机器学习入门复习要点2 （Python学习笔记）

python部分

==========

Python 的函数具有非常灵活的参数形态，既可以实现简单的调用，又可以传入非常复杂的
参数。
其中一个*号代表list或者tuple，**代表map或者dic（字典）

默认参数一定要用不可变对象，如果是可变对象，运行会有逻辑错误！
要注意定义可变参数和关键字参数的语法：
***args 是可变参数， args 接收的是一个 tuple；**
****kw 是关键字参数， kw 接收的是一个 dict。**

以及调用函数时如何传入可变参数和关键字参数的语法：
可变参数既可以直接传入： func(1, 2, 3)，又可以先组装 list 或 tuple，再通过*args 传入：
func(*(1, 2, 3))；
关键字参数既可以直接传入： func(a=1, b=2)，又可以先组装 dict，再通过**kw 传入： func
(**{'a': 1, 'b': 2})。

==========

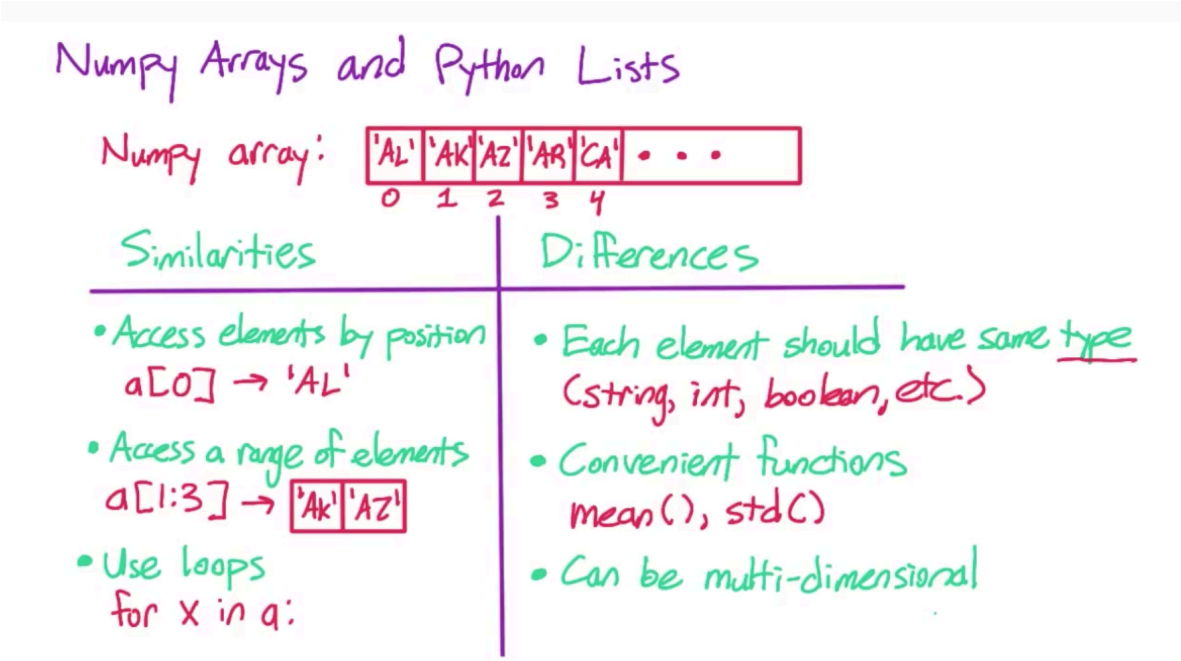
Python中的zip()与*zip()函数详解
https://www.cnblogs.com/waltsmith/p/8029539.html

==========

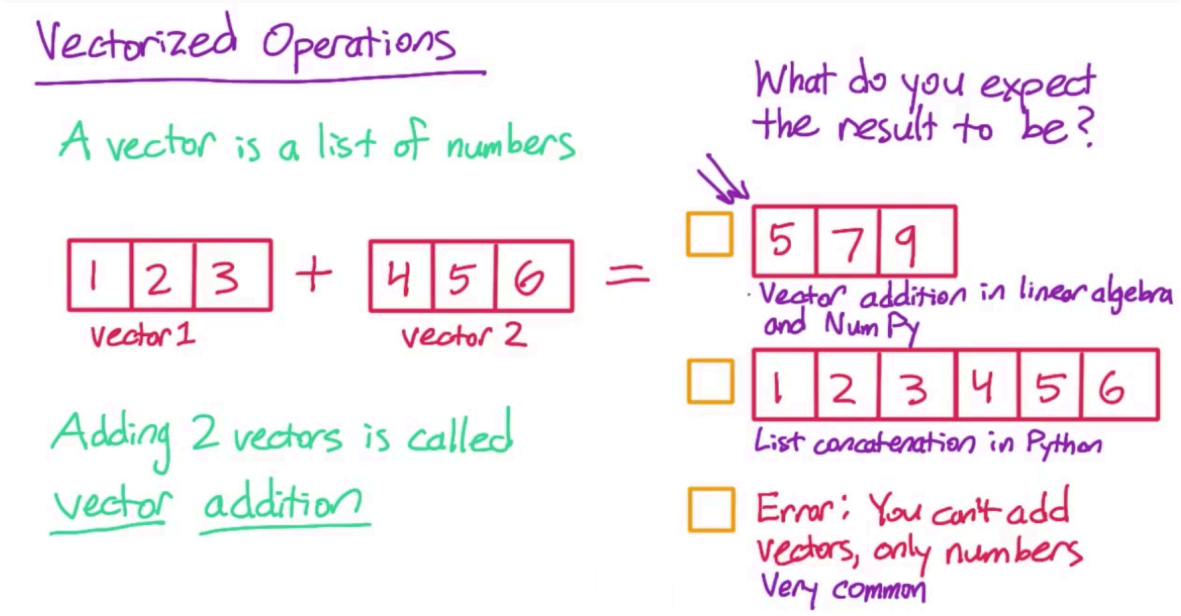
Numpy部分

一维运算部分

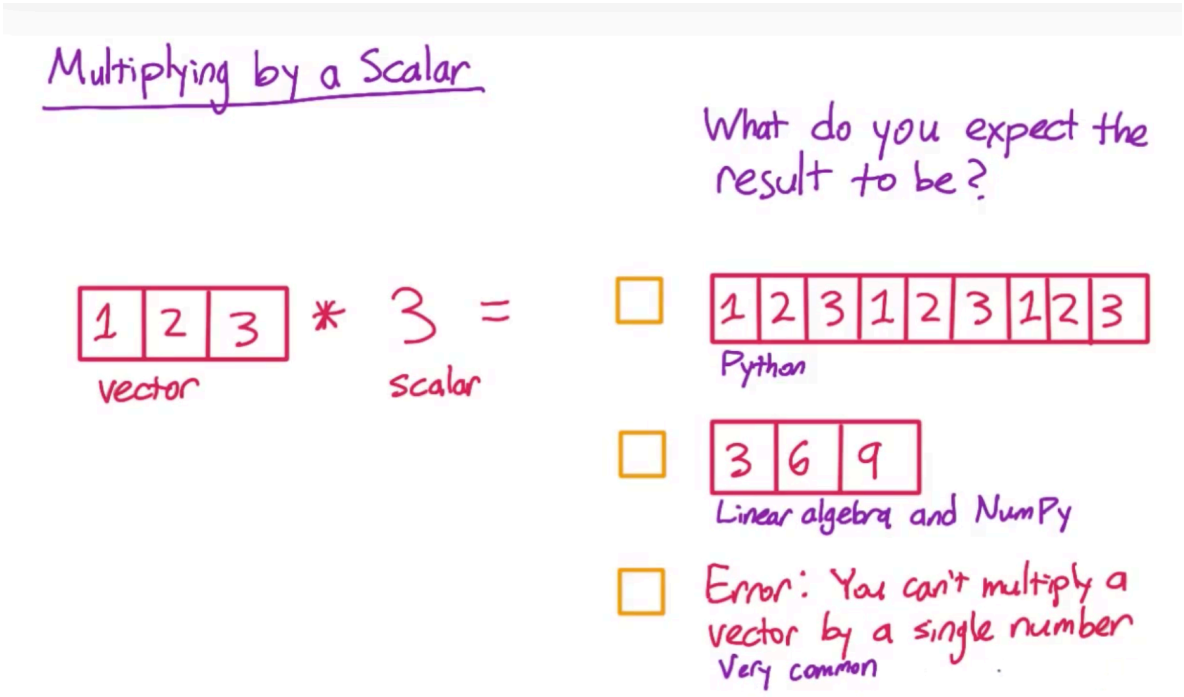
Numpy的List必须是同种类型的元素 而python则不必， 导致Numpy在计算mean() std时效率比较高



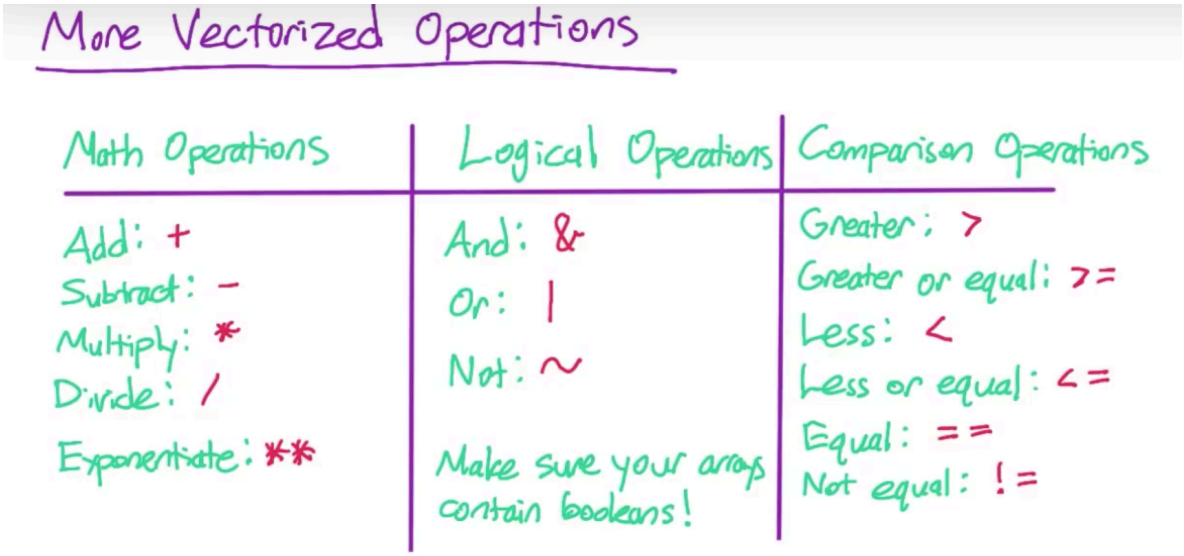
对于Numpy是向量操作（加、减、点积、叉积等），python是列表操作



向量乘以标量的运算中

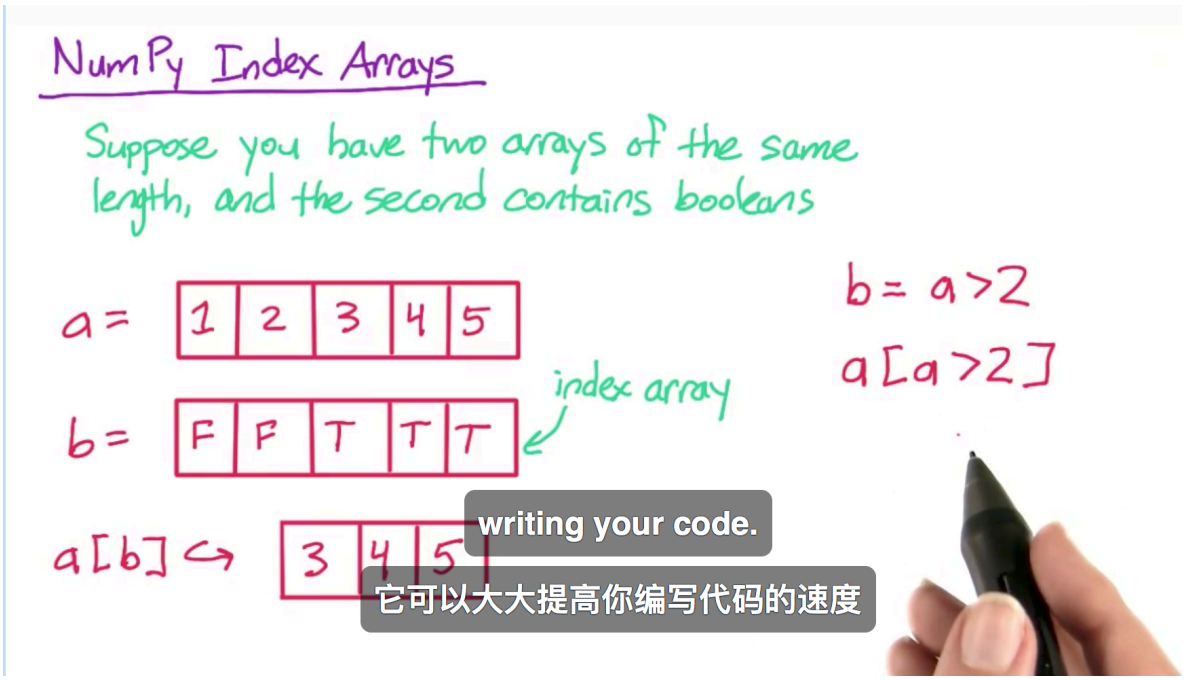


Numpy的其他运算

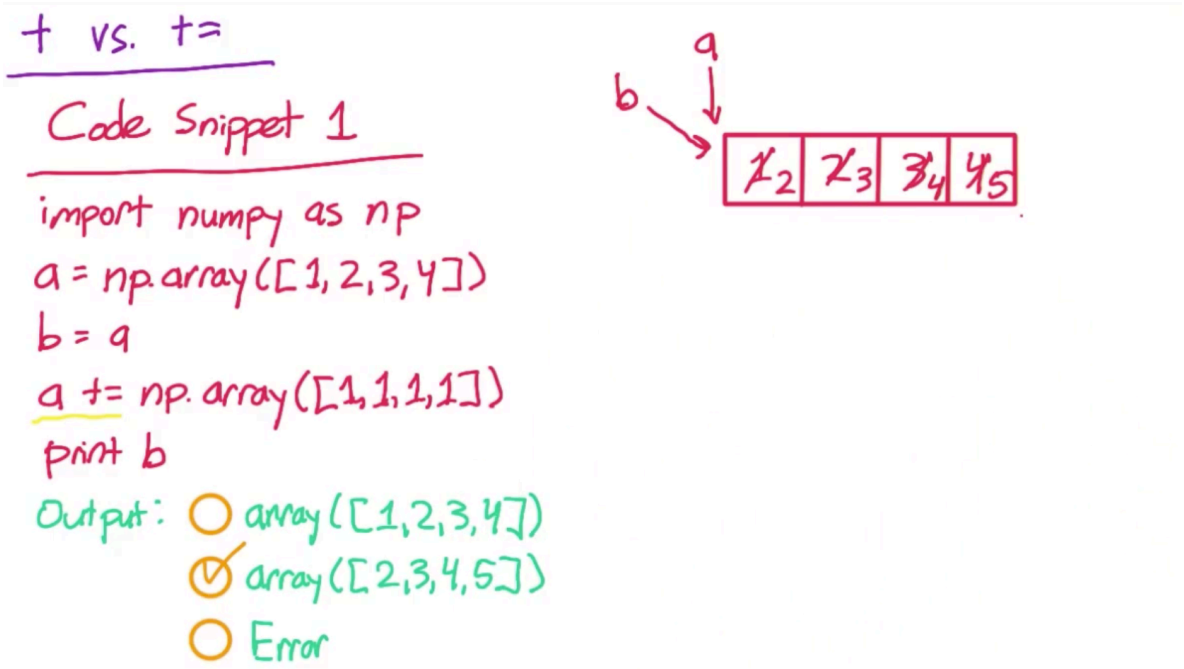


Numpy索引数组

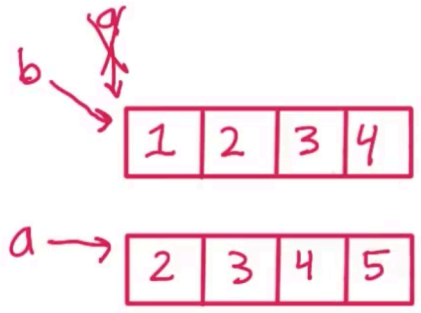
其中a[a>2] 括号内的a可以看作是遍历a数组中每一个元素 如果该元素大于2 则返回True，效果和索引数组b的作用一致



+ vs +=
+= 直接改变原指向元素
+ 创建新元素



+ vs. +=

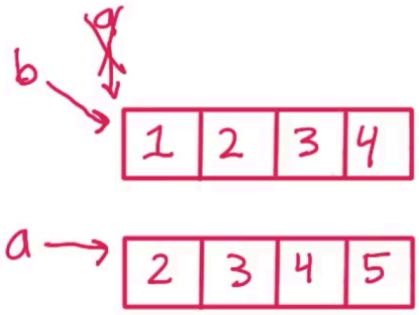


Code Snippet 2

```
import numpy as np
a = np.array([1,2,3,4])
b = a
a = a + np.array([1,1,1,1])
print b
```

Output:
- ✓ array([1,2,3,4])
- ○ array([2,3,4,5])
- ○ Error

+ vs. +=



Code Snippet 2

```
import numpy as np
a = np.array([1,2,3,4])
b = a
a = a + np.array([1,1,1,1])
print b
```

Output:
- ✓ array([1,2,3,4])
- ○ array([2,3,4,5])
- ○ Error

in-place vs not in-place

In-Place vs. Not In-Place

+= operates in-place while + does not

Operations that are not in-place are much easier to think about!

In-Place vs. Not In-Place

Code snippet:

```
import numpy as np
a = np.array([1,2,3,4,5])
slice = a[:3]
slice[0] = 100
print a
```

Output

- ○ array([1,2,3,4,5])
- ✓ array([100,2,3,4,5])
- ○ Error

==========

二维运算部分

Two-Dimensional Data

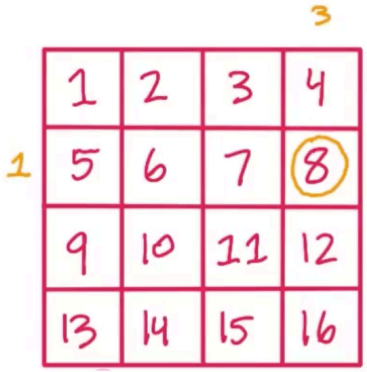
Python: List of lists

NumPy: 2D array ← start here

Pandas: DataFrame

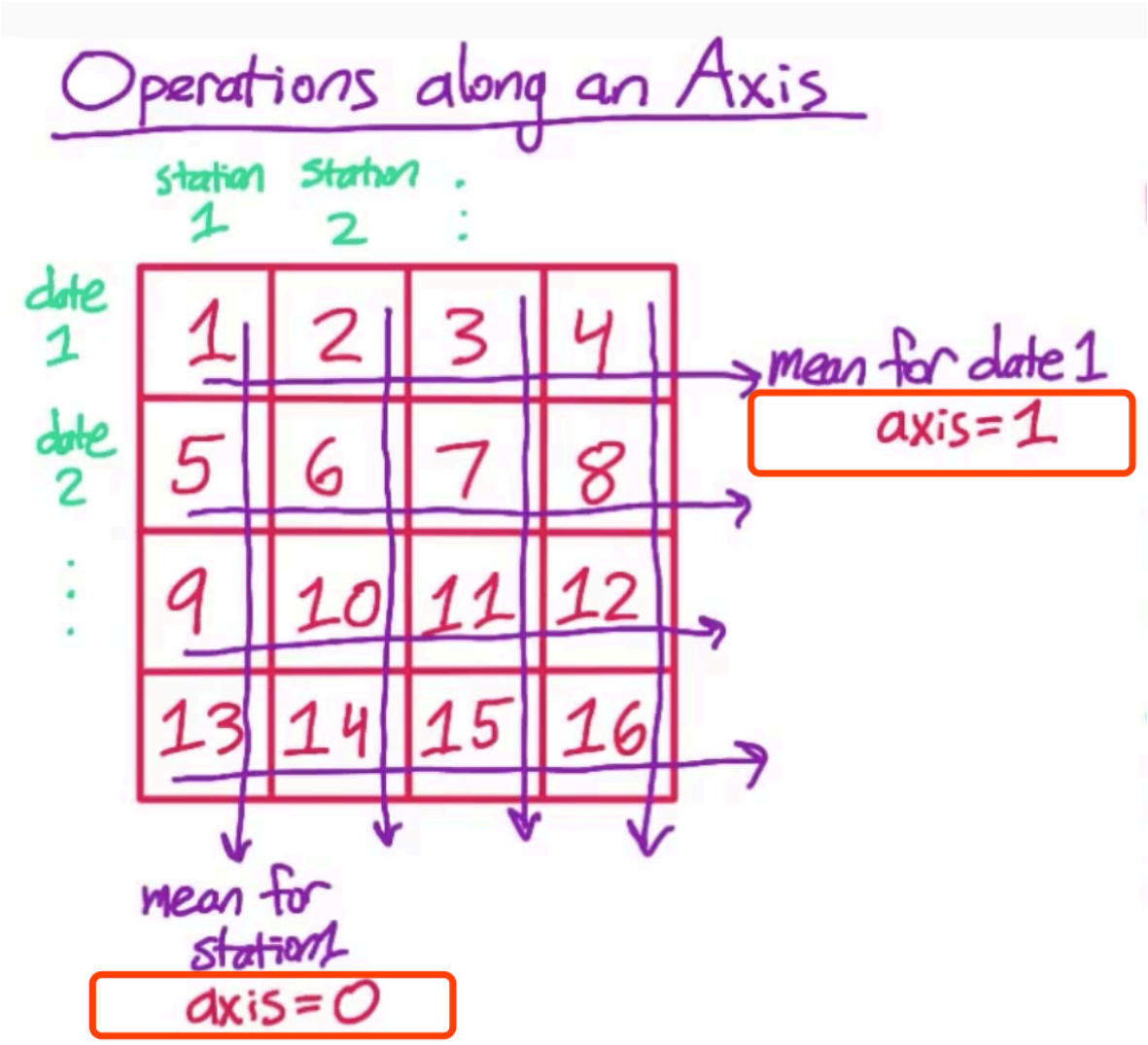
2D arrays, as opposed to array of arrays:

- More memory efficient
- Accessing elements is a bit different
  a[1,3] rather than a[1][3]
- mean(), std(), etc. operate on entire array



Write a function to:
1. Find the max rides on the first day
2. Find the mean rides per day

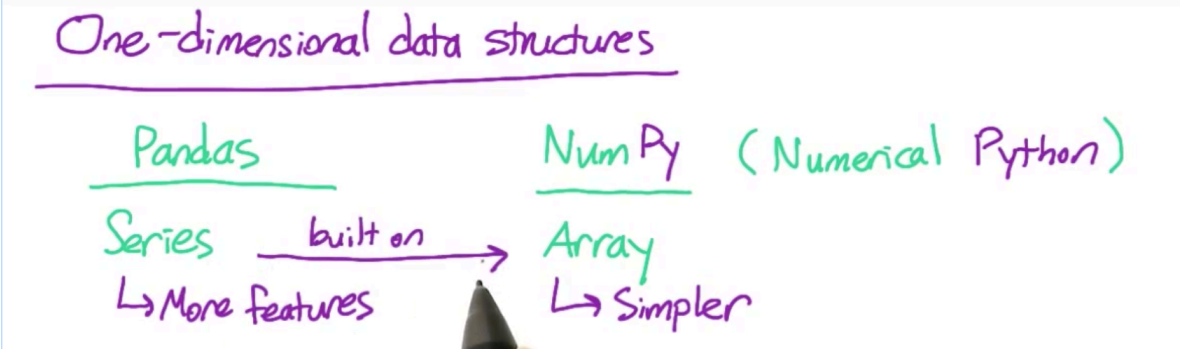
```
import numpy as np

# Subway ridership for 5 stations on 10 different days
ridership = np.array([
    [   0,    0,    2,    5,    0],
    [1478, 3877, 3674, 2328, 2539],
    [1613, 4088, 3991, 6461, 2691],
    [1560, 3392, 3826, 4787, 2613],
    [1608, 4802, 3932, 4477, 2705],
    [1576, 3933, 3909, 4979, 2685],
    [  95,  229,  255,  496,  201],
    [   2,    0,    1,   27,    0],
    [1438, 3785, 3589, 4174, 2215],
    [1342, 4043, 4009, 4665, 3033]
])

# Change False to True for each block of code to see what it does

# Accessing elements
if False:
    print ridership[1, 3]
    print ridership[1:3, 3:5]
    print ridership[1, :]
```



pandas部分

一维运算部分

pandas vs numpy



Pandas Series

A series is similar to a NumPy array, but with extra functionality

e.g. s.describe()

Similarities:

- Accessing elements s[0], s[3:7]
- Looping for x in s
- Convenient functions s.mean(), s.max()
- Vectorized Operations s1+s2
- Implemented in C (fast!)

Pandas Series

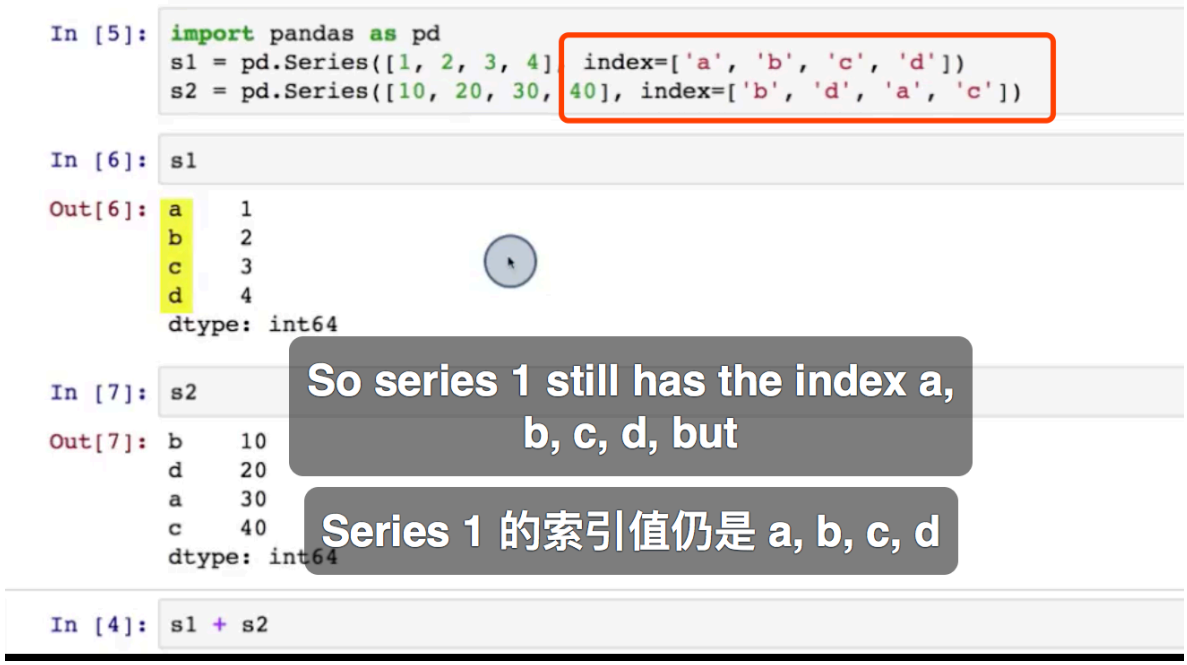
A series is similar to a NumPy array, but with extra functionality

e.g. s.describe()

Similarities:

- Accessing elements s[0], s[3:7]
- Looping for x in s
- Convenient functions s.mean(), s.max()
- Vectorized Operations s1+s2
- Implemented in C (fast!)

pandas series的向量运算按照索引进行匹配 而不是位置；无法匹配索引的元素，填充缺失值



pandas series apply函数 （返回值仍然是series）
想对series中每个元素执行一个非pandas内建函数时 可以使用apply函数

Non Built-In Calculations

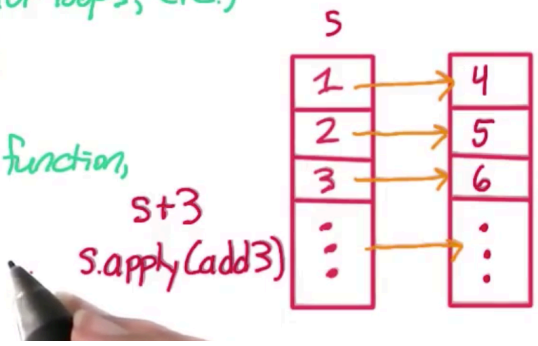
So far, you've used built-in functions (e.g. mean())
and operations (e.g. +)

1. Treat the series as a list (for loops, etc.)
2. Use the function apply()

apply() takes a series and a function,
and returns a new series

```
def add3(x):
    return x+3
```

s+3
s.apply(add3)



Non Built-In Calculations

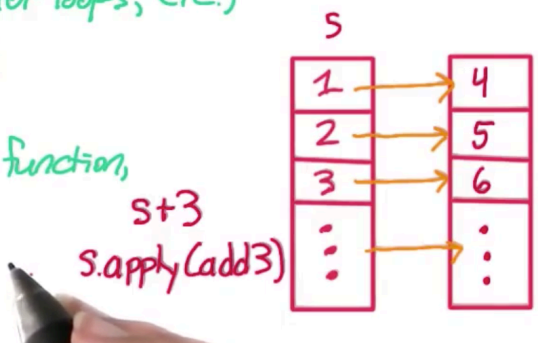
So far, you've used built-in functions (e.g. mean())
and operations (e.g. +)

1. Treat the series as a list (for loops, etc.)
2. Use the function apply()

apply() takes a series and a function,
and returns a new series

```
def add3(x):
    return x+3
```

s+3
s.apply(add3)



二维运算部分

```
ridership_df = pd.DataFrame({
    'R003': [0, 1478, 1613, 1560, 1608, 1576, 95, 2, 1438, 1342],
    'R004': [0, 3877, 4088, 3392, 4802, 3933, 229, 0, 3785, 4043],
    'R005': [2, 3674, 3991, 3826, 3932, 3909, 255, 1, 3589, 4009],
    'R006': [5, 2328, 6461, 4787, 4477, 4979, 496, 27, 4174, 4665],
    'R007': [0, 2539, 2691, 2613, 2705, 2685, 201, 0, 2215, 3033]
}, index=[
    '05-01-11', '05-02-11', '05-03-11', '05-04-11', '05-05-11',
    '05-06-11', '05-07-11', '05-08-11', '05-09-11', '05-10-11'
])
```

```
In [3]: ridership_df.loc['05-02-11']

Out[3]: R003    1478
        R004    3877
        R005    3674
        R006    2328
        R007    2539
```

```
In [4]: ridership_df.iloc[9]

Out[4]: R003    1342
        R004    4043
        R005    4009
        R006    4665
        R007    3033
        Name: 05-10-11, dtype: int64
```

Out[2]:

	R003	R004	R005	R006	R007
05-01-11	0	0	2	5	0
05-02-11	1478	3877	3674	2328	2539
05-03-11	1613	4088	3991	6461	2691
05-04-11	1560	3392	3826	4787	2613
05-05-11	1608	4802	3932	4477	2705
05-06-11	1576	3933	3909	4979	2685
05-07-11	95	229	255	496	201
05-08-11	2	0			
05-09-11					
05-10-11	1342	4043	4009	4665	3033

For example iloc at 0,
3 gives the element for row 0, column 3.

比如 iloc[0, 3] 就是第0行和第3列的元素

```
In [5]: ridership_df.iloc[0, 3]

Out[5]: 5
```

Out[2]:

	R003	R004	R005	R006	R007
05-01-11	0	0	2	5	0
05-02-11	1478	3877	3674	2328	2539
05-03-11	1613	4088	3991	6461	2691
05-04-11	1560	3392	3826	4787	2613
05-05-11	1608	4802	3932	4477	2705
05-06-11	1576	3933	3909	4979	2685
05-07-11	95	229	255	496	201
05-08-11	2	0	1	27	0
05-09-11	1438	3785	3589	4174	2215
05-10-11	1342	4043	4009	4665	3033

```
In [6]: ridership_df.loc['05-04-11', 'R004']
Out[6]: 3392
```

```
In [8]: ridership_df.values
Out[8]: array([[   0,    0,    2,    5,    0],
               [1478, 3877, 3674, 2328, 2539],
               [1613, 4088, 3991, 6461, 2691],
               [1560, 3392, 3826, 4787, 2613],
               [1576, 3933, 3909, 4979, 2685],
               [   2,    0,    1,   27,    0],
               [1438, 3785, 3589, 4174, 2215],
               [1342,
```

You can also use .values to get a 2D NumPy array containing only the values

你还可以利用 .values 得到

pandas加载csv

```
import pandas as pd

pd.read_csv('filename.csv')
```

```
subway_df.head()
```

	UNIT	DATEn	TIMEn	ENTRIESn	EXITSn	ENTRIESn_hourly	EXITSn_hourly	datetime
0	R003	05-01-11	00:00:00	4388333	2911002	0	0	2011-05-01 00:00:00
1	R003	05-01-11	04:00:00	4388333	2911002	0	0	2011-05-01 04:00:00
2	R003	05-01-11	12:00:00	4388333	2911002	0	0	2011-05-01 12:00:00
3	R003	05-01-11	16:00:00	4388333	2911002	0	0	2011-05-01 16:00:00
4	R003	05-01-11	20:00:00	4388333	2911002	0	0	2011-05-01 20:00:00

The head function returns a smaller data frame so

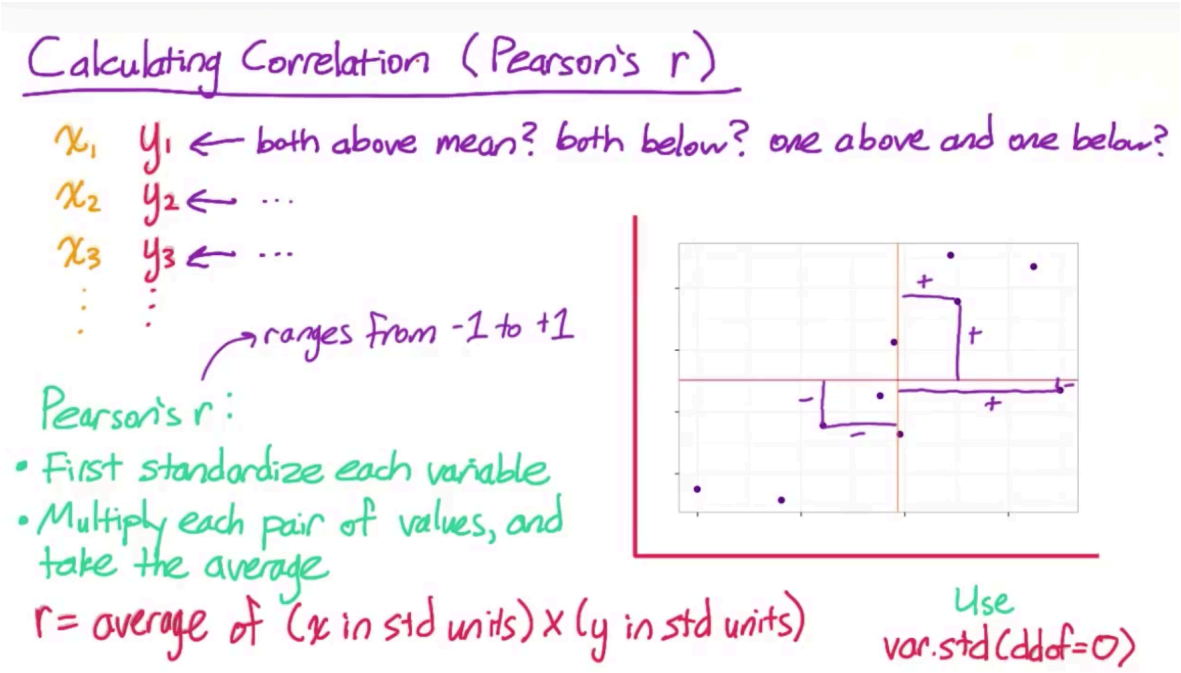
head 函数返回一个较小的数据框

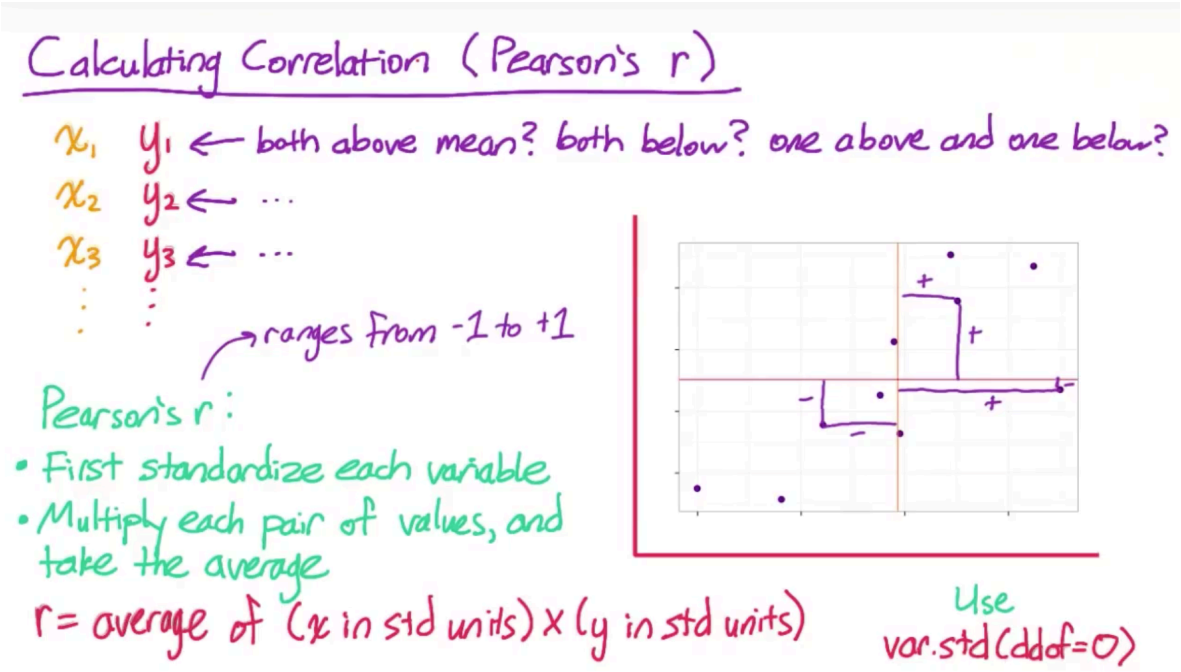
```
subway_df.describe()
```

	ENTRIESn	EXITSn	ENTRIESn_hourly	EXITSn_hourly	hour	day_
count	4.264900e+04	4.264900e+04	42649.000000	42649.000000	42649.000000	4264
mean	2.812486e+07	1.986993e+07	1886.589955	1361.487866	10.046754	2.905
std	3.043607e+07	2.028986e+07	2952.385585	2183.845409	6.938928	2.079
min	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000
25%	1.039762e+07	7.613712e+06	274.000000	237.000000	4.000000	1.000
50%	1.818389e+07	1.331609e+07	905.000000	664.000000	12.000000	3.000
75%	3.263049e+07	2.393771e+07	2255.000000	1537.000000	16.000000	5.000
max	2.357746e+08	1.493782e+08	32814.000000	34828.000000	20.000000	6.000

I can also use the describe function to see some statistics about each column

计算相关性时 (Pearson's r)





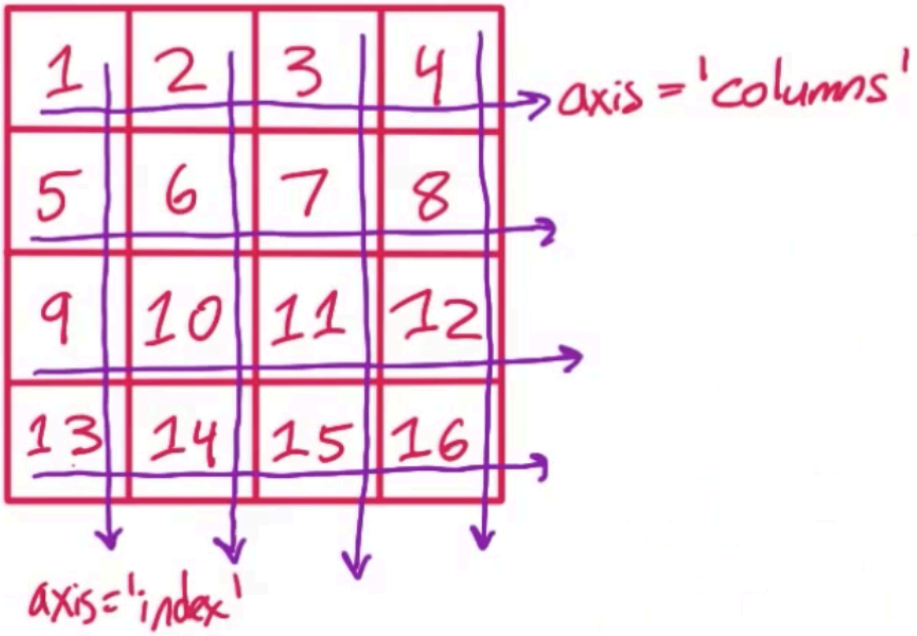
注意：为使计算能够正确进行，我们应该在 .std() 函数中将“ddof”参数的值设置为 0。

注意，计算得出的默认标准偏差类型在 numpy 的 .std() 和 pandas 的 .std() 函数之间是不同的。默认情况下，numpy 计算的是总体标准偏差，ddof = 0。另一方面，pandas 计算的是样本标准偏差，ddof = 1。如果我们知道所有的分数，那么我们就有了总体——因此，要使用 pandas 进行归一化处理，我们需要将“ddof”设置为 0。

pandas的轴名

Pandas Axis Names

Instead of axis=0 or axis=1, you can use axis='index' or axis='columns'




```
In [1]: import pandas as pd

        entries_and_exits = pd.DataFrame({
            'ENTRIESn': [3144312, 3144335, 3144353, 3144424, 3144594,
                         3144808, 3144895, 3144905, 3144941, 3145094],
            'EXITSn': [1088151, 1088159, 1088177, 1088231, 1088275,
                       1088317, 1088328, 1088331, 1088420, 1088753]
        })

In [2]: entries_and_exits.shift(1)
```

Out[2]:

	ENTRIESn	EXITSn
0	NaN	NaN
1	3144312	1088151
2	3144335	1088159
3	3144353	1088177
4	3144424	1088231

the third row corresponds to the second row, and so on.

第三行对应原数据框的第二行

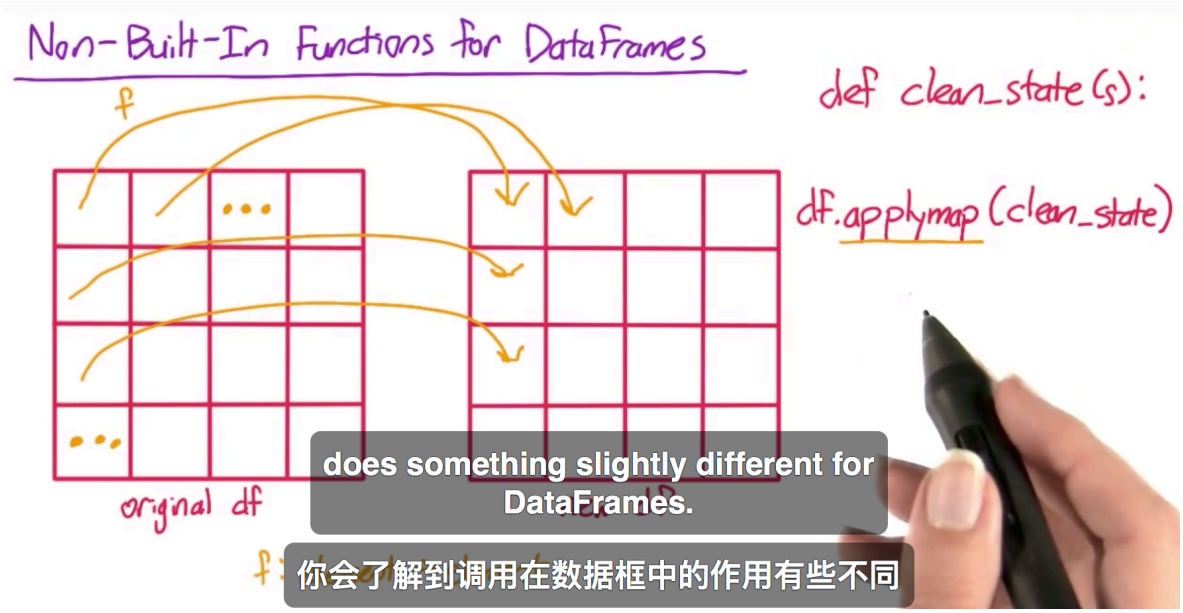
pandas.DataFrame.shift

`DataFrame.shift(periods=1, freq=None, axis=0)`

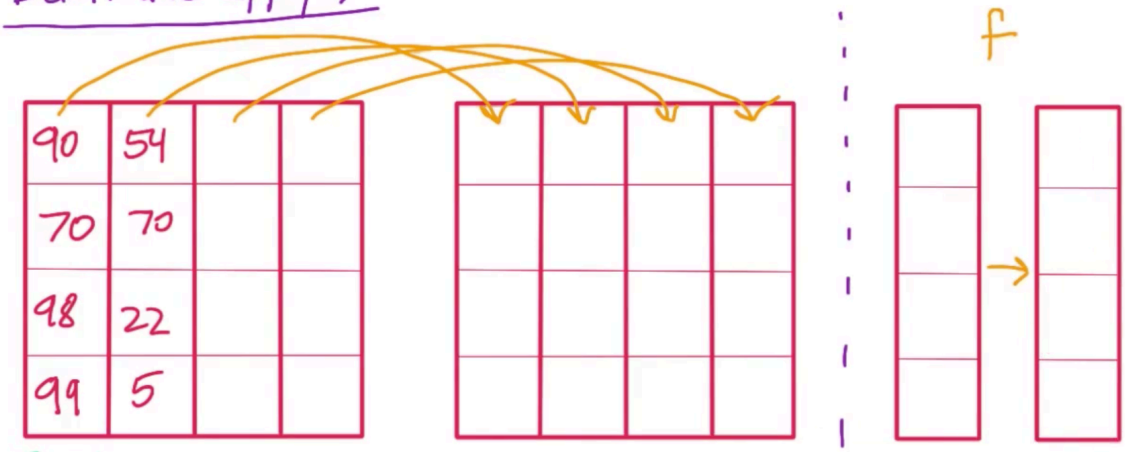
Shift index by desired number of periods with an optional time freq

Parameters:	**periods** : *int* Number of periods to move, can be positive or negative **freq** : *DateOffset, timedelta, or time rule string, optional* Increment to use from the tseries module or time rule (e.g. 'EOM'). See Notes. **axis** : *{0 or 'index', 1 or 'columns'}*
Returns:	**shifted** : *DataFrame*

DataFrame中的applymap（对df的每一个元素）和apply（对df的每一行|列（series））



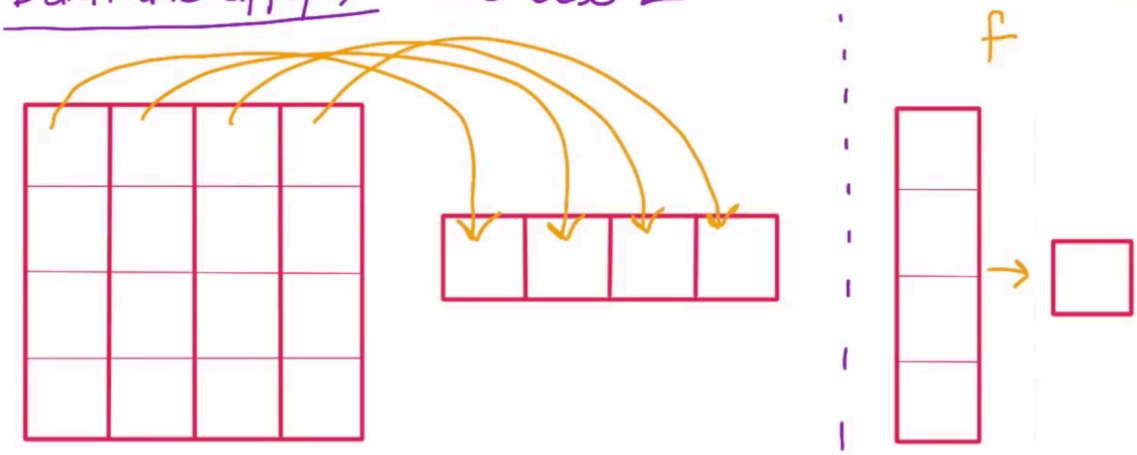
DataFrame apply()



Example:
Grading on a

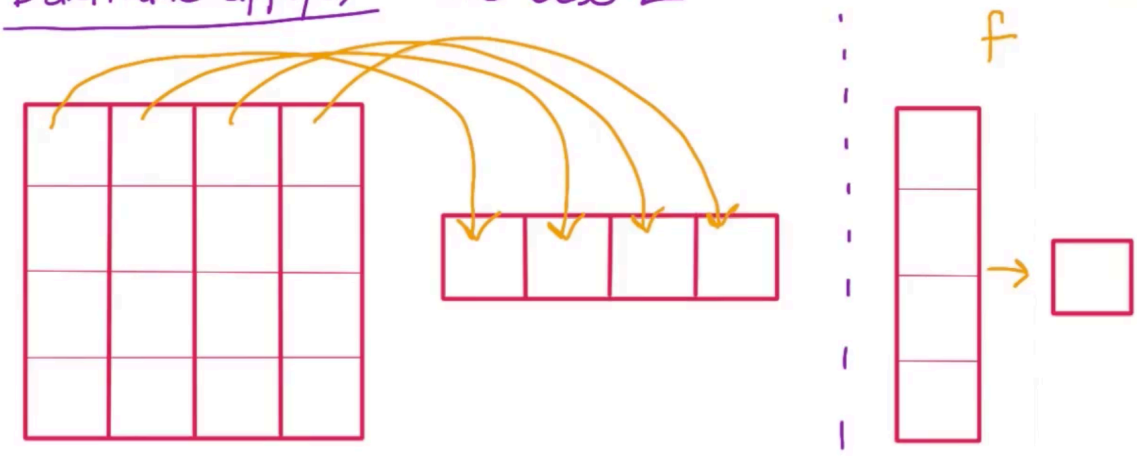
df.apply(f)

DataFrame apply() - Use Case 2

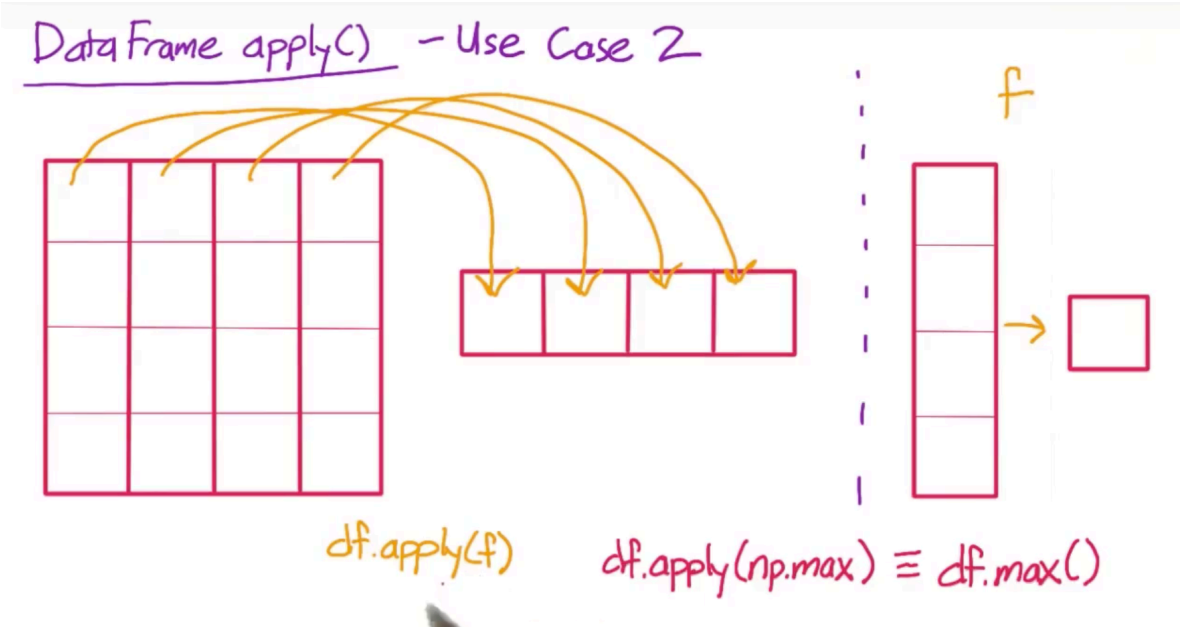


df.apply(f) df.apply(np.max) ≡ df.max()

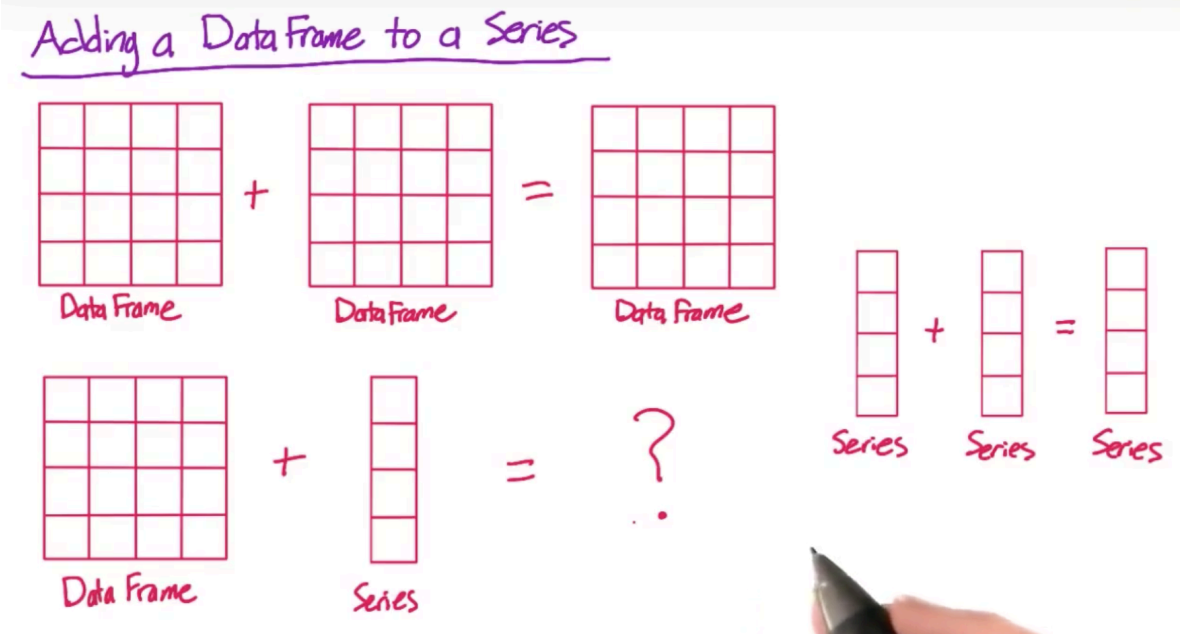
DataFrame apply() - Use Case 2



df.apply(f) df.apply(np.max) ≡ df.max()



DataFrame+Series时的运算




```
import pandas as pd

s = pd.Series([1, 2, 3, 4])

df = pd.DataFrame({
    0: [10, 20, 30, 40],
    1: [50, 60, 70, 80],
    2: [90, 100, 110, 120],
    3: [130, 140, 150, 160]
})
```

```
In [2]: df
```

Out[2]:

	0	1	2	3
0	10	50	90	130
1	20	60	100	140
2	30	70	110	150
3	40	80	120	160

```
In [3]: df+s
```

Out[3]:

	0	1	2	3
0	11	52	93	134
1	21	62	103	144
2	31	72	113	154
3	41	82	123	164

It looks like the value 1 was added to each value in the first column,

可以看到 第一列的每个值都加了 1

DataFrame的GroupBy 返回DataFrameGroupBy对象
groupby对象的内置函数（如sum等）详见
http://pandas.pydata.org/pandas-docs/stable/api.html#groupby

```
import pandas as pd

engagement_df = pd.DataFrame({
    'account_key': ['1200', '1200', '1200', '1200', '1200',
                    '1200', '1200', '1175', '1175', '1175',
                    '1175', '1175', '1175', '1175'],
    'utc_date': ['2015-03-04', '2015-03-05', '2015-03-06', '2015-03-07',
                 '2015-03-08', '2015-03-09', '2015-03-10', '2015-04-02',
                 '2015-04-03', '2015-04-04', '2015-04-05', '2015-04-06',
                 '2015-04-07', '2015-04-08'],
    'total_minutes_visited': [114.9, 43.4, 187.8, 150.1, 191.6, 0, 8.8,
                              2.7, 0, 0, 0, 0, 0, 0]
})
```

Then the code I would write to get

```
In [2]: engagement_df.groupby('account_key').sum()['total_minutes_visited'].mean()
Out[2]: 349.65000000000003
```

合并两个DataFrame 使用merge函数
http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html?highlight=merge#pandas.DataFrame.merge

pandas.DataFrame.merge

`DataFrame.merge(right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=True, indicator=False, validate=None)` ¶ [source]

Merge DataFrame objects by performing a database-style join operation by columns or indexes.