In [249]:

```
first
```

Out[249]:

```
array([[ 3.06697775, -1.61717946]])
```

In [250]:

```
second
```

Out[250]:

```
array([[ 0.58352709,  0.90912396],  
       [-0.01475339, -0.2022391 ]])
```

In [251]:

```
third
```

Out[251]:

```
array([[-0.20794753, -0.70554815],  
       [-1.57742344,  0.57600379]])
```

In [252]:

```
arr = np.arange(6)
```

In [253]:

```
arr1 = arr.reshape((3, 2))
```

In [254]:

```
arr2 = np.random.randn(3, 2)
```

In [255]:

```
np.r_[arr1, arr2]
```

Out[255]:

```
array([[ 0.        ,  1.        ],
       [ 2.        ,  3.        ],
       [ 4.        ,  5.        ],
       [ 0.59365797,  0.97563235],
       [ 0.06593813, -0.52650573],
       [ 0.2592866 , -0.4780047 ]])
```

In [256]:

```
np.c_[np.r_[arr1, arr2], arr]
```

Out[256]:

```
array([[ 0.        ,  1.        ,  0.        ],
       [ 2.        ,  3.        ,  1.        ],
       [ 4.        ,  5.        ,  2.        ],
       [ 0.59365797,  0.97563235,  3.        ],
       [ 0.06593813, -0.52650573,  4.        ],
       [ 0.2592866 , -0.4780047 ,  5.        ]])
```

In [257]:

```
np.c_[1:6, -10:-5]
```

Out[257]:

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

In [258]:

```
arr = np.arange(3)
```

In [259]:

```
arr
```

Out[259]:

```
array([0, 1, 2])
```

In [260]:

```
arr.repeat(3)
```

Out[260]:

```
array([0, 0, 0, 1, 1, 1, 2, 2, 2])
```

In [261]:

```
arr.repeat([2, 3, 4])
```

Out[261]:

```
array([0, 0, 1, 1, 1, 2, 2, 2, 2])
```

In [262]:

```
arr = np.random.randn(2, 2)
```

In [263]:

```
arr
```

Out[263]:

```
array([[ 0.4883005 ,  1.31331934],
       [ 3.03693874, -0.74344171]])
```

In [264]:

```
arr.repeat(2, axis=0)
```

Out[264]:

```
array([[ 0.4883005 ,  1.31331934],
       [ 0.4883005 ,  1.31331934],
       [ 3.03693874, -0.74344171],
       [ 3.03693874, -0.74344171]])
```

In [265]:

```
arr.repeat([2, 3], axis=0)
```

Out[265]:

```
array([[ 0.4883005 ,  1.31331934],
       [ 0.4883005 ,  1.31331934],
       [ 3.03693874, -0.74344171],
       [ 3.03693874, -0.74344171],
       [ 3.03693874, -0.74344171]])
```

In [266]:

```
arr.repeat([2, 3], axis=1)
```

Out[266]:

```
array([[ 0.4883005 ,  0.4883005 ,  1.31331934,  1.31331934,  1.31331934],
       [ 3.03693874,  3.03693874, -0.74344171, -0.74344171, -0.74344171]])
```

In [267]:

```
arr
```

Out[267]:

```
array([[ 0.4883005 ,  1.31331934],  
       [ 3.03693874, -0.74344171]])
```

In [268]:

```
np.tile(arr, 2)
```

Out[268]:

```
array([[ 0.4883005 ,  1.31331934,  0.4883005 ,  1.31331934],  
       [ 3.03693874, -0.74344171,  3.03693874, -0.74344171]])
```

In [269]:

```
arr
```

Out[269]:

```
array([[ 0.4883005 ,  1.31331934],  
       [ 3.03693874, -0.74344171]])
```

In [270]:

```
np.tile(arr, (2, 1))
```

Out[270]:

```
array([[ 0.4883005 ,  1.31331934],  
       [ 3.03693874, -0.74344171],  
       [ 0.4883005 ,  1.31331934],  
       [ 3.03693874, -0.74344171]])
```

In [271]:

```
np.tile(arr, (3, 2))
```

Out[271]:

```
array([[ 0.4883005 ,  1.31331934,  0.4883005 ,  1.31331934],  
       [ 3.03693874, -0.74344171,  3.03693874, -0.74344171],  
       [ 0.4883005 ,  1.31331934,  0.4883005 ,  1.31331934],  
       [ 3.03693874, -0.74344171,  3.03693874, -0.74344171],  
       [ 0.4883005 ,  1.31331934,  0.4883005 ,  1.31331934],  
       [ 3.03693874, -0.74344171,  3.03693874, -0.74344171]])
```

In [272]:

```
arr = np.arange(10) * 100
```

In [273]:

```
inds = [7, 1, 2, 6]
```

In [274]:

```
arr[inds]
```

Out[274]:

```
array([700, 100, 200, 600])
```

In [275]:

```
arr.take(inds)
```

Out[275]:

```
array([700, 100, 200, 600])
```

In [276]:

```
arr.put(inds, 42)
```

In [277]:

```
arr
```

Out[277]:

```
array([ 0, 42, 42, 300, 400, 500, 42, 42, 800, 900])
```

In [278]:

```
arr.put(inds, [40, 41, 42, 43])
```

In [279]:

```
arr
```

Out[279]:

```
array([ 0, 41, 42, 300, 400, 500, 43, 40, 800, 900])
```

In [280]:

```
inds = [2, 0, 2, 1]
```

In [281]:

```
arr = np.random.randn(2, 4)
```

In [282]:

```
arr
```

Out[282]:

```
array([[ 0.8988355 ,  0.6009292 ,  1.33750498, -1.14937752],
       [ 0.7972014 ,  2.37922409, -0.2925739 ,  0.52161322]])
```

In [283]:

```
arr.take(indices, axis=1)
```

Out[283]:

```
array([[ 1.33750498,  0.8988355 ,  1.33750498,  0.6009292 ],
       [-0.2925739 ,  0.7972014 , -0.2925739 ,  2.37922409]])
```

A.3 Broadcasting

In [284]:

```
arr = np.arange(5)
```

In [285]:

```
arr
```

Out[285]:

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

In [286]:

```
arr * 4
```

Out[286]:

```
array([ 0,  4,  8, 12, 16])
```

In [287]:

```
arr = np.random.randn(4, 3)
```

In [288]:

```
arr.mean(0)
```

Out[288]:

```
array([-0.20632143,  0.32105222,  0.58076385])
```

In [289]:

```
demeaned = arr - arr.mean(0)
```

In [290]:

```
demeaned
```

Out[290]:

```
array([[ 1.7397372 , -0.76944682,  0.81904024],
       [ 0.64250075, -0.47805927,  0.13569024],
       [-1.77998428, -1.01359319, -0.50057946],
       [-0.60225367,  2.26109928, -0.45415101]])
```

In [291]:

```
demeaned.mean(0)
```

Out[291]:

```
array([-5.55111512e-17,  0.00000000e+00, -4.16333634e-17])
```

In [292]:

```
arr
```

Out[292]:

```
array([[ 1.53341577, -0.4483946 ,  1.39980408],
       [ 0.43617933, -0.15700706,  0.71645408],
       [-1.9863057 , -0.69254097,  0.08018439],
       [-0.8085751 ,  2.5821515 ,  0.12661283]])
```

In [293]:

```
row_means = arr.mean(1)
```

In [294]:

```
row_means.shape
```

Out[294]:

```
(4,)
```

In [296]:

```
row_means.reshape((4, 1))
```

Out[296]:

```
array([[ 0.82827508],
       [ 0.33187545],
       [-0.86622076],
       [ 0.63339641]])
```

In [297]:

```
demeaned = arr - row_means.reshape((4, 1))
```

In [298]:

```
demeaned.mean(1)
```

Out[298]:

```
array([-7.40148683e-17, -1.85037171e-17, -7.40148683e-17,  7.40148683e-17])
```

In [299]:

```
arr = arr.mean(1)
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-299-8b8ada26fac0> in <module>  
----> 1 arr = arr.mean(1)
```

ValueError: operands could not be broadcast together with shapes (4,3) (4,)

In [300]:

```
arr = arr.mean(1).reshape((4, 1))
```

Out[300]:

```
array([[ 0.70514069, -1.27666969,  0.571529 ],
       [ 0.10430388, -0.48888251,  0.38457863],
       [-1.12008494,  0.17367979,  0.94640515],
       [-1.44197151,  1.94875509, -0.50678358]])
```

In [301]:

```
arr = np.zeros((4, 4))
```

In [302]:

```
arr_3d = arr[:, np.newaxis, :]
```

In [303]:

```
arr_3d.shape
```

Out[303]:

```
(4, 1, 4)
```

In [304]:

```
arr_1d = np.random.normal(size=3)
```

In [305]:

```
arr_1d[:, np.newaxis]
```

Out[305]:

```
array([[0.51057819],
       [2.13160623],
       [1.73162171]])
```

In [306]:

```
arr_1d[np.newaxis, :]
```

Out[306]:

```
array([[0.51057819, 2.13160623, 1.73162171]])
```

In [307]:

```
arr = np.random.randn(3, 4, 5)
```

In [308]:

```
depth_means = arr.mean(2)
```

In [309]:

```
depth_means
```

Out[309]:

```
array([[ 0.19165733,  0.24342376,  0.8066721 , -0.81734736],
       [ 0.39677529,  0.26711305, -0.03044733, -0.45986645],
       [-0.56064883, -0.76490699, -0.26239358,  0.16140352]])
```

In [310]:

```
depth_means.shape
```

Out[310]:

```
(3, 4)
```

In [311]:

```
demeaned = arr - depth_means[:, :, np.newaxis]
```

In [312]:

```
demeaned.mean(2)
```

Out[312]:

```
array([[-3.33066907e-17, -2.22044605e-17, -1.33226763e-16,
       8.88178420e-17],
      [-8.88178420e-17,  0.00000000e+00,  6.66133815e-17,
       -8.88178420e-17],
      [-4.44089210e-17,  1.11022302e-16, -1.66533454e-17,
       -1.11022302e-17]])
```

In [316]:

```
def demean_axis(arr, axis=0):
    means = arr.mean(axis)
```

```
indexer = [slice(None)] * arr.ndim
indexer[axis] = np.newaxis
return arr - means[indexer]
```

In [318]:

```
arr = np.zeros((4, 3))
```

In [319]:

```
arr[:] = 5
```

In [320]:

```
arr
```

Out[320]:

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

In [321]:

```
col = np.array([1.28, -0.42, 0.44, 1.6])
```

In [324]:

```
arr[:] = col[:, np.newaxis]
```

In [325]:

```
arr
```

Out[325]:

```
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 [326]:

```
arr[:2] = [[-1.37], [0.509]]
```

In [327]:

```
arr
```

Out[327]:

```
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 Advanced ufunc Usage

In [329]:

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

In [330]:

```
np.add.reduce(arr)
```

Out[330]:

45

In [331]:

```
arr.sum()
```

Out[331]:

45

In [332]:

```
np.random.seed(12346)
```

In [333]:

```
arr = np.random.randn(5, 5)
```

In [334]:

```
arr[::-2].sort(1)
```

In [335]:

```
arr[:, :-1] < arr[:, 1:]
```

Out[335]:

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

In [337]:

```
np.logical_and.reduce(arr[:, :-1] < arr[:, 1:], axis=1)
```

Out[337]:

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

In [338]:

```
arr = np.arange(15).reshape((3, 5))
```

In [339]:

```
np.add.accumulate(arr, axis=1)
```

Out[339]:

```
array([[ 0,  1,  3,  6, 10],
       [ 5, 11, 18, 26, 35],
       [10, 21, 33, 46, 60]], dtype=int32)
```

In [340]:

```
arr = np.arange(3).repeat([1, 2, 2])
```

In [341]:

```
arr
```

Out[341]:

```
array([0, 1, 1, 2, 2])
```

In [342]:

```
np.multiply.outer(arr, np.arange(5))
```

Out[342]:

```
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]])
```

In [343]:

```
x, y = np.random.randn(3, 4), np.random.randn(5)
```

In [344]:

```
result = np.subtract.outer(x, y)
```

In [345]:

```
result.shape
```

Out[345]:

```
(3, 4, 5)
```

In [346]:

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

In [347]:

```
np.add.reduceat(arr, [0, 5, 8])
```

Out[347]:

```
array([10, 18, 17], dtype=int32)
```

In [348]:

```
arr = np.multiply.outer(np.arange(4), np.arange(5))
```

In [349]:

```
arr
```

Out[349]:

```
array([[ 0,  0,  0,  0,  0],
       [ 0,  1,  2,  3,  4],
       [ 0,  2,  4,  6,  8],
       [ 0,  3,  6,  9, 12]])
```

In [350]:

```
np.add.reduceat(arr, [0, 2, 4], axis=1)
```

Out[350]:

```
array([[ 0,  0,  0],
       [ 1,  5,  4],
       [ 2, 10,  8],
       [ 3, 15, 12]]], dtype=int32)
```

In [351]:

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

In [352]:

```
addThem = np.frompyfunc(add_elements, 2, 1)
```

In [353]:

```
addThem(np.arange(8), np.arange(8))
```

Out[353]:

```
array([0, 2, 4, 6, 8, 10, 12, 14], dtype=object)
```

In [354]:

```
addThem = np.vectorize(add_elements, otypes=[np.float64])
```

In [355]:

```
addThem(np.arange(8), np.arange(8))
```

Out[355]:

```
array([ 0.,  2.,  4.,  6.,  8., 10., 12., 14.])
```

In [356]:

```
arr = np.random.randn(10000)
```

In [357]:

```
%timeit addThem(arr, arr)
```

```
4.63 ms ± 712 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [358]:

```
%timeit np.add(arr, arr)
```

```
11.5 µs ± 6.64 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

A.5 Structured and Record Arrays

In [359]:

```
dtype = [('x', np.float64), ('y', np.int32)]
```

In [360]:

```
sarr = np.array([(1.5, 6), (np.pi, -2)], dtype=dtype)
```

In [361]:

```
sarr
```

Out[361]:

```
array([(1.5, 6), (3.14159265, -2)],  
      dtype=[('x', '<f8'), ('y', '<i4')])
```

In [362]:

```
sarr[0]
```

Out[362]:

```
(1.5, 6)
```

In [363]:

```
sarr[0]['y']
```

Out[363]:

```
6
```

In [364]:

```
sarr['x']
```

Out[364]:

```
array([1.5, 3.14159265])
```

In [365]:

```
dtype = [('x', np.int64, 3), ('y', np.int32)]
```

In [366]:

```
arr = np.zeros(4, dtype=dtype)
```

In [367]:

```
arr
```

Out[367]:

```
array([[0, 0, 0], 0), ([0, 0, 0], 0), ([0, 0, 0], 0), ([0, 0, 0], 0)],  
      dtype=[('x', '<i8', (3,)), ('y', '<i4'))])
```

In [368]:

```
arr[0]['x']
```

Out[368]:

```
array([0, 0, 0], dtype=int64)
```

In [369]:

```
arr['x']
```

Out[369]:

```
array([[0, 0, 0],  
       [0, 0, 0],  
       [0, 0, 0],  
       [0, 0, 0]], dtype=int64)
```

In [370]:

```
dtype = [('x', [('a', 'f8'), ('b', 'f4')]), ('y', np.int32)]
```

In [371]:

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

In [372]:

```
data['x']
```

Out[372]:

```
array([(1., 2.), (3., 4.)], dtype=[('a', '<f8'), ('b', '<f4')])
```

In [373]:

```
data['y']
```

Out[373]:

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

In [374]:

```
data['x']['a']
```

Out[374]:

```
array([1., 3.])
```

A.6 More About Sorting

In [375]:

```
arr = np.random.randn(6)
```

In [376]:

```
arr.sort()
```

In [377]:

```
arr
```

Out[377]:

```
array([-1.08199644,  0.37588273,  0.80139193,  1.13969136,  1.28881614,
       1.84126094])
```

In [378]:

```
arr = np.random.randn(3, 5)
```

In [379]:

```
arr
```

Out[379]:

```
array([[-0.33176812, -1.47108206,  0.87050269, -0.08468875, -1.13286962],
       [-1.01114869, -0.34357617,  2.17140268,  0.12337075, -0.01893118],
       [ 0.17731791,  0.7423957 ,  0.85475634,  1.03797268, -0.32899594]])
```

In [380]:

```
arr[:, 0].sort()
```

In [381]:

```
arr
```

Out[381]:

```
array([[-1.01114869, -1.47108206,  0.87050269, -0.08468875, -1.13286962],
       [-0.33176812, -0.34357617,  2.17140268,  0.12337075, -0.01893118],
       [ 0.17731791,  0.7423957 ,  0.85475634,  1.03797268, -0.32899594]])
```

In [382]:

```
arr = np.random.randn(5)
```

In [383]:

```
arr
```

Out[383]:

```
array([-1.11807759, -0.24152521, -2.0051193 ,  0.73788753, -1.06137462])
```

In [384]:

```
np.sort(arr)
```

Out[384]:

```
array([-2.0051193 , -1.11807759, -1.06137462, -0.24152521,  0.73788753])
```

In [385]:

```
arr
```

Out[385]:

```
array([-1.11807759, -0.24152521, -2.0051193 ,  0.73788753, -1.06137462])
```

In [386]:

```
arr = np.random.randn(3, 5)
```

In [387]:

```
arr
```

Out[387]:

```
array([[ 0.59545348, -0.26822958,  1.33885804, -0.18715572,  0.91108374],
       [-0.32150045,  1.00543901, -0.51683937,  1.19251887, -0.19893404],
       [ 0.39691349, -1.76381537,  0.60709023, -0.22215536, -0.21707838]])
```

In [388]:

```
arr.sort(axis=1)
```

In [389]:

```
arr
```

Out[389]:

```
array([[-0.26822958, -0.18715572,  0.59545348,  0.91108374,  1.33885804],
       [-0.51683937, -0.32150045, -0.19893404,  1.00543901,  1.19251887],
       [-1.76381537, -0.22215536, -0.21707838,  0.39691349,  0.60709023]])
```

In [390]:

```
arr[:, ::-1]
```

Out[390]:

```
array([[ 1.33885804,  0.91108374,  0.59545348, -0.18715572, -0.26822958],
       [ 1.19251887,  1.00543901, -0.19893404, -0.32150045, -0.51683937],
       [ 0.60709023,  0.39691349, -0.21707838, -0.22215536, -1.76381537]])
```

In [393]:

```
values = np.array([5, 0, 1, 3, 2])
```

In [394]:

```
indexer = values.argsort()
```

In [395]:

```
indexer
```

Out[395]:

```
array([1, 2, 4, 3, 0], dtype=int64)
```

In [396]:

```
values[indexer]
```

Out[396]:

```
array([0, 1, 2, 3, 5])
```

In [397]:

```
arr = np.random.randn(3, 5)
```

In [398]:

```
arr[0] = values
```

In [399]:

```
arr
```

Out[399]:

```
array([[ 5.          ,  0.          ,  1.          ,  3.          ,  2.          ],
       [-0.36360302, -0.13775933,  2.17773731, -0.47280687,  0.8356152 ],
       [-0.20885016,  0.23159352,  0.72798172, -1.3918432 ,  1.99558262]])
```

In [400]:

```
arr[:, arr[0].argsort()]
```

Out[400]:

```
array([[ 0.          ,  1.          ,  2.          ,  3.          ,  5.          ],
       [-0.13775933,  2.17773731,  0.8356152 , -0.47280687, -0.36360302],
       [ 0.23159352,  0.72798172,  1.99558262, -1.3918432 , -0.20885016]])
```

In [401]:

```
first_name = np.array(['Bob', 'Jane', 'Steve', 'Bill', 'Barbara'])
```

In [402]:

```
last_name = np.array(['Jones', 'Arnold', 'Arnold', 'Jones', 'Walters'])
```

In [403]:

```
sorter = np.lexsort((first_name, last_name))
```

In [404]:

```
sorter
```

Out[404]:

```
array([1, 2, 3, 0, 4], dtype=int64)
```

In [405]:

```
zip(last_name[sorter], first_name[sorter])
```

Out[405]:

```
<zip at 0x1c99621d9c8>
```

In [407]:

```
values = np.array(['2:first', '2:second', '1:first', '1:second', '1:third'])
```

In [408]:

```
key = np.array([2, 2, 1, 1, 1])
```

In [409]:

```
indexer = key.argsort(kind='mergesort')
```

In [410]:

```
indexer
```

Out[410]:

```
array([2, 3, 4, 0, 1], dtype=int64)
```

In [411]:

```
values.take(indexer)
```

Out[411]:

```
array(['1:first', '1:second', '1:third', '2:first', '2:second'],
      dtype='<U8')
```

In [412]:

```
np.random.seed(12345)
```

In [413]:

```
arr = np.random.randn(20)
```

In [414]:

```
arr
```

Out[414]:

```
array([-0.20470766,  0.47894334, -0.51943872, -0.5557303 ,  1.96578057,
       1.39340583,  0.09290788,  0.28174615,  0.76902257,  1.24643474,
      1.00718936, -1.29622111,  0.27499163,  0.22891288,  1.35291684,
     0.88642934, -2.00163731, -0.37184254,  1.66902531, -0.43856974])
```

In [415]:

```
np.partition(arr, 3)
```

Out[415]:

```
array([-2.00163731, -1.29622111, -0.5557303 , -0.51943872, -0.37184254,
       -0.43856974, -0.20470766,  0.28174615,  0.76902257,  0.47894334,
      1.00718936,  0.09290788,  0.27499163,  0.22891288,  1.35291684,
     0.88642934,  1.39340583,  1.96578057,  1.66902531,  1.24643474])
```

In [416]:

```
indices = np.argpartition(arr, 3)
```

In [417]:

```
indices
```

Out[417]:

```
array([16, 11,  3,  2, 17, 19,  0,   7,   8,   1,  10,   6,  12, 13, 14, 15,   5,
       4, 18,   9], dtype=int64)
```

In [418]:

```
arr.take(indices)
```

Out[418]:

```
array([-2.00163731, -1.29622111, -0.5557303 , -0.51943872, -0.37184254,
       -0.43856974, -0.20470766,  0.28174615,  0.76902257,  0.47894334,
      1.00718936,  0.09290788,  0.27499163,  0.22891288,  1.35291684,
     0.88642934,  1.39340583,  1.96578057,  1.66902531,  1.24643474])
```

In [419]:

```
arr = np.array([0, 1, 7, 12, 15])
```

In [420]:

```
arr.searchsorted(9)
```

Out[420]:

3

In [421]:

```
arr.searchsorted([0, 8, 11, 16])
```

Out[421]:

```
array([0, 3, 3, 5], dtype=int64)
```

In [422]:

```
arr = np.array([0, 0, 0, 1, 1, 1, 1])
```

In [423]:

```
arr.searchsorted([0, 1])
```

Out[423]:

```
array([0, 3], dtype=int64)
```

In [424]:

```
arr.searchsorted([0, 1], side='right')
```

Out[424]:

```
array([3, 7], dtype=int64)
```

In [425]:

```
data = np.floor(np.random.uniform(0, 10000, size=50))
```

In [426]:

```
bins = np.array([0, 100, 1000, 5000, 10000])
```

In [427]:

```
data
```

Out[427]:

```
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.])
```

In [428]:

```
labels = bins.searchsorted(data)
```

In [429]:

```
labels
```

Out[429]:

```
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, 3, 4, 4, 3, 3, 3, 4, 3, 3, 4, 4, 4, 4, 4, 4,
       4, 3, 3, 4, 4, 3], dtype=int64)
```

In [430]:

```
pd.Series(data).groupby(labels).mean()
```

Out[430]:

```
2    498.000000
3   3064.277778
4  7389.035714
dtype: float64
```

A.7 Writing Fast NumPy Functions with Numba

In [431]:

```
import numpy as np
```

In [432]:

```
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
```

In [433]:

```
x = np.random.randn(10000000)
```

In [434]:

```
y = np.random.randn(10000000)
```

In [435]:

```
%timeit mean_distance(x, y)
```

```
11.3 s ± 3.26 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [436]:

```
%timeit (x - y).mean()
```

```
78.4 ms ± 1.73 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [444]:

```
import numba as nb
```

In [445]:

```
numba_mean_distance = nb.jit(mean_distance)
```

...

In [442]:

```
@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
```

In [443]:

```
%timeit numba_mean_distance(x, y)
```

...

In [446]:

```
from numba import float64, njit
```

In [447]:

```
@njit(float64(float64[:, :], float64[:, :]))
def mean_distance(x, y):
    return (x - y).mean()
```

In [448]:

```
from numba import vectorize
```

In [449]:

```
@vectorize
def nb_add(x, y):
    return x + y
```

In [450]:

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

In [451]:

```
nb_add(x, x)
```

Out[451]:

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18], dtype=int64)
```

A.8 Advanced Array Input and Output

In [453]:

```
mmap = np.memmap('my mmap', dtype='float64', mode='w+', shape=(10000, 10000))
```

In [454]:

```
mmap
```

Out[454]:

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

In [455]:

```
section = mmap[:5]
```

In [456]:

```
section[:] = np.random.randn(5, 10000)
```

In [457]:

```
mmap.flush()
```

In [458]:

```
mmap
```

Out[458]:

```
memmap([[ 1.37140985,  0.93127837,  0.60573747, ... , -0.62115557,
          -0.46780136,  0.47874865],
        [ 0.42296545,  0.83060431,  0.69976547, ... ,  1.28831447,
          0.58858679, -1.42755372],
        [ 2.16005954, -1.24616489,  2.44470054, ... ,  0.86866129,
          0.28019716,  2.13008671],
        ... ,
        [ 0.          ,  0.          ,  0.          , ... ,  0.          ,
          0.          ,  0.          ],
        [ 0.          ,  0.          ,  0.          , ... ,  0.          ,
          0.          ,  0.          ],
        [ 0.          ,  0.          ,  0.          , ... ,  0.          ,
          0.          ,  0.          ]])
```

In [459]:

```
del mmap
```

In [460]:

```
mmap = np.memmap('my mmap', dtype='float64', shape=(10000, 10000))
```

In [461]:

```
mmap
```

Out[461]:

```
memmap([[ 1.37140985,  0.93127837,  0.60573747, ..., -0.62115557,
          -0.46780136,  0.47874865],
         [ 0.42296545,  0.83060431,  0.69976547, ...,  1.28831447,
          0.58858679, -1.42755372],
         [ 2.16005954, -1.24616489,  2.44470054, ...,  0.86866129,
          0.28019716,  2.13008671],
         ...,
         [ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.          ],
         [ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.          ],
         [ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.          ],
         [ 0.          ,  0.          ,  0.          , ...,  0.          ,
          0.          ,  0.        ]])
```

A.9 Performance Tips

In [462]:

```
arr_c = np.ones((1000, 1000), order='C')
```

In [463]:

```
arr_f = np.ones((1000, 1000), order='F')
```

In [464]:

```
arr_c.flags
```

Out[464]:

```
C_CONTIGUOUS : True
F_CONTIGUOUS : False
OWNDATA : True
WRITEABLE : True
ALIGNED : True
WRITEBACKIFCOPY : False
UPDATEIFCOPY : False
```

In [465]:

```
arr_f.flags
```

Out[465]:

```
C_CONTIGUOUS : False
F_CONTIGUOUS : True
OWNDATA : True
WRITEABLE : True
ALIGNED : True
WRITEBACKIFCOPY : False
UPDATEIFCOPY : False
```

In [466]:

```
arr_f.flags.f_contiguous
```

Out[466]:

True

In [467]:

```
%timeit arr_c.sum(1)
```

1.56 ms ± 169 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

In [468]:

```
%timeit arr_f.sum(1)
```

1.15 ms ± 120 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

In [469]:

```
arr_f.copy('C').flags
```

Out[469]:

```
C_CONTIGUOUS : True
F_CONTIGUOUS : False
OWNDATA : True
WRITEABLE : True
ALIGNED : True
WRITEBACKIFCOPY : False
UPDATEIFCOPY : False
```

In [470]:

```
arr_c[:50].flags.contiguous
```

Out[470]:

True

In [471]:

```
arr_c[:, :50].flags
```

Out[471]:

```
C_CONTIGUOUS : False
F_CONTIGUOUS : False
OWNDATA : False
WRITEABLE : True
ALIGNED : True
WRITEBACKIFCOPY : False
UPDATEIFCOPY : False
```

B. MORE ON THE IPYTHON SYSTEM

B.1 Using the Command History

In [1]:

```
2 ** 27
```

Out[1]:

```
134217728
```

In [2]:

```
-
```

Out[2]:

```
134217728
```

In [3]:

```
foo = 'bar'
```

In [4]:

```
foo
```

Out[4]:

```
'bar'
```

Şu şekilde de kullanılabilir:

```
_i27  
_27  
exec(_i27)
```

B.2 Interacting with the Operating System

In [5]:

```
ip_info = !ifconfig wlan0 | grep "inet "
```

In [6]:

```
ip_info[0].strip()
```

Out[6]:

```
"'ifconfig' is not recognized as an internal or external command,"
```

In [7]:

```
foo = 'test*'
```

In [8]:

```
!ls $foo
```

```
...
```

In [9]:

```
%alias ll ls -l
```

In [10]:

```
ll /usr
```

...

In [11]:

```
%alias test_alias (cd examples; ls; cd ..)
```

In [12]:

```
test_alias
```

...

In [13]:

```
%bookmark py4da /home/wesm/code/pydata-book
```

In [14]:

```
cd py4da
```

```
(bookmark:py4da) -> /home/wesm/code/pydata-book  
[WinError 3] Sistem belirtilen yolu bulamıyor: '/home/wesm/code/pydata-book'  
C:\Users\Kader\Desktop
```

In [15]:

```
%bookmark -l
```

Current bookmarks:

```
py4da -> /home/wesm/code/pydata-book
```

B.3 Software Development Tools

In [16]:

```
run examples/ipython_bug.py
```

```
-----
AssertionError                                                 Traceback (most recent call last)
~\Desktop\examples\ipython_bug.py in <module>
    13     throws_an_exception()
    14
--> 15 calling_things()

~\Desktop\examples\ipython_bug.py in calling_things()
    11 def calling_things():
    12     works_fine()
--> 13     throws_an_exception()
    14
    15 calling_things()

~\Desktop\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 []:

```
%debug
```

```
> c:\users\kader\desktop\examples\ipython_bug.py(9)throws_an_exception()
    7     a = 5
    8     b = 6
--> 9     assert(a + b == 10)
   10
   11 def calling_things():

ipdb> u
> c:\users\kader\desktop\examples\ipython_bug.py(13)calling_things()
    11 def calling_things():
    12     works_fine()
--> 13     throws_an_exception()
    14
    15 calling_things()
```

In []:

```
run -d examples/ipython_bug.py
```

```
Breakpoint 1 at c:\users\kader\desktop\examples\ipython_bug.py:1
```

```
NOTE: Enter 'c' at the ipdb> prompt to continue execution.
```

```
> c:\users\kader\desktop\examples\ipython_bug.py(1)<module>()
```

```
1---> 1 def works_fine():
    2     a = 5
    3     b = 6
    4     assert(a + b == 11)
    5
```

```
ipdb> s
```

```
> c:\users\kader\desktop\examples\ipython_bug.py(6)<module>()
```

```
    4     assert(a + b == 11)
    5
```

```
----> 6 def throws_an_exception():
    7     a = 5
    8     b = 6
```

```
ipdb> b 12
```

```
Breakpoint 2 at c:\users\kader\desktop\examples\ipython_bug.py:12
```

```
ipdb> c
```

```
> c:\users\kader\desktop\examples\ipython_bug.py(12)calling_things()
```

```
    10
    11 def calling_things():
2---> 12     works_fine()
    13     throws_an_exception()
    14
```

```
ipdb> n
```

```
> c:\users\kader\desktop\examples\ipython_bug.py(13)calling_things()
```

```
    11 def calling_things():
2    12     works_fine()
---> 13     throws_an_exception()
    14
    15 calling_things()
```

```
ipdb> s
```

```
--Call--
```

```
> c:\users\kader\desktop\examples\ipython_bug.py(6)throws_an_exception()
```

```
    4     assert(a + b == 11)
    5
----> 6 def throws_an_exception():
    7     a = 5
    8     b = 6
```

```
ipdb> n
```

```
> c:\users\kader\desktop\examples\ipython_bug.py(7)throws_an_exception()
```

```
    5
    6 def throws_an_exception():
----> 7     a = 5
    8     b = 6
    9     assert(a + b == 10)
```

```
ipdb> n
```

```
> c:\users\kader\desktop\examples\ipython_bug.py(8)throws_an_exception()
```

```
    6 def throws_an_exception():
    7     a = 5
----> 8     b = 6
```

```
9     assert(a + b == 10)
10

ipdb> n
> c:\users\kader\desktop\examples\ipython_bug.py(9) throws_an_exception()
    7     a = 5
    8     b = 6
----> 9     assert(a + b == 10)
   10
11 def calling_things():

ipdb> !a
5
ipdb> !b
6
```

In [1]:

```
from IPython.core.debugger import Pdb
```

In [2]:

```
def set_trace():
    Pdb(color_scheme='Linux').set_trace(sys._getframe().f_back)
```

In [3]:

```
def debug(f, *args, **kwargs):
    pdb = Pdb(color_scheme='Linux')
    return pdb.runcall(f, *args, **kwargs)
```

In [4]:

```
def f(x, y, z=1):
    tmp = x + y
    return tmp / z
```

In []:

```
debug(f, 1, 2, z=3)
```

```
C:\Users\Kader\anaconda3\lib\site-packages\ipykernel_launcher.py:2: DeprecationWarning: The `color_scheme` argument is deprecated since version 5.1
```

```
> <ipython-input-4-359ec13d6433>(2)f()
    1 def f(x, y, z=1):
----> 2     tmp = x + y
      3     return tmp / z
```

In []:

```
%run -d examples/ipython_bug.py
```

```
Breakpoint 1 at c:\users\kader\desktop\examples\ipython_bug.py:1
NOTE: Enter 'c' at the ipdb> prompt to continue execution.
> c:\users\kader\desktop\examples\ipython_bug.py(1)<module>()
1---> 1 def works_fine():
      2     a = 5
      3     b = 6
      4     assert(a + b == 11)
      5
```

In []:

```
%run -d -b2 examples/ipython_bug.py
```

```
Breakpoint 1 at c:\users\kader\desktop\examples\ipython_bug.py:2
NOTE: Enter 'c' at the ipdb> prompt to continue execution.
> c:\users\kader\desktop\examples\ipython_bug.py(1)<module>()
----> 1 def works_fine():
      2     a = 5
      3     b = 6
      4     assert(a + b == 11)
      5
```

```
ipdb> c
> c:\users\kader\desktop\examples\ipython_bug.py(2)works_fine()
  1 def works_fine():
----> 2     a = 5
      3     b = 6
      4     assert(a + b == 11)
      5
```

Aşağıdaki ifadeyi bir defa çalıştırarak tekrarlardan kurtuluyoruz:

```
import time
start = time.time()
for i in range(iterations):
    elapsed_per = (time.time() - start) / iterations
```

In [1]:

```
strings = ['foo', 'foobar', 'baz', 'qux', 'python', 'Guido Van Rossum'] * 100000
```

In [2]:

```
method1 = [x for x in strings if x.startswith('foo')]
```

In [3]:

```
method2 = [x for x in strings if x[:3] == 'foo']
```

In [4]:

```
%time method1 = [x for x in strings if x.startswith('foo')]
```

Wall time: 204 ms

In [5]:

```
%time method2 = [x for x in strings if x[:3] == 'foo']
```

Wall time: 150 ms

In [6]:

```
%timeit [x for x in strings if x.startswith('foo')]
```

213 ms ± 89.6 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [7]:

```
%timeit [x for x in strings if x[:3] == 'foo']
```

144 ms ± 84.5 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [8]:

```
x = 'foobar'
```

In [9]:

```
y = 'foo'
```

In [10]:

```
%timeit x.startswith(y)
```

The slowest run took 4.73 times longer than the fastest. This could mean that an intermediate result is being cached.

728 ns ± 558 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

In [11]:

```
%timeit x[:3] == y
```

The slowest run took 7.23 times longer than the fastest. This could mean that an intermediate result is being cached.

654 ns ± 705 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

In [12]:

```
import numpy as np
from numpy.linalg import eigvals
```

In [13]:

```
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)
```

```
%prun -l 7 -s cumulative run_experiment()
```

```
c.TerminalIPythonApp.extensions = ['line_profiler']
```

In [16]:

```
from numpy.random import randn
```

In [17]:

```
def add_and_sum(x, y):
    added = x + y
    summed = added.sum(axis=1)
    return summed
```

In [18]:

```
def call_function():
    x = randn(1000, 1000)
    y = randn(1000, 1000)
    return add_and_sum(x, y)
```

```
%run prof_mod
```

In [19]:

```
x = randn(3000, 3000)
```

In [20]:

```
y = randn(3000, 3000)
```

In [21]:

```
%prun add_and_sum(x, y)
```

In [22]:

```
%lprun -f add_and_sum add_and_sum(x, y)
```

...

In [23]:

```
%lprun -f add_and_sum -f call_function call_function()
```

...

B.4 Tips for Productive Code Development Using IPython

```
import some_lib
```

```
x = 5
y = [1, 2, 3, 4]
result = some_lib.get_answer(x, y)
```

```
import some_lib
import importlib
importlib.reload(some_lib)
```

```
from my_functions import g
```

In [24]:

```
def f(x, y):
    return g(x + y)
```

In [25]:

```
def main():
    x = 6
    y = 7.5
    result = x + y
```

In [26]:

```
if __name__ == '__main__':
    main()
```

B.5 Advanced IPython Features

```
class Message:
    def __init__(self, msg):
        self.msg = msg
```

```
x = Message('I have a secret')
```

```
class Message:
    def __init__(self, msg):
        self.msg = msg
```

In [27]:

```
def __repr__(self):
    return 'Message: %s' % self.msg
```

```
x = Message('I have a secret')
```

