Lecture

June 27, 2025

1 Scientific Programming with Python:

2 Data structures - NumPy, Pandas & beyond

Federica Lionetto (federica.lionetto@gmail.com)

The content of the lecture might be reused, also in parts, under the CC-licence by-sa 4.0

2.1 NumPy

2.1.1 NumPy vs. standard Python, need for speed

[1]:
```
# Creating a standard Python list
L = list(range(1000))
```

[2]:
```
# How long does it take to calculate the element-wise square?
%timeit [i**2 for i in L]
```

```
44.7 s ± 1.72 s per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

[3]:
```
# Now do the same with a NumPy array
import numpy as np
a = np.arange(1000)
```

[4]:
```
%timeit a**2
```

```
937 ns ± 24.9 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

2.1.2 Details about NumPy

[5]:
```
np.__version__
```

[5]:
```
'2.2.4'
```

[6]:
```
np.show_config()
```

```
Build Dependencies:
  blas:
    detection method: pkgconfig
    found: true
```

```
    include directory: /usr/include/x86_64-linux-gnu
    lib directory: /usr/lib/x86_64-linux-gnu
    name: blas
    openblas configuration: unknown
    pc file directory: /usr/lib/x86_64-linux-gnu/pkgconfig
    version: 3.12.1
  lapack:
    detection method: pkgconfig
    found: true
    include directory: /usr/include/x86_64-linux-gnu
    lib directory: /usr/lib/x86_64-linux-gnu
    name: lapack
    openblas configuration: unknown
    pc file directory: /usr/lib/x86_64-linux-gnu/pkgconfig
    version: 3.12.1
Compilers:
  c:
    args: -g, -O2, -Werror=implicit-function-declaration, -ffile-prefix-
map=$BUILDDIR=.,
      -fstack-protector-strong, -fstack-clash-protection, -Wformat,
-Werror=format-security,
      -fcf-protection, -Wdate-time, -D_FORTIFY_SOURCE=2
    commands: cc
    linker: ld.bfd
    linker args: -Wl,-z,relro, -g, -O2, -Werror=implicit-function-declaration,
-ffile-prefix-map=$BUILDDIR=.,
      -fstack-protector-strong, -fstack-clash-protection, -Wformat,
-Werror=format-security,
      -fcf-protection, -Wdate-time, -D_FORTIFY_SOURCE=2
    name: gcc
    version: 14.2.0
  c++:
    args: -g, -O2, -ffile-prefix-map=$BUILDDIR=., -fstack-protector-strong,
-fstack-clash-protection,
      -Wformat, -Werror=format-security, -fcf-protection, -Wdate-time,
-D_FORTIFY_SOURCE=2
    commands: c++
    linker: ld.bfd
    linker args: -Wl,-z,relro, -g, -O2, -ffile-prefix-map=$BUILDDIR=., -fstack-
protector-strong,
      -fstack-clash-protection, -Wformat, -Werror=format-security, -fcf-
protection,
      -Wdate-time, -D_FORTIFY_SOURCE=2
    name: gcc
    version: 14.2.0
  cython:
    commands: cython
    linker: cython
```

```
    name: cython
    version: 3.0.11
Machine Information:
  build:
    cpu: x86_64
    endian: little
    family: x86_64
    system: linux
  host:
    cpu: x86_64
    endian: little
    family: x86_64
    system: linux
SIMD Extensions:
  baseline:
  - SSE
  - SSE2
  - SSE3
  found:
  - SSSE3
  - SSE41
  - POPCNT
  - SSE42
  - AVX
  - F16C
  - FMA3
  - AVX2
  not found:
  - AVX512F
  - AVX512CD
  - AVX512_KNL
  - AVX512_KNM
  - AVX512_SKX
  - AVX512_CLX
  - AVX512_CNL
  - AVX512_ICL
  - AVX512_SPR
```

BLAS: Basic Linear Algebra Subprograms
LAPACK: Linear Algebra Package

2.1.3 Creating NumPy arrays

[7]:
```
a = np.array([1,2,4])
print(a)
```

```
[1 2 4]
```

[8]:
```
b = np.arange(1,15,3)
print(b)
```

```
[ 1  4  7 10 13]
```

Note: You should not use `np.arange` to create arrays containing floating point numbers. `arange` uses a `==` comparison for the last element. This can be affected by floating-point imprecisions.

[9]:
```
c = np.linspace(0,1,6)
print(c)
```

```
[0.  0.2 0.4 0.6 0.8 1. ]
```

[10]:
```
d = np.empty((1,3))
print(d)
```

```
[[3.37887e-319 0.00000e+000 0.00000e+000]]
```

[11]:
```
e = np.zeros((2,5,3))
print(e)
```

```
[[[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.]]]
```

[12]:
```
f = np.ones((3,3))
print(f)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

[13]:
```
g = np.eye(4)
print(g)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

[14]:
```
h = np.identity(4)
print(h)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

[15]:
```
i = np.diag(np.array([1,2,3,4]))
print(i)
```

```
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
```

[16]:
```
l = np.diag(np.array([1,2,3,4]),k=-1)
print(l)
```

```
[[0 0 0 0 0]
 [1 0 0 0 0]
 [0 2 0 0 0]
 [0 0 3 0 0]
 [0 0 0 4 0]]
```

[17]:
```
m = np.diag(np.array([1,2,3,4]),k=2)
print(m)
```

```
[[0 0 1 0 0 0]
 [0 0 0 2 0 0]
 [0 0 0 0 3 0]
 [0 0 0 0 0 4]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]]
```

[18]:
```
# arrays = [a,b,c,d,e,f,g,h,i,l,m]
# for array in arrays :
#     print(array)
#     print('')
```

2.1.4 NumPy arrays of random numbers

[19]:
```
a = np.random.rand(4)
b = np.random.rand(4,3)
c = np.random.randint(1,3,(2,3))
d = np.random.randn(4,5)
e = np.random.poisson(3,5)

arrays = [a,b,c,d,e]
for array in arrays :
    print(array)
    print('')
```

```
[0.79675125 0.79834459 0.80720566 0.84975774]

[[0.39868422 0.75006983 0.46613861]
 [0.63299908 0.72617476 0.73996072]
 [0.77651208 0.15466578 0.38101609]
 [0.30016638 0.79985435 0.48315564]]

[[2 1 2]
 [1 1 2]]

[[ 0.75524    -0.36309407 -0.62805736  1.8463238   0.27922017]
 [-1.38953922 -0.97571376  0.70919036 -2.82000355 -1.12213861]
 [-1.05716836  1.66680457  2.3606188  -0.37471271  0.85836016]
 [-0.45245693  0.06581539 -1.21390832 -1.21211947 -0.11421357]]

[4 7 4 6 8]
```

[20]:
```
# Random seed
np.random.seed(1)
arr1 = np.random.rand(5)
print('Array with 5 elements, random seed 10:')
print(arr1)
```

```
Array with 5 elements, random seed 10:
[4.17022005e-01 7.20324493e-01 1.14374817e-04 3.02332573e-01
 1.46755891e-01]
```

[21]:
```
arr2 = np.random.rand(10)
print('Array with 10 elements, random seed not set:')
print(arr2)
```

```
Array with 10 elements, random seed not set:
[0.09233859 0.18626021 0.34556073 0.39676747 0.53881673 0.41919451
 0.6852195  0.20445225 0.87811744 0.02738759]
```

[22]:
```
np.random.seed(1)
arr3 = np.random.rand(10)
print('Array with 10 elements, random seed 10:')
print(arr3)
```

```
Array with 10 elements, random seed 10:
[4.17022005e-01 7.20324493e-01 1.14374817e-04 3.02332573e-01
 1.46755891e-01 9.23385948e-02 1.86260211e-01 3.45560727e-01
 3.96767474e-01 5.38816734e-01]
```

2.1.5 Basic operations

[23]:
```
a = np.random.rand(3,4)
b = np.random.rand(3,4)
print(a)
print('')
print(b)
```

```
[[0.41919451 0.6852195  0.20445225 0.87811744]
 [0.02738759 0.67046751 0.4173048  0.55868983]
 [0.14038694 0.19810149 0.80074457 0.96826158]]

[[0.31342418 0.69232262 0.87638915 0.89460666]
 [0.08504421 0.03905478 0.16983042 0.8781425 ]
 [0.09834683 0.42110763 0.95788953 0.53316528]]
```

[24]:
```
a+b
```

[24]:
```
array([[0.73261869, 1.37754212, 1.0808414 , 1.7727241 ],
       [0.1124318 , 0.70952229, 0.58713522, 1.43683233],
       [0.23873377, 0.61920911, 1.7586341 , 1.50142686]])
```

[25]:
```
a-b
```

[25]:
```
array([[ 0.10577034, -0.00710312, -0.6719369 , -0.01648923],
       [-0.05765662,  0.63141273,  0.24747438, -0.31945267],
       [ 0.0420401 , -0.22300614, -0.15714496,  0.43509629]])
```

[26]:
```
a*b
```

[26]:
```
array([[0.1313857 , 0.47439296, 0.17917973, 0.78556971],
       [0.00232916, 0.02618496, 0.07087105, 0.49060928],
       [0.01380661, 0.08342205, 0.76702484, 0.51624346]])
```

[27]:
```
a/b
```

[27]:
```
array([[ 1.33746706,  0.98974017,  0.23328934,  0.98156818],
       [ 0.32203948, 17.16735966,  2.45718525,  0.63621773],
       [ 1.4274678 ,  0.47042959,  0.83594668,  1.81606268]])
```

[28]:
```
# Add 3.0 to every element
a+3.0
```

[28]:
```
array([[3.41919451, 3.6852195 , 3.20445225, 3.87811744],
       [3.02738759, 3.67046751, 3.4173048 , 3.55868983],
       [3.14038694, 3.19810149, 3.80074457, 3.96826158]])
```

[29]:
```
# Conditions
a>b
```

[29]:
```
array([[ True, False, False, False],
       [False,  True,  True, False],
       [ True, False, False,  True]])
```

[30]:
```
a.min()
```

[30]:
```
np.float64(0.027387593197926163)
```

[31]:
```
a.min(axis=0)
```

[31]:
```
array([0.02738759, 0.19810149, 0.20445225, 0.55868983])
```

[32]:
```
a.min(axis=1)
```

[32]:
```
array([0.20445225, 0.02738759, 0.14038694])
```

[33]:
```
# Numpy has its own set of functions
np.exp(b)
```

[33]:
```
array([[1.36810173, 1.99835155, 2.40221001, 2.44637335],
       [1.0887652 , 1.03982745, 1.18510386, 2.40642563],
       [1.1033454 , 1.52364825, 2.60619038, 1.70431843]])
```

[34]:
```
# Numpy has its own set of functions
np.cos(b)
```

[34]:
```
array([[0.95128341, 0.7697655 , 0.63993001, 0.62582565],
       [0.99638592, 0.99923746, 0.98561344, 0.63858169],
       [0.99516785, 0.91263673, 0.57524759, 0.86120259]])
```

[35]:
```
# Functions in the math library are not able to handle multi-element data
import math
math.exp(b)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[35], line 3
      1 # Functions in the math library are not able to handle multi-element da
      2 import math
----> 3 math.exp(b)

TypeError: only length-1 arrays can be converted to Python scalars
```

2.1.6 Data representation

[36]:
```
a = np.array([1,0,-2],dtype=np.int64)
print(a)
```

```
[ 1  0 -2]
```

[37]:
```
b = np.array(a,dtype=np.int8)
print(b)
```

```
[ 1  0 -2]
```

[38]:
```
c = np.array(a,dtype=int) # the default python int
print(c)
```

```
[ 1  0 -2]
```

[39]:
```
d = np.array(a,dtype=np.float64)
print(d)
```

```
[ 1.  0. -2.]
```

[40]:
```
e = np.array(a,dtype=np.bool)
print(e)
```

```
[ True False  True]
```

[41]:
```
e.dtype
```

[41]:
```
dtype('bool')
```

[42]:
```
print('3 elements np.int64 correspond to', a.nbytes, 'bytes')
print('3 elements np.int8 correspond to', b.nbytes, 'bytes')
print('3 elements np.int correspond to', c.nbytes, 'bytes')
print('3 elements np.float64 correspond to', d.nbytes, 'bytes')
print('3 elements np.bool correspond to', e.nbytes, 'bytes')
```

```
3 elements np.int64 correspond to 24 bytes
3 elements np.int8 correspond to 3 bytes
3 elements np.int correspond to 24 bytes
3 elements np.float64 correspond to 24 bytes
3 elements np.bool correspond to 3 bytes
```

[43]:
```
a = np.ones((3,4),dtype=np.int8)
b = np.ones((3,4),dtype=np.int64)
```

[44]:
```
print('np.int8 bytes:')
print([hex(el) for el in a.tobytes(order='A')])
print('')
print('np.int64 bytes:')
print([hex(el) for el in b.tobytes(order='A')])
```

```
np.int8 bytes:
['0x1', '0x1', '0x1', '0x1', '0x1', '0x1', '0x1', '0x1', '0x1', '0x1', '0x1',
'0x1']

np.int64 bytes:
['0x1', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x1', '0x0', '0x0',
'0x0', '0x0', '0x0', '0x0', '0x0', '0x1', '0x0', '0x0', '0x0', '0x0', '0x0',
'0x0', '0x0', '0x1', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x1',
'0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x1', '0x0', '0x0', '0x0',
'0x0', '0x0', '0x0', '0x0', '0x1', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0',
'0x0', '0x1', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x1', '0x0',
'0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x1', '0x0', '0x0', '0x0', '0x0',
'0x0', '0x0', '0x0', '0x1', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0',
'0x1', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0']
```

[45]:
```
16*16-1
```

[45]: 255

[46]:
```
# in numpy < 2.0, the following line silently overflows, leading to unexpected␣
 ↪entries
# in numpy >= 2.0, an OverflowError is raised
a = np.array([1,10,11,16,255,256],dtype=np.uint8)
print(a) # array([  1,  10,  11,  16, 255,   0] in numpy < 2.0
b = np.array([1,10,11,16,255,256],dtype=np.int64)
print(b)
```

```
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
Cell In[46], line 3
      1 # in numpy < 2.0, the following line silently overflows, leading to␣
 ↪unexpected entries
      2 # in numpy >= 2.0, an OverflowError is raised
----> 3 a = np.array([1,10,11,16,255,256],dtype=np.uint8)
      4 print(a) # array([  1,  10,  11,  16, 255,   0] in numpy < 2.0
      5 b = np.array([1,10,11,16,255,256],dtype=np.int64)

OverflowError: Python integer 256 out of bounds for uint8
```

[47]:
```
# even in numpy >= 2.0 you are not totaly safe though:
c = np.array([1,10,11,16,255],dtype=np.uint8)+1
print(c)
```

```
[ 2 11 12 17  0]
```

2.1.7 Data structure

[48]:
```
a = np.ones((3,4),dtype=np.int8)
b = np.ones((3,4),dtype=np.int64)
```

[49]:
```
print(a.ndim)
print(a.shape)
print(a.size)
print(a.itemsize)
```

```
2
(3, 4)
12
1
```

[50]:
```
print(b.ndim)
print(b.shape)
print(b.size)
print(b.itemsize)
```

```
2
(3, 4)
12
8
```

[51]:
```
print(a.nbytes)
print(b.nbytes)
```

```
12
96
```

[52]:
```
print(a.data)
```

```
<memory at 0x7f86bb9a25a0>
```

[53]:
```
print(a.data.tobytes())
print(a.tobytes())
```

```
b'\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01'
b'\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01'
```

[54]:
```
print(a.flags)
print('')
print(a.T.flags)
```

```
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
```

```
  C_CONTIGUOUS : False
  F_CONTIGUOUS : True
  OWNDATA : False
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
```

[55]:
```
print(a.strides)
print(b.strides)
print(a.T.strides)
```

```
(4, 1)
(32, 8)
(1, 4)
```

2.1.8 Shape manipulation

[56]:
```
# Let's define an array of values distributed according to a Normal distribution
a = np.random.randn(3,4)
print(a)
```

```
[[-1.10061918  1.14472371  0.90159072  0.50249434]
 [ 0.90085595 -0.68372786 -0.12289023 -0.93576943]
 [-0.26788808  0.53035547 -0.69166075 -0.39675353]]
```

[57]:
```
print(a.reshape(1,12))
print('')
print(a)
```

```
[[-1.10061918  1.14472371  0.90159072  0.50249434  0.90085595 -0.68372786
  -0.12289023 -0.93576943 -0.26788808  0.53035547 -0.69166075 -0.39675353]]

[[-1.10061918  1.14472371  0.90159072  0.50249434]
 [ 0.90085595 -0.68372786 -0.12289023 -0.93576943]
 [-0.26788808  0.53035547 -0.69166075 -0.39675353]]
```

[58]:
```
print(a.resize(1,12))
print('')
print(a)
```

```
None

[[-1.10061918  1.14472371  0.90159072  0.50249434  0.90085595 -0.68372786
  -0.12289023 -0.93576943 -0.26788808  0.53035547 -0.69166075 -0.39675353]]
```

[59]:
```
# Need to define a as in the beginning again
a = np.random.randn(3,4)
print(a)
```

```
[[-0.6871727  -0.84520564 -0.67124613 -0.0126646 ]
 [-1.11731035  0.2344157   1.65980218  0.74204416]
 [-0.19183555 -0.88762896 -0.74715829  1.6924546 ]]
```

[60]:
```
print(a.ravel())
print('')
print(a)
```

```
[-0.6871727  -0.84520564 -0.67124613 -0.0126646  -1.11731035  0.2344157
  1.65980218  0.74204416 -0.19183555 -0.88762896 -0.74715829  1.6924546 ]

[[-0.6871727  -0.84520564 -0.67124613 -0.0126646 ]
 [-1.11731035  0.2344157   1.65980218  0.74204416]
 [-0.19183555 -0.88762896 -0.74715829  1.6924546 ]]
```

[61]:
```
print(a.T)
```

```
[[-0.6871727  -1.11731035 -0.19183555]
 [-0.84520564  0.2344157  -0.88762896]
 [-0.67124613  1.65980218 -0.74715829]
 [-0.0126646   0.74204416  1.6924546 ]]
```

[62]:
```
# Bad practices
b = np.random.randn(4)
print(b.shape)
print(b.T.shape)
```

```
(4,)
(4,)
```

[63]:
```
# Good practices
c = np.random.randn(4,1)
print(c.shape)
print(c.T.shape)
```

```
(4, 1)
(1, 4)
```

2.1.9 Accessing array elements

[64]:
```
a = np.ones((3,4),dtype=np.int64)
print(a)
```

```
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

[65]:
```
b = a
```

[66]:
```
a[0,0]=0
```

[67]:
```
print(b)
```

```
[[0 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

[68]:
```
c = a.copy()
```

[69]:
```
a[1,1]=0
```

[70]:
```
print(a)
```

```
[[0 1 1 1]
 [1 0 1 1]
 [1 1 1 1]]
```

[71]:
```
print(c)
```

```
[[0 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

[72]:
```
print(b)
```

```
[[0 1 1 1]
 [1 0 1 1]
 [1 1 1 1]]
```

2.1.10 Get the data

[73]:
```
# Let's have a look at the loadEx.txt file
!head loadEx.txt
```

```
94.820 76.280 33.020 29.660 25.460
91.610 71.480 31.710 29.610 25.460
94.820 68.130 32.630 31.460 25.910
93.190 71.050 35.250 31.780 27.020
92.780 71.320 35.950 32.700 27.490
96.460 72.880 35.780 32.240 26.750
97.280 73.260 35.160 31.640 26.500
97.690 72.880 35.250 31.780 26.410
96.660 72.990 35.250 31.270 26.650
97.790 73.960 34.750 32.470 26.480
```

[74]:
```
data = np.loadtxt('loadEx.txt',delimiter=' ',comments="#")
```

[75]:
```
print(data)
```

```
[[ 94.82  76.28  33.02  29.66  25.46]
 [ 91.61  71.48  31.71  29.61  25.46]
 [ 94.82  68.13  32.63  31.46  25.91]
 …
 [160.51 196.19 166.71 132.29 113.32]
 [160.05 195.71 165.84 131.83 113.11]
 [160.44 193.83 164.32 129.98 112.18]]
```

[76]:
```
print(data.shape)
```

```
(3773, 5)
```

[77]:
```
data[0,1]
```

[77]:
```
np.float64(76.28)
```

[78]:
```
# A more complex example
dt = np.dtype([('name','S7'),('mass',np.float32),
 ('position',[('x',np.float32),('y',np.float32),('z',np.float32)]),
 ('velocity',[('x',np.float32),('y',np.float32),('z',np.float32)])])
```

[79]:
```
solarData = np.loadtxt('Solar.txt',dtype=dt)
```

[80]:
```
print(solarData)
```

```
[(b'Sun', 3.3294600e+05, ( 2.13e-03, -1.60e-03, -1.20e-04), ( 5.01e-06,
4.08e-06, -1.24e-07))
 (b'Mercury', 5.5273525e-02, ( 1.62e-01,  2.64e-01,  6.94e-03), (-2.97e-02,
1.56e-02,  4.00e-03))
 (b'Venus', 8.1499749e-01, ( 3.02e-01,  6.54e-01, -8.44e-03), (-1.85e-02,
8.32e-03,  1.18e-03))
 (b'Earth', 1.0000000e+00, ( 5.66e-01, -8.46e-01, -9.12e-05), ( 1.40e-02,
9.49e-03, -5.81e-07))
 (b'Mars', 1.0744685e-01, (-4.34e-01, -1.43e+00, -1.93e-02), ( 1.39e-02,
-2.88e-03, -4.02e-04))
 (b'Jupiter', 3.1782812e+02, (-2.78e+00,  4.47e+00,  4.35e-02), (-6.50e-03,
-3.62e-03,  1.61e-04))
 (b'Saturn', 9.5160904e+01, (-6.08e+00, -7.84e+00,  3.78e-01), ( 4.10e-03,
-3.43e-03, -1.04e-04))
 (b'Uranus', 1.4535757e+01, ( 1.95e+01,  4.68e+00, -2.35e-01), (-9.48e-04,
3.64e-03,  2.58e-05))
 (b'Neptune', 1.7146999e+01, ( 2.73e+01, -1.23e+01, -3.77e-01), ( 1.27e-03,
2.88e-03, -8.85e-05))
 (b'Pluto', 2.1909999e-03, ( 6.91e+00, -3.19e+01,  1.42e+00), ( 3.14e-03,
3.08e-05, -9.18e-04))
 (b'Halley', 3.6800001e-11, (-2.05e+01,  2.51e+01, -9.76e+00), (-7.71e-05,
9.54e-04, -1.79e-04))
```

```
 (b'Moon', 1.2303100e-02, ( 5.64e-01, -8.44e-01, -3.23e-04), ( 1.36e-02,
9.18e-03,  8.97e-06))]
```

[81]:
```
solarData['name']
```

[81]:
```
array([b'Sun', b'Mercury', b'Venus', b'Earth', b'Mars', b'Jupiter',
       b'Saturn', b'Uranus', b'Neptune', b'Pluto', b'Halley', b'Moon'],
      dtype='|S7')
```

[82]:
```
solarData['position']['x']
```

[82]:
```
array([ 2.13e-03,  1.62e-01,  3.02e-01,  5.66e-01, -4.34e-01, -2.78e+00,
       -6.08e+00,  1.95e+01,  2.73e+01,  6.91e+00, -2.05e+01,  5.64e-01],
      dtype=float32)
```

[83]:
```
solarData['position']['x'][np.where(solarData['name']==b'Sun')]
```

[83]:
```
array([0.00213], dtype=float32)
```

2.1.11 Broadcasting

[84]:
```
a = np.random.rand(3,5)
b = np.random.rand(8)
```

[85]:
```
c = a[...,np.newaxis]*b
print(c.shape)
```

```
(3, 5, 8)
```

[86]:
```
d = np.random.rand(1,10)
e = np.random.rand(10,1)
print(d.shape)
print(d)
print('')
print(e.shape)
print(e)
```

```
(1, 10)
[[0.57367949 0.00287033 0.61714491 0.3266449  0.5270581  0.8859421
  0.35726976 0.90853515 0.62336012 0.01582124]]

(10, 1)
[[0.92943723]
 [0.69089692]
 [0.99732285]
 [0.17234051]
 [0.13713575]
 [0.93259546]
 [0.69681816]
```

```
 [0.06600017]
 [0.75546305]
 [0.75387619]]
```

[87]:
```
# Explicit broadcasting.
dd,ee = np.broadcast_arrays(d,e)
print(dd.shape)
print(ee.shape)
```

```
(10, 10)
(10, 10)
```

[88]:
```
d[0,0]=-1.0
```

[89]:
```
dd
```

[89]:
```
array([[-1.        ,  0.00287033,  0.61714491,  0.3266449 ,  0.5270581 ,
         0.8859421 ,  0.35726976,  0.90853515,  0.62336012,  0.01582124],
       [-1.        ,  0.00287033,  0.61714491,  0.3266449 ,  0.5270581 ,
         0.8859421 ,  0.35726976,  0.90853515,  0.62336012,  0.01582124],
       [-1.        ,  0.00287033,  0.61714491,  0.3266449 ,  0.5270581 ,
         0.8859421 ,  0.35726976,  0.90853515,  0.62336012,  0.01582124],
       [-1.        ,  0.00287033,  0.61714491,  0.3266449 ,  0.5270581 ,
         0.8859421 ,  0.35726976,  0.90853515,  0.62336012,  0.01582124],
       [-1.        ,  0.00287033,  0.61714491,  0.3266449 ,  0.5270581 ,
         0.8859421 ,  0.35726976,  0.90853515,  0.62336012,  0.01582124],
       [-1.        ,  0.00287033,  0.61714491,  0.3266449 ,  0.5270581 ,
         0.8859421 ,  0.35726976,  0.90853515,  0.62336012,  0.01582124],
       [-1.        ,  0.00287033,  0.61714491,  0.3266449 ,  0.5270581 ,
         0.8859421 ,  0.35726976,  0.90853515,  0.62336012,  0.01582124],
       [-1.        ,  0.00287033,  0.61714491,  0.3266449 ,  0.5270581 ,
         0.8859421 ,  0.35726976,  0.90853515,  0.62336012,  0.01582124],
       [-1.        ,  0.00287033,  0.61714491,  0.3266449 ,  0.5270581 ,
         0.8859421 ,  0.35726976,  0.90853515,  0.62336012,  0.01582124],
       [-1.        ,  0.00287033,  0.61714491,  0.3266449 ,  0.5270581 ,
         0.8859421 ,  0.35726976,  0.90853515,  0.62336012,  0.01582124]])
```

[90]:
```
# ee
```

[91]:
```
print(dd.strides)
print(ee.strides)
```

```
(0, 8)
(8, 0)
```

2.1.12 Simple indexing

[92]:
```
# Notice that this does not use additional memory!!!
a = np.arange(100).reshape(10,10)
```

[93]:
```
# Access rows
a[4:9]
```

[93]:
```
array([[40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89]])
```

[94]:
```
# Access columns
a[:,3:8]
```

[94]:
```
array([[ 3,  4,  5,  6,  7],
       [13, 14, 15, 16, 17],
       [23, 24, 25, 26, 27],
       [33, 34, 35, 36, 37],
       [43, 44, 45, 46, 47],
       [53, 54, 55, 56, 57],
       [63, 64, 65, 66, 67],
       [73, 74, 75, 76, 77],
       [83, 84, 85, 86, 87],
       [93, 94, 95, 96, 97]])
```

[95]:
```
# Negative indices
a[:,-1]
```

[95]:
```
array([ 9, 19, 29, 39, 49, 59, 69, 79, 89, 99])
```

[96]:
```
# Ranges
a[-2::-3,1:6:2]
```

[96]:
```
array([[81, 83, 85],
       [51, 53, 55],
       [21, 23, 25]])
```

2.1.13 Fancy indexing

[97]:
```
a[:,[1,3,1]]
```

[97]:
```
array([[ 1,  3,  1],
       [11, 13, 11],
       [21, 23, 21],
       [31, 33, 31],
```

```
       [41, 43, 41],
       [51, 53, 51],
       [61, 63, 61],
       [71, 73, 71],
       [81, 83, 81],
       [91, 93, 91]])
```

[98]:
```
a[[1,3,1]][:,[1,3,1]]
```

[98]:
```
array([[11, 13, 11],
       [31, 33, 31],
       [11, 13, 11]])
```

[99]:
```
a[[1,3,1],[1,3,1]]
```

[99]:
```
array([11, 33, 11])
```

[100]:
```
# Multidimensional arrays indexed by multidimensional arrays.
y = np.arange(35).reshape(5,7)
print(y)
```

```
[[ 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 32 33 34]]
```

[101]:
```
# If the index arrays have a matching shape,
# and there is an index array for each dimension of the array being indexed,
# the resultant array has the same shape as the index arrays,
# and the values correspond to the index set for each position in the index␣
 ↪arrays.
# [0,0], [2,1], and [4,2] elements of the indexed array.
y[np.array([0,2,4]), np.array([0,1,2])]
```

[101]:
```
array([ 0, 15, 30])
```

[102]:
```
# If the index arrays do not have the same shape, a broadcasting is tried.
# [0,1], [2,1], and [4,1] elements of the indexed array.
y[np.array([0,2,4]), 1]
```

[102]:
```
array([ 1, 15, 29])
```

[103]:
```
# If we provide just one index array, the rows are selected but the columns are␣
 ↪kept as they were in the indexed array.
y[np.array([0,2,4])]
```

[103]:
```
array([[ 0,  1,  2,  3,  4,  5,  6],
       [14, 15, 16, 17, 18, 19, 20],
       [28, 29, 30, 31, 32, 33, 34]])
```

[104]:
```
# Fancy indexing.
i0 = np.random.randint(0,10,(8,1,8)) # Matrix of random integers between 0 and␣
 ↪10 with shape (8,1,8).
i1 = np.random.randint(0,10,(2,8)) # Matrix of random integers between 0 and 10␣
 ↪with shape (2,8).
```

[105]:
```
a[i0,i1] # creates a 8×2×8 array
```

[105]:
```
array([[[80, 92, 28, 72, 51, 54, 40, 54],
        [81, 97, 23, 71, 56, 56, 49, 56]],

       [[80, 52, 88, 12, 11, 84, 70,  4],
        [81, 57, 83, 11, 16, 86, 79,  6]],

       [[30, 42, 28,  2, 31, 54, 10, 24],
        [31, 47, 23,  1, 36, 56, 19, 26]],

       [[40, 32,  8, 62,  1, 74, 20, 84],
        [41, 37,  3, 61,  6, 76, 29, 86]],

       [[30,  2, 88, 42, 21, 94,  0, 34],
        [31,  7, 83, 41, 26, 96,  9, 36]],

       [[80, 12, 48, 32, 31, 64, 70, 34],
        [81, 17, 43, 31, 36, 66, 79, 36]],

       [[50, 32, 28, 42, 41,  4, 30, 34],
        [51, 37, 23, 41, 46,  6, 39, 36]],

       [[80, 32, 58, 62, 71, 54, 10, 74],
        [81, 37, 53, 61, 76, 56, 19, 76]]])
```

[106]:
```
a[i0,i1].shape
```

[106]:
```
(8, 2, 8)
```

[107]:
```
i0[0,0,0]
```

[107]:
```
np.int64(8)
```

[108]:
```
i1[0,0]
```

[108]:
```
np.int64(0)
```

[109]:
```
a[8,0]
```

[109]:
```
np.int64(80)
```

[110]:
```
i0[1,0,0]
```

[110]:
```
np.int64(8)
```

[111]:
```
i1[0,0]
```

[111]:
```
np.int64(0)
```

[112]:
```
a[7,7]
```

[112]:
```
np.int64(77)
```

3 Pandas

[113]:
```
import pandas as pd
```

[114]:
```
!head SMI.csv
```

```
Date,Open,High,Low,Close,Adj Close,Volume
1990-11-09,1378.900024,1389.0,1375.300049,1387.099976,1387.099976,0.0
1990-11-12,1388.099976,1408.099976,1388.099976,1407.5,1407.5,0.0
1990-11-13,1412.199951,1429.400024,1411.400024,1415.199951,1415.199951,0.0
1990-11-14,1413.599976,1413.599976,1402.099976,1410.300049,1410.300049,0.0
1990-11-15,1410.599976,1416.699951,1405.099976,1405.699951,1405.699951,0.0
1990-11-16,1405.699951,1407.400024,1389.400024,1395.199951,1395.199951,0.0
1990-11-19,1395.599976,1417.900024,1395.599976,1416.0,1416.0,0.0
1990-11-20,1414.800049,1415.0,1404.699951,1405.800049,1405.800049,0.0
1990-11-21,1405.599976,1405.599976,1396.699951,1398.400024,1398.400024,0.0
```

3.0.1 Loading of data and basic manipulation

[115]:
```
# Series object
s = pd.Series([1,3,5,np.nan,6,8])
print(s)
print(type(s))
```

```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
<class 'pandas.core.series.Series'>
```

[116]:
```
# Dataframe object
ts = pd.read_csv('SMI.csv')
```

[117]:
```
type(ts)
```

[117]:
```
pandas.core.frame.DataFrame
```

[118]:
```
ts.head()
```

[118]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1990-11-09	1378.900024	1389.000000	1375.300049	1387.099976	1387.099976	0.0
1	1990-11-12	1388.099976	1408.099976	1388.099976	1407.500000	1407.500000	0.0
2	1990-11-13	1412.199951	1429.400024	1411.400024	1415.199951	1415.199951	0.0
3	1990-11-14	1413.599976	1413.599976	1402.099976	1410.300049	1410.300049	0.0
4	1990-11-15	1410.599976	1416.699951	1405.099976	1405.699951	1405.699951	0.0

[119]:
```
ts.tail()
```

[119]:

	Date	Open	High	Low	Close	Adj Close	Volume
6738	2017-08-28	8864.230469	8864.230469	8864.230469	8864.230469	8864.230469	0.0
6739	2017-08-29	8814.540039	8814.540039	8814.540039	8814.540039	8814.540039	0.0
6740	2017-08-30	8851.259766	8851.259766	8851.259766	8851.259766	8851.259766	0.0
6741	2017-08-31	8925.450195	8925.450195	8925.450195	8925.450195	8925.450195	0.0
6742	2017-09-01	8941.620117	8941.620117	8941.620117	8941.620117	8941.620117	0.0

[120]:
```
ts.index
```

[120]:
```
RangeIndex(start=0, stop=6743, step=1)
```

[121]:
```
ts.columns
```

[121]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'],
dtype='object')
```

[122]:
```
ts['Open'][:10].values
```

[122]:
```
array([1378.900024, 1388.099976, 1412.199951, 1413.599976, 1410.599976,
       1405.699951, 1395.599976, 1414.800049, 1405.599976, 1400.      ])
```

[123]:
```
ts = ts.sort_values('Date')
ts.head()
```

[123]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1990-11-09	1378.900024	1389.000000	1375.300049	1387.099976	1387.099976	0.0
1	1990-11-12	1388.099976	1408.099976	1388.099976	1407.500000	1407.500000	0.0
2	1990-11-13	1412.199951	1429.400024	1411.400024	1415.199951	1415.199951	0.0
3	1990-11-14	1413.599976	1413.599976	1402.099976	1410.300049	1410.300049	0.0
4	1990-11-15	1410.599976	1416.699951	1405.099976	1405.699951	1405.699951	0.0

[124]:
```
# Find minimum and maximum values in a given column
print(ts['Volume'].min())
print(ts['Volume'].max())
```

```
0.0
346767700.0
```

[125]:
```
# Find index corresponding to mininum and maximum values in a given column
# Careful!!!
print(ts['Volume'].idxmin())
print(ts['Volume'].idxmax())
```

```
0
6079
```

[126]:
```
# Access rows
ts[6079:6080]
```

[126]:

	Date	Open	High	Low	Close	Adj Close	Volume
6079	2015-01-15	9259.200195	9277.200195	7932.200195	8400.599609	8400.599609	346767700.0

[127]:
```
# Modify index
ts.index = pd.to_datetime(ts.pop("Date"))
```

[128]:
```
ts = ts.sort_index()
```

[129]:
```
ts.tail()
```

[129]:
```
                  Open         High          Low        Close    Adj Close  \
Date
2017-08-28  8864.230469  8864.230469  8864.230469  8864.230469  8864.230469
2017-08-29  8814.540039  8814.540039  8814.540039  8814.540039  8814.540039
2017-08-30  8851.259766  8851.259766  8851.259766  8851.259766  8851.259766
2017-08-31  8925.450195  8925.450195  8925.450195  8925.450195  8925.450195
2017-09-01  8941.620117  8941.620117  8941.620117  8941.620117  8941.620117

            Volume
Date
2017-08-28     0.0
2017-08-29     0.0
2017-08-30     0.0
2017-08-31     0.0
2017-09-01     0.0
```

[130]:
```
import datetime as dt
ts[ts.index>dt.datetime(2010,1,1)].head()
```

[130]:
```
                  Open         High          Low        Close    Adj Close  \
Date
2010-01-04  6578.500000  6631.399902  6576.000000  6631.399902  6631.399902
2010-01-05  6620.700195  6622.399902  6547.399902  6579.299805  6579.299805
2010-01-06  6598.200195  6607.799805  6550.100098  6559.399902  6559.399902
2010-01-07  6536.500000  6574.200195  6494.899902  6555.399902  6555.399902
2010-01-08  6574.700195  6635.799805  6574.000000  6617.899902  6617.899902

                Volume
Date
2010-01-04  59150000.0
2010-01-05  65848500.0
2010-01-06  52305400.0
2010-01-07  64539000.0
2010-01-08  74761300.0
```

[131]:
```
ts["Adj Close"].head()
```

[131]:
```
Date
1990-11-09    1387.099976
1990-11-12    1407.500000
1990-11-13    1415.199951
1990-11-14    1410.300049
1990-11-15    1405.699951
```

```
Name: Adj Close, dtype: float64
```

[132]:
```
ts["Adj Close"].describe()
```

[132]:
```
count    6743.000000
mean     5957.266658
std      2236.843089
min      1287.599976
25%      4561.000000
50%      6374.700195
75%      7790.649902
max      9531.500000
Name: Adj Close, dtype: float64
```

[133]:
```
# Access parameters of describe
ts['Adj Close'].describe()['count']
```

[133]:
```
np.float64(6743.0)
```

3.0.2 Timeseries applications

[134]:
```
# Resampling of time series
# Creating a series with 9 timestamps, each one corresponding to one minute
index = pd.date_range('1/6/2018', periods=9, freq='min')
series = pd.Series(range(9), index=index)
print(series)
```

```
2018-01-06 00:00:00    0
2018-01-06 00:01:00    1
2018-01-06 00:02:00    2
2018-01-06 00:03:00    3
2018-01-06 00:04:00    4
2018-01-06 00:05:00    5
2018-01-06 00:06:00    6
2018-01-06 00:07:00    7
2018-01-06 00:08:00    8
Freq: min, dtype: int64
```

[135]:
```
# Downsample the series in bins of 3 minutes each and sum over the same bin
series.resample('3min').sum()
```

[135]:
```
2018-01-06 00:00:00     3
2018-01-06 00:03:00    12
2018-01-06 00:06:00    21
Freq: 3min, dtype: int64
```

[136]:
```
# Label the bin using the upper bound
series.resample('3min', label='right').sum()
```

[136]:
```
2018-01-06 00:03:00     3
2018-01-06 00:06:00    12
2018-01-06 00:09:00    21
Freq: 3min, dtype: int64
```

[137]:
```
# DataFrame.resample(rule, axis=0)
# The object must have a datetime-like index
ts_monthly = ts["Adj Close"].resample("ME").
 ↪apply(["median","mean","std","count","max","min"]).head()
```

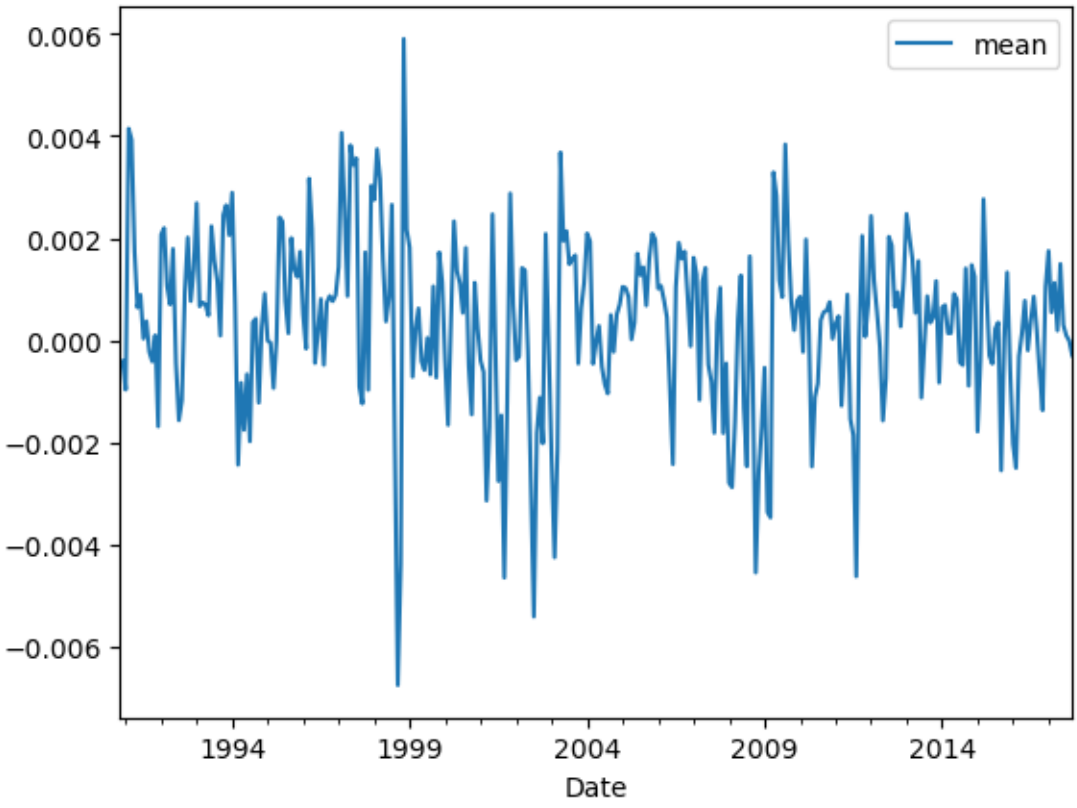
[138]:
```
ts_monthly
```

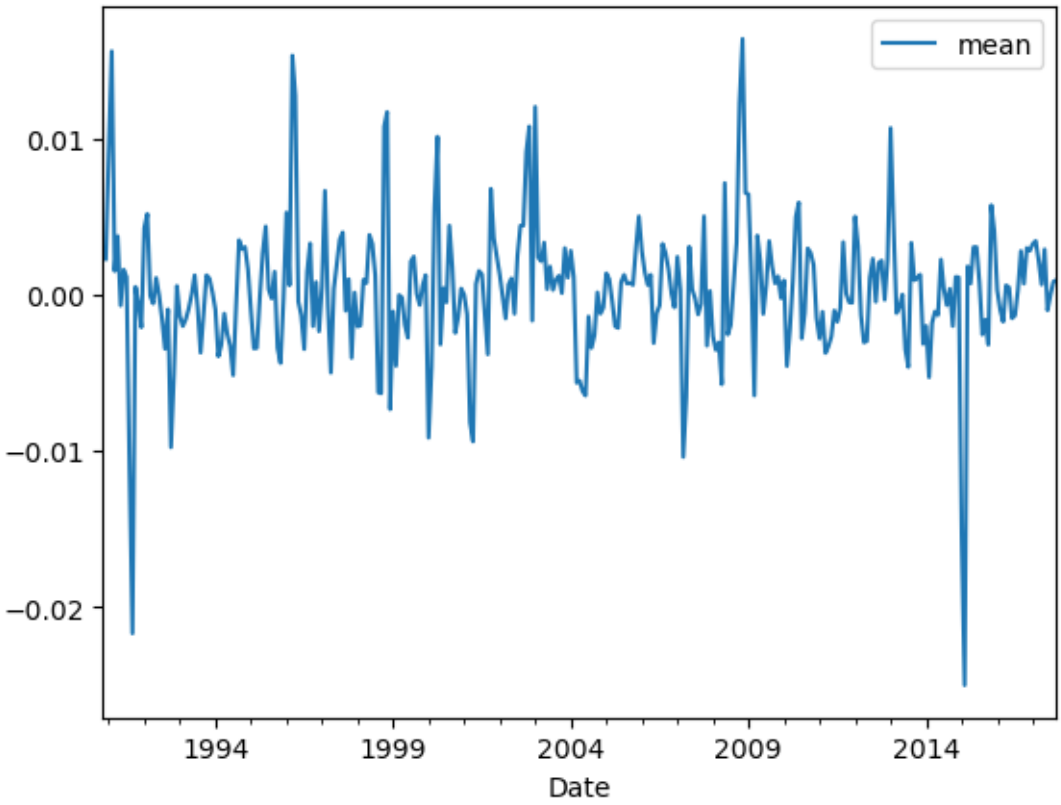
[138]:

Date	median	mean	std	count	max	min
1990-11-30	1392.000000	1390.387497	20.156853	16	1416.000000	1353.699951
1990-12-31	1405.349976	1404.744446	21.675076	18	1450.300049	1371.199951
1991-01-31	1350.000000	1357.580956	44.510815	21	1438.599976	1287.599976
1991-02-28	1538.799988	1530.924988	50.931718	20	1603.199951	1448.099976
1991-03-31	1614.649964	1611.519995	24.735199	20	1650.599976	1559.000000

3.0.3 Basic visualisation

[139]:
```
day_return = ts["Adj Close"].pct_change().dropna()
mean_30day = day_return.rolling(30).mean()

import numpy as np

minmax_30day = day_return.rolling(30).apply(lambda x: (np.max(x)+np.min(x))*0.5)

mean_30day.resample("ME").apply(["mean"]).plot()
minmax_30day.resample("ME").apply(["mean"]).plot()

import matplotlib.pyplot as plt
plt.show()
```


mean
0.006
0.004
0.002
0.000
−0.002
−0.004
−0.006
1994
1999
2004
2009
2014
Date



3.0.4 Creating timeseries and filling missing values

[140]:
```
dates = pd.date_range(ts.index.min(),ts.index.max(),freq="D")
print(dates)
```

```
DatetimeIndex(['1990-11-09', '1990-11-10', '1990-11-11', '1990-11-12',
               '1990-11-13', '1990-11-14', '1990-11-15', '1990-11-16',
               '1990-11-17', '1990-11-18',
               …
               '2017-08-23', '2017-08-24', '2017-08-25', '2017-08-26',
               '2017-08-27', '2017-08-28', '2017-08-29', '2017-08-30',
               '2017-08-31', '2017-09-01'],
              dtype='datetime64[ns]', length=9794, freq='D')
```

[141]:
```
ts_alldays = pd.Series(index=dates,data=ts["Adj Close"])
```

[142]:
```
ts_alldays.head()
```

[142]:
```
1990-11-09    1387.099976
1990-11-10            NaN
1990-11-11            NaN
```

```
1990-11-12    1407.500000
1990-11-13    1415.199951
Freq: D, Name: Adj Close, dtype: float64
```

[143]:
```
ts_alldays.ffill(inplace=True)
ts_alldays.head()
```

[143]:
```
1990-11-09    1387.099976
1990-11-10    1387.099976
1990-11-11    1387.099976
1990-11-12    1407.500000
1990-11-13    1415.199951
Freq: D, Name: Adj Close, dtype: float64
```

3.0.5 Hook up to data sources

http://pandas-datareader.readthedocs.io/en/latest/

[144]:
```
# pd.__version__
```

[145]:
```
# pip install pandas-datareader
```

[146]:
```
version = [int(v) for v in pd.__version__.split('.')]
if (version[0] == 0 and version[1] >= 17) or (version[0] >= 1): # Test if␣
 ↪version is >= 0.17
    from pandas_datareader import data, wb
else:
    from pandas.io import data, wb
```

[147]:
```
# Retrieve information from FRED
import datetime
start = datetime.datetime(2010, 1, 1)
end = datetime.datetime(2018, 1, 1)
# df = data.DataReader('F', 'google', start, end)
df = data.DataReader('GDP', 'fred', start, end)
print(df.shape)
print(df.head())
print(df.tail())
```

```
(33, 1)
                GDP
DATE
2010-01-01  14764.610
2010-04-01  14980.193
2010-07-01  15141.607
2010-10-01  15309.474
2011-01-01  15351.448
                GDP
```

```
DATE
2017-01-01  19280.084
2017-04-01  19438.643
2017-07-01  19692.595
2017-10-01  20037.088
2018-01-01  20328.553
```

[148]:
```
# Let's say we want to compare the Gross Domestic Products per capita in␣
 ↪constant dollars in North America
# wb.search('gdp.*capita.*const')
```

3.0.6 Spreadsheet operations

[149]:
```
# Let's use the download function to acquire the data from the World Bank's␣
 ↪servers
# gdp_data = wb.download(indicator='NY.GDP.PCAP.
 ↪KD',country=['CH','US','GB','DE'],start=2006,end=2016)
# gdp_data.head(20)
```

[150]:
```
# gdp_data.shape
```

[151]:
```
# gdp_data.columns
```

[152]:
```
# gdp_data.unstack(level=0)
```

[153]:
```
# gdp_data.unstack(level=1)
```

[154]:
```
# gdp_data.groupby(level=0).mean()
```

[155]:
```
# gdp_data.groupby(level=0).std()
```

3.0.7 And some further work with Dataframes

[156]:
```
df_us_zip = pd.read_csv("us_postal_codes.csv")
```

[157]:
```
df_us_zip.shape
```

[157]:
```
(40933, 7)
```

[158]:
```
df_us_zip.columns
```

[158]:
```
Index(['Zip Code', 'Place Name', 'State', 'State Abbreviation', 'County',
       'Latitude', 'Longitude'],
      dtype='object')
```

[159]:
```
df_us_zip.describe()
```

[159]:
```
           Zip Code      Latitude     Longitude
count  40933.000000  40933.000000  40933.000000
mean   49819.569858     38.596225    -91.082332
std    27808.948650      5.255750     15.763730
min      501.000000      7.112800   -176.658100
25%    26451.000000     35.052600    -97.308100
50%    49036.000000     39.152200    -87.976700
75%    73042.000000     41.894300    -80.142300
max    99950.000000     71.234600    171.237000
```

[160]:
```
df_us_zip.dtypes
```

[160]:
```
Zip Code                int64
Place Name             object
State                  object
State Abbreviation     object
County                 object
Latitude              float64
Longitude             float64
dtype: object
```

[161]:
```
df_us_zip.head()
```

[161]:
```
   Zip Code  Place Name          State State Abbreviation     County  \
0       501  Holtsville       New York                NY    Suffolk
1       544  Holtsville       New York                NY    Suffolk
2      1001      Agawam  Massachusetts                MA    Hampden
3      1002     Amherst  Massachusetts                MA  Hampshire
4      1003     Amherst  Massachusetts                MA  Hampshire

   Latitude  Longitude
0   40.8154   -73.0451
1   40.8154   -73.0451
2   42.0702   -72.6227
3   42.3671   -72.4646
4   42.3919   -72.5248
```

[162]:
```
df_us_state_coord = df_us_zip.get(["State␣
↪Abbreviation","Latitude","Longitude"]).groupby(["State Abbreviation"]).mean()
```

[163]:
```
df_us_state_coord.shape
```

[163]:
```
(57, 2)
```

[164]:
```
df_us_state_coord.head()
```

[164]:
```
                     Latitude   Longitude
State Abbreviation
```

```
AA                  33.036400  -82.249300
AK                  61.456423 -152.486981
AL                  32.886361  -86.813639
AP                  32.349325 -112.935950
AR                  35.124723  -92.402676
```

[165]:
```
# How many entries have "Washington" as "Place Name"?
# Let's count the unique values in the "Place Name" field
# Series.value_counts(normalize=False, sort=True, ascending=False, bins=None,␣
↪dropna=True)
df_us_zip["Place Name"].value_counts().head()
```

[165]:
```
Place Name
Washington    295
Houston       187
New York      146
El Paso       139
Dallas        114
Name: count, dtype: int64
```

[166]:
```
# Cross-check
df_us_zip[df_us_zip['Place Name']=='Washington'].shape
```

[166]:
```
(295, 7)
```

[167]:
```
df_us_places = df_us_zip.get(["Place Name","State␣
↪Abbreviation","Latitude","Longitude"])
df_us_places = df_us_places.groupby(["Place Name","State Abbreviation"]).mean()
print(df_us_places.shape)
df_us_places
```

```
(29545, 2)
```

[167]:

Place Name	State Abbreviation	Latitude	Longitude
APO	AA	33.03640	-82.2493
	AP	34.28285	-105.0628
Aaronsburg	PA	40.89870	-77.4562
Abbeville	AL	31.57550	-85.2790
	GA	31.96480	-83.3068
…		…	…
Zuni	NM	35.06840	-108.8336
	VA	36.84370	-76.8110
Zurich	MT	48.58440	-109.0304
Zwingle	IA	42.27750	-90.7507
Zwolle	LA	31.61380	-93.6636

[29545 rows x 2 columns]

[168]:
```
df_us_places.reset_index(inplace=True)
print(df_us_places.shape)
print(df_us_places.columns)
```

```
(29545, 4)
Index(['Place Name', 'State Abbreviation', 'Latitude', 'Longitude'],
dtype='object')
```

[169]:
```
df_us_places["Place Name"].value_counts().head()
```

[169]:
```
Place Name
Franklin       27
Clinton        26
Madison        26
Washington     26
Springfield    24
Name: count, dtype: int64
```

[170]:
```
# Cross-check
df_us_places[df_us_places['Place Name']=='Franklin'].shape
```

[170]:
```
(27, 4)
```

[171]:
```
# Mapping
# Map values of Series using input correspondence (a dict, Series, or function).
# Series.map(arg, na_action=None)
df_us_places["isSwiss"] = df_us_places["Place Name"].map(lambda x: any([s in x␣
 ↪for s in ["Zurich", "Berne", "Basel", "Lucerne", "Glarus", "Geneva"]]))
df_us_places[df_us_places["isSwiss"]]
```

[171]:

	Place Name	State Abbreviation	Latitude	Longitude	isSwiss
2096	Berne	IN	40.6716	-84.9343	True
2097	Berne	NY	42.6108	-74.1466	True
7523	East Berne	NY	42.6191	-74.0555	True
9916	Geneva	AL	31.0414	-85.8847	True
9917	Geneva	FL	28.7503	-81.1114	True
9918	Geneva	GA	32.5799	-84.5508	True
9919	Geneva	IA	42.6755	-93.1294	True
9920	Geneva	ID	42.3136	-111.0722	True
9921	Geneva	IL	41.8860	-88.3110	True
9922	Geneva	IN	40.6071	-84.9621	True
9923	Geneva	MN	43.8235	-93.2671	True
9924	Geneva	NE	40.5277	-97.6096	True
9925	Geneva	NY	42.8637	-76.9913	True
9926	Geneva	OH	41.8029	-80.9474	True
14181	Lake Geneva	FL	29.7683	-81.9907	True
14182	Lake Geneva	WI	42.5881	-88.4554	True
14253	Lake Zurich	IL	42.2165	-88.0769	True

```
15522         Lucerne                CA   39.0783  -122.7846     True
15523         Lucerne                CO   40.4824  -104.7054     True
15524         Lucerne                IN   40.8614   -86.4077     True
15525         Lucerne                MO   40.4382   -93.2867     True
15526  Lucerne Valley                CA   34.4470  -116.9189     True
15527    Lucernemines                PA   40.5567   -79.1515     True
18544      New Geneva                PA   39.7884   -79.9092     True
18547      New Glarus                WI   42.8143   -89.6437     True
29542          Zurich                MT   48.5844  -109.0304     True
```

3.0.8 Merging data

[172]:
```
df1 = df_us_zip[:5].copy()
df2 = df_us_zip[5:10].copy()
print(df1.head())
print(df2.head())
```

```
   Zip Code  Place Name          State State Abbreviation     County  \
0       501  Holtsville       New York                NY    Suffolk
1       544  Holtsville       New York                NY    Suffolk
2      1001      Agawam  Massachusetts                MA    Hampden
3      1002     Amherst  Massachusetts                MA  Hampshire
4      1003     Amherst  Massachusetts                MA  Hampshire

   Latitude  Longitude
0   40.8154   -73.0451
1   40.8154   -73.0451
2   42.0702   -72.6227
3   42.3671   -72.4646
4   42.3919   -72.5248
   Zip Code   Place Name          State State Abbreviation     County  \
5      1004      Amherst  Massachusetts                MA  Hampshire
6      1005        Barre  Massachusetts                MA  Worcester
7      1007  Belchertown  Massachusetts                MA  Hampshire
8      1008    Blandford  Massachusetts                MA    Hampden
9      1009   Bondsville  Massachusetts                MA    Hampden

   Latitude  Longitude
5   42.3845   -72.5132
6   42.4097   -72.1084
7   42.2751   -72.4110
8   42.1829   -72.9361
9   42.2061   -72.3405
```

[173]:
```
dfs = [df1,df2]
```

[174]:
```
result = pd.concat(dfs)
print(result)
```

```
   Zip Code   Place Name          State State Abbreviation     County  \
0       501   Holtsville       New York                NY    Suffolk
1       544   Holtsville       New York                NY    Suffolk
2      1001       Agawam  Massachusetts                MA    Hampden
3      1002      Amherst  Massachusetts                MA  Hampshire
4      1003      Amherst  Massachusetts                MA  Hampshire
5      1004      Amherst  Massachusetts                MA  Hampshire
6      1005        Barre  Massachusetts                MA  Worcester
7      1007  Belchertown  Massachusetts                MA  Hampshire
8      1008    Blandford  Massachusetts                MA    Hampden
9      1009   Bondsville  Massachusetts                MA    Hampden

   Latitude  Longitude
0   40.8154   -73.0451
1   40.8154   -73.0451
2   42.0702   -72.6227
3   42.3671   -72.4646
4   42.3919   -72.5248
5   42.3845   -72.5132
6   42.4097   -72.1084
7   42.2751   -72.4110
8   42.1829   -72.9361
9   42.2061   -72.3405
```

[175]:
```
df1 = df_us_zip[['Zip Code','Place Name','State']][:5].copy()
df2 = df_us_zip[['Zip Code','Latitude','Longitude']][3:8].copy()
dfs = [df1,df2]
print(df1)
print(df2)
```

```
   Zip Code  Place Name          State
0       501  Holtsville       New York
1       544  Holtsville       New York
2      1001      Agawam  Massachusetts
3      1002     Amherst  Massachusetts
4      1003     Amherst  Massachusetts
   Zip Code  Latitude  Longitude
3      1002   42.3671   -72.4646
4      1003   42.3919   -72.5248
5      1004   42.3845   -72.5132
6      1005   42.4097   -72.1084
7      1007   42.2751   -72.4110
```

[176]:
```
result = pd.concat(dfs,axis=1)
print(result)
```

```
   Zip Code  Place Name          State  Zip Code  Latitude  Longitude
0     501.0  Holtsville       New York       NaN       NaN        NaN
1     544.0  Holtsville       New York       NaN       NaN        NaN
```

```
2    1001.0      Agawam  Massachusetts       NaN       NaN        NaN
3    1002.0     Amherst  Massachusetts    1002.0   42.3671   -72.4646
4    1003.0     Amherst  Massachusetts    1003.0   42.3919   -72.5248
5       NaN         NaN            NaN    1004.0   42.3845   -72.5132
6       NaN         NaN            NaN    1005.0   42.4097   -72.1084
7       NaN         NaN            NaN    1007.0   42.2751   -72.4110
```

[177]:
```
result = pd.merge(df1,df2,how='inner',on='Zip Code')
print(result)
```

```
   Zip Code Place Name          State  Latitude  Longitude
0      1002    Amherst  Massachusetts   42.3671   -72.4646
1      1003    Amherst  Massachusetts   42.3919   -72.5248
```

[178]:
```
result = pd.merge(df1,df2,how='left',on='Zip Code')
print(result)
```

```
   Zip Code  Place Name          State  Latitude  Longitude
0       501  Holtsville       New York       NaN        NaN
1       544  Holtsville       New York       NaN        NaN
2      1001      Agawam  Massachusetts       NaN        NaN
3      1002     Amherst  Massachusetts   42.3671   -72.4646
4      1003     Amherst  Massachusetts   42.3919   -72.5248
```

[179]:
```
result = pd.merge(df1,df2,how='right',on='Zip Code')
print(result)
```

```
   Zip Code Place Name          State  Latitude  Longitude
0      1002    Amherst  Massachusetts   42.3671   -72.4646
1      1003    Amherst  Massachusetts   42.3919   -72.5248
2      1004        NaN            NaN   42.3845   -72.5132
3      1005        NaN            NaN   42.4097   -72.1084
4      1007        NaN            NaN   42.2751   -72.4110
```

[180]:
```
result = pd.merge(df1,df2,how='outer',on='Zip Code')
print(result)
```

```
   Zip Code  Place Name          State  Latitude  Longitude
0       501  Holtsville       New York       NaN        NaN
1       544  Holtsville       New York       NaN        NaN
2      1001      Agawam  Massachusetts       NaN        NaN
3      1002     Amherst  Massachusetts   42.3671   -72.4646
4      1003     Amherst  Massachusetts   42.3919   -72.5248
5      1004         NaN            NaN   42.3845   -72.5132
6      1005         NaN            NaN   42.4097   -72.1084
7      1007         NaN            NaN   42.2751   -72.4110
```

3.1 Pickle, JSON and YAML files

[181]:
```
import pickle
import json
import yaml
```

Let's define a class Foo().

[182]:
```
class Foo():
    def __init__(self):
        self.x = "bar"
```

[183]:
```
# Create object of class Foo() and write to Pickle file
obj = Foo()
with open("example.pkl","wb") as f_o:
    pickle.dump(obj,f_o)
```

[184]:
```
# Show as string
pickle.dumps(obj)
```

[184]:
```
b'\x80\x04\x95%\x00\x00\x00\x00\x00\x00\x00\x8c\x08__main__\x94\x8c\x03Foo\x94\x
93\x94)\x81\x94}\x94\x8c\x01x\x94\x8c\x03bar\x94sb.'
```

[185]:
```
# Read from Pickle file
with open("example.pkl","rb") as f_i:
    new_obj = pickle.load(f_i)
print(new_obj.x)
```

```
bar
```

[186]:
```
# Create a dictionary and write to JSON file
entry = {"1" : "Hello", "2" : "Bye", "3" : 4.35}
with open("example.json","w") as f_o:
    json.dump(entry,f_o)
```

[187]:
```
# Show as string
json.dumps(entry)
```

[187]:
```
'{"1": "Hello", "2": "Bye", "3": 4.35}'
```

[188]:
```
# Read from JSON file
with open("example.json","r") as f_i:
    new_entry = json.load(f_i)
print(new_entry)
```

```
{'1': 'Hello', '2': 'Bye', '3': 4.35}
```

[189]:
```
# Create a dictionary and write to YAML file
data = {
```

```
    'first_data':[1,2,3,4,5],
    'second_data':'Just a string.',
    'third_data': dict(a=1.1,b=1.2,c=1.3),
}
with open('example.yaml','w') as f_o :
    yaml.dump(data,f_o,default_flow_style=False)
```

[190]:
```
# Read from YAML file
with open('example.yaml','r') as f_i:
    new_data = yaml.load(f_i, Loader=yaml.SafeLoader)
print(new_data)
print(new_data['third_data']['a'])
```

```
{'first_data': [1, 2, 3, 4, 5], 'second_data': 'Just a string.', 'third_data':
{'a': 1.1, 'b': 1.2, 'c': 1.3}}
1.1
```

[191]:
```
%%writefile example2.yaml
- &flag red
- green
- blue
- *flag
```

```
Overwriting example2.yaml
```

[192]:
```
!head example2.yaml
```

```
- &flag red
- green
- blue
- *flag
```

[193]:
```
with open('example2.yaml','r') as f_i:
    data_example2 = yaml.load(f_i, Loader=yaml.SafeLoader)
print(data_example2)
```

```
['red', 'green', 'blue', 'red']
```

4 sqlite3

[194]:
```
import sqlite3 as sql
!cp Solar_bkup.db Solar.db
```

[195]:
```
conn = sql.connect("Solar.db")
```

[196]:
```
results = conn.execute("SELECT * FROM solarsystem")
```

[197]:
```
results.description
```

[197]:
```
(('index', None, None, None, None, None, None),
 ('name', None, None, None, None, None, None),
 ('mass', None, None, None, None, None, None),
 ('x', None, None, None, None, None, None),
 ('y', None, None, None, None, None, None),
 ('z', None, None, None, None, None, None),
 ('vx', None, None, None, None, None, None),
 ('vy', None, None, None, None, None, None),
 ('vz', None, None, None, None, None, None))
```

[198]:
```
for row in results:
    print(row)
```

```
(0, 'Sun', 332946.0, 0.00213, -0.0016, -0.00011999999999999999, 5.01e-06,
4.08e-06, -1.24e-07)
(1, 'Mercury', 0.055273525999999996, 0.162, 0.264, 0.006940000000000001,
-0.0297, 0.0156, 0.004)
(2, 'Venus', 0.814997513, 0.302, 0.654, -0.008440000000000001, -0.0185,
0.008320000000000001, 0.00118)
(3, 'Earth', 1.0, 0.5660000000000001, -0.846, -9.120000000000001e-05, 0.014,
0.00949, -5.81e-07)
(4, 'Mars', 0.107446849, -0.434, -1.43, -0.0193, 0.0139, -0.00288,
-0.00040199999999999996)
(5, 'Jupiter', 317.828133, -2.78, 4.47, 0.0435, -0.0065, -0.0036200000000000004,
0.000161)
(6, 'Saturn', 95.1609041, -6.08, -7.84, 0.37799999999999995, 0.0041, -0.00343,
-0.00010400000000000001)
(7, 'Uranus', 14.5357566, 19.5, 4.68, -0.235, -0.0009480000000000001, 0.00364,
2.58e-05)
(8, 'Neptune', 17.147000000000002, 27.3, -12.3, -0.377, 0.0012699999999999999,
0.00288, -8.85e-05)
(9, 'Pluto', 0.002191, 6.91, -31.9, 1.42, 0.00314, 3.0799999999999996e-05,
-0.000918)
(10, 'Halley', 3.68e-11, -20.5, 25.1, -9.76, -7.709999999999999e-05,
0.0009539999999999999, -0.0001790000000000001)
(11, 'Moon', 0.0123031, 0.564, -0.8440000000000001, -0.000323, 0.0136,
0.009179999999999999, 8.97e-06)
```

[199]:
```
conn.execute("DELETE FROM solarsystem WHERE name='Pluto'")
conn.commit()
```

[200]:
```
results = conn.execute("SELECT * FROM solarsystem")
for row in results:
    print(row)
```

```
(0, 'Sun', 332946.0, 0.00213, -0.0016, -0.00011999999999999999, 5.01e-06,
4.08e-06, -1.24e-07)
(1, 'Mercury', 0.055273525999999996, 0.162, 0.264, 0.006940000000000001,
```

```
-0.0297, 0.0156, 0.004)
(2, 'Venus', 0.814997513, 0.302, 0.654, -0.008440000000000001, -0.0185,
0.008320000000000001, 0.00118)
(3, 'Earth', 1.0, 0.5660000000000001, -0.846, -9.120000000000001e-05, 0.014,
0.00949, -5.81e-07)
(4, 'Mars', 0.107446849, -0.434, -1.43, -0.0193, 0.0139, -0.00288,
-0.00040199999999999996)
(5, 'Jupiter', 317.828133, -2.78, 4.47, 0.0435, -0.0065, -0.0036200000000000004,
0.000161)
(6, 'Saturn', 95.1609041, -6.08, -7.84, 0.37799999999999995, 0.0041, -0.00343,
-0.00010400000000000001)
(7, 'Uranus', 14.5357566, 19.5, 4.68, -0.235, -0.0009480000000000001, 0.00364,
2.58e-05)
(8, 'Neptune', 17.147000000000002, 27.3, -12.3, -0.377, 0.0012699999999999999,
0.00288, -8.85e-05)
(10, 'Halley', 3.68e-11, -20.5, 25.1, -9.76, -7.709999999999999e-05,
0.0009539999999999999, -0.00017900000000000001)
(11, 'Moon', 0.0123031, 0.564, -0.8440000000000001, -0.000323, 0.0136,
0.009179999999999999, 8.97e-06)
```

[201]:
```
death_star = [12,'Death Star',0.1,0.564,-0.845,-9.12e-05,0.014,0.00949,-5.
↪81e-07]
conn.execute("INSERT INTO solarsystem VALUES (?,?,?,?,?,?,?,?,?)",death_star)
```

[201]:
```
<sqlite3.Cursor at 0x7f86b0461440>
```

[202]:
```
results = conn.execute("SELECT * FROM solarsystem")
for row in results:
    print(row)
```

```
(0, 'Sun', 332946.0, 0.00213, -0.0016, -0.00011999999999999999, 5.01e-06,
4.08e-06, -1.24e-07)
(1, 'Mercury', 0.055273525999999996, 0.162, 0.264, 0.006940000000000001,
-0.0297, 0.0156, 0.004)
(2, 'Venus', 0.814997513, 0.302, 0.654, -0.008440000000000001, -0.0185,
0.008320000000000001, 0.00118)
(3, 'Earth', 1.0, 0.5660000000000001, -0.846, -9.120000000000001e-05, 0.014,
0.00949, -5.81e-07)
(4, 'Mars', 0.107446849, -0.434, -1.43, -0.0193, 0.0139, -0.00288,
-0.00040199999999999996)
(5, 'Jupiter', 317.828133, -2.78, 4.47, 0.0435, -0.0065, -0.0036200000000000004,
0.000161)
(6, 'Saturn', 95.1609041, -6.08, -7.84, 0.37799999999999995, 0.0041, -0.00343,
-0.00010400000000000001)
(7, 'Uranus', 14.5357566, 19.5, 4.68, -0.235, -0.0009480000000000001, 0.00364,
2.58e-05)
(8, 'Neptune', 17.147000000000002, 27.3, -12.3, -0.377, 0.0012699999999999999,
0.00288, -8.85e-05)
```

```
(10, 'Halley', 3.68e-11, -20.5, 25.1, -9.76, -7.709999999999999e-05,
0.0009539999999999999, -0.00017900000000000001)
(11, 'Moon', 0.0123031, 0.564, -0.8440000000000001, -0.000323, 0.0136,
0.009179999999999999, 8.97e-06)
(12, 'Death Star', 0.1, 0.564, -0.845, -9.12e-05, 0.014, 0.00949, -5.81e-07)
```

[203]:
```
more_death_stars = list()
for i in range(10):
    death_star[0] +=1
    death_star[1]  = "Death Star "+str(i)
    more_death_stars.append(death_star.copy())
```

[204]:
```
conn.executemany("INSERT INTO solarsystem VALUES (?,?,?,?,?,?,?,?,?
 ↪)",more_death_stars)
```

[204]:
```
<sqlite3.Cursor at 0x7f86b0461940>
```

[205]:
```
conn.commit()
```

[206]:
```
def dict_factory(cursor, row):
    d = {}
    for idx,col in enumerate(cursor.description):
        d[col[0]] = row[idx]
    return d
conn.row_factory = dict_factory
```

[207]:
```
max_mass = 1.0
results = conn.execute("SELECT name,mass FROM solarsystem WHERE mass<?
 ↪",[max_mass])
for row in results:
    print(row)
```

```
{'name': 'Mercury', 'mass': 0.055273525999999996}
{'name': 'Venus', 'mass': 0.814997513}
{'name': 'Mars', 'mass': 0.107446849}
{'name': 'Halley', 'mass': 3.68e-11}
{'name': 'Moon', 'mass': 0.0123031}
{'name': 'Death Star', 'mass': 0.1}
{'name': 'Death Star 0', 'mass': 0.1}
{'name': 'Death Star 1', 'mass': 0.1}
{'name': 'Death Star 2', 'mass': 0.1}
{'name': 'Death Star 3', 'mass': 0.1}
{'name': 'Death Star 4', 'mass': 0.1}
{'name': 'Death Star 5', 'mass': 0.1}
{'name': 'Death Star 6', 'mass': 0.1}
{'name': 'Death Star 7', 'mass': 0.1}
{'name': 'Death Star 8', 'mass': 0.1}
{'name': 'Death Star 9', 'mass': 0.1}
```

[208]:
```
results = conn.execute("SELECT AVG(mass) as mean_mass, COUNT(*) as n, mass>1.0␣
 ↪as larger_than_earth "+
                       "FROM solarsystem WHERE mass<>1.0 GROUP BY mass<1.0")
for row in results:
    print(row)
```

```
{'mean_mass': 66678.13435874, 'n': 5, 'larger_than_earth': 1}
{'mean_mass': 0.1306263117523, 'n': 16, 'larger_than_earth': 0}
```