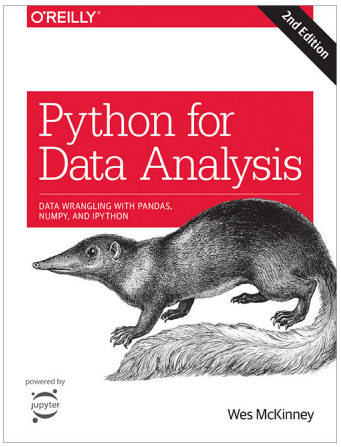


En repartant du dataset "people.csv" (cf. lesson4) dans sa version "clean" finale:

- mettre la colonne `inactive` à `true` pour tous les users dont le `last_seen` date d'au moins un an
  import time
  people.last_seen =
  pd.to_datetime(people.last_seen,unit='s')
  df.clean.last_seen < '2018-9-30
  people[people.last_seen < '2019-08-04']
- avec une regex: filtrer les numéros de téléphone invalides
  people[people.phone.str.contains(' ')] ???
- ajouter une colonne indiquant si le numéro de tel correspond à un téléphone portable (06/07)
- ajouter une colonne indiquant si les coordonnées GPS de l'utilisateur correspondent bien au "country"
  user API

Reshaping:
Concater : ----- + si deux dataset ont la meme numbre de ligne
dfa + dfb ----- padans additioner selon index 0,1,2,3,4
(NaN + numbre ---> NaN)

dfa.**set_index('date')** + dfb.set_index('date') -------additioner selon index date
dfa.**set_index('date').add(** dfb.set_index('date') , fill_value=)

dfb.set_index('date')**.reindex**(dfa.index).fillna(0) -------- use dfb index on dfa, change to same shape

pd.**concat**([dfa,dfb])
pd.concat([dfa,dfb],axis=1)
pd.concat([**dfa.set_index('date')** , **dfb.set_index('date')**],axis=1,sort=False)

Melt (wide to long)
df.melt(id_vars='indicateur')
df.setindex('indicateur').T ------ same Transposer
df.setindex('indicateur').unstack ? ----- plusieur niveau d'index

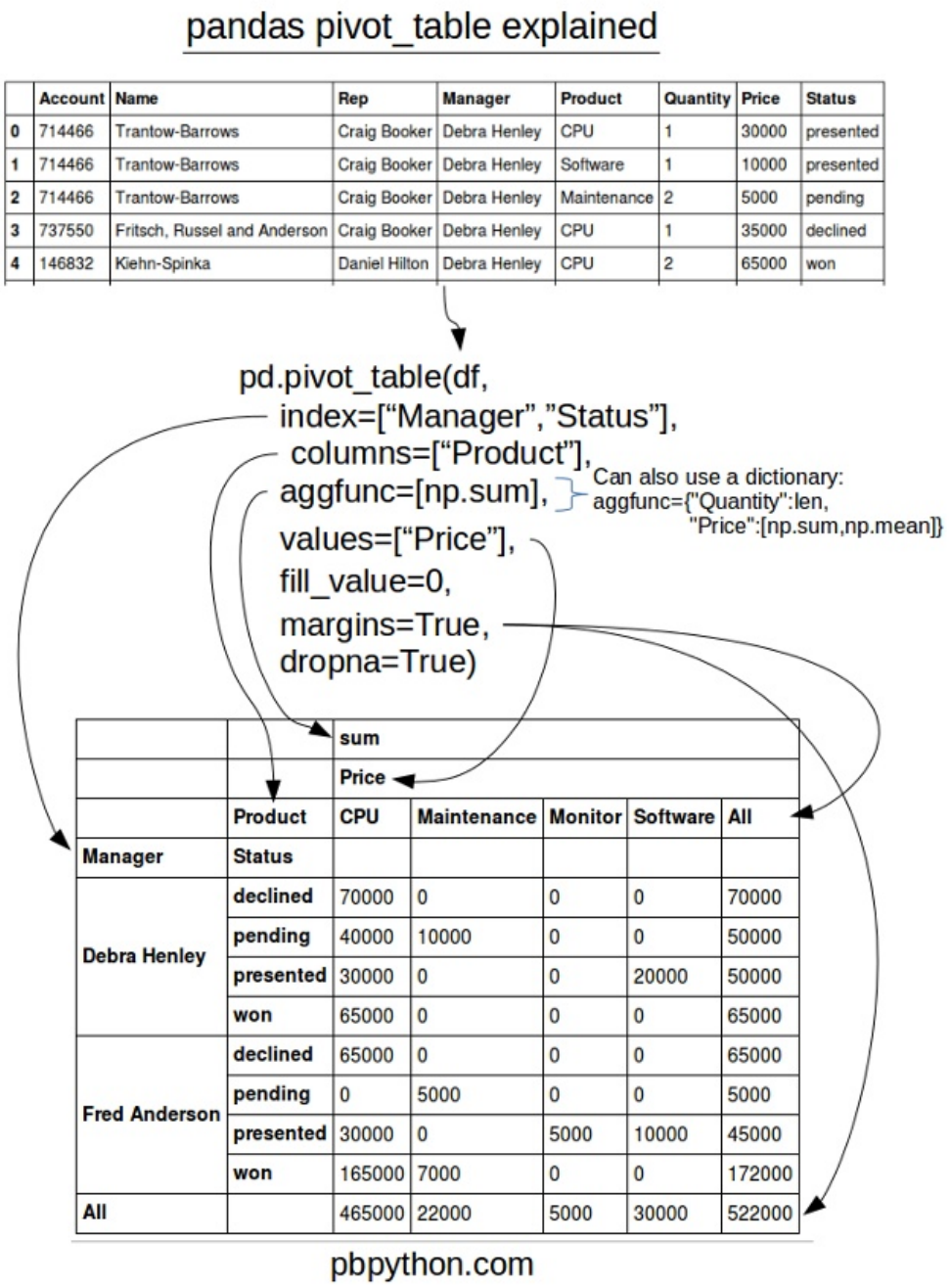
Pivote(long to wide)
london
london
Paris
Paris

temp.pivot()
pandas.pivot_table(*data, values=None, index=None, columns=None, aggfunc="mean", fill_value=None, margins=False, dropna=True, margins_name="All", observed=False*)[source]

Create a spreadsheet-style pivot table as a DataFrame. The levels in the pivot table will be stored in MultiIndex objects (hierarchical indexes) on the index and

columns of the result DataFrame.

temp.pivot_table(XXXXXXXXXX*indexd*、*values*、*columns*、*aggfunc*)



	Account	Name	Rep	Manager	Product	Quantity	Price	Status
0	714466	Trantow-Barrows	Craig Booker	Debra Henley	CPU	1	30000	presented
1	714466	Trantow-Barrows	Craig Booker	Debra Henley	Software	1	10000	presented
2	714466	Trantow-Barrows	Craig Booker	Debra Henley	Maintenance	2	5000	pending
3	737550	Fritsch, Russel and Anderson	Craig Booker	Debra Henley	CPU	1	35000	declined
4	146832	Kiehn-Spinka	Daniel Hilton	Debra Henley	CPU	2	65000	won

		sum				
		Price				
	Product	CPU	Maintenance	Monitor	Software	All
Manager	Status					
Debra Henley	declined	70000	0	0	0	70000
	pending	40000	10000	0	0	50000
	presented	30000	0	0	20000	50000
	won	65000	0	0	0	65000
Fred Anderson	declined	65000	0	0	0	65000
	pending	0	5000	0	0	5000
	presented	30000	0	5000	10000	45000
	won	165000	7000	0	0	172000
All		465000	22000	5000	30000	522000

Merge(need same column in the 2 df)/join
beer.merge(price, left_on='', right_on='product_name',how='outer/left/right**')**
df.merge(data.col.rename[])
temperature.merge(tmin, **on=['date','city'])** --------- explicitly define the columns on which we merge, otherwise pandas will merge on all the common columns(name)
left_index = True , right_index ------ merge can do as join with left_index,right_index
tomorrow = todayDate + **pd.timedelta(days=1, seconds , microseconds ,)** ??? ------ 当前时间加一天

Function application, GroupBy & window

DataFrame.apply(self, func[, axis, …])	Apply a function along an axis of the DataFrame.
DataFrame.applymap(self, func)	Apply a function to a Dataframe elementwise.
DataFrame.pipe(self, func, *args, **kwargs)	Apply func(self, *args, **kwargs).

DataFrame.agg(self, func[, axis])	Aggregate using one or more operations over the specified axis.
DataFrame.aggregate(self, func[, axis])	Aggregate using one or more operations over the specified axis.
DataFrame.transform(self, func[, axis])	Call func on self producing a DataFrame with transformed values and that has the same axis length as self.
DataFrame.groupby(self[, by, axis, level, …])	Group DataFrame or Series using a mapper or by a Series of columns.
DataFrame.rolling(self, window[, …])	Provide rolling window calculations.
DataFrame.expanding(self[, min_periods, …])	Provide expanding transformations.
DataFrame.ewm(self[, com, span, halflife, …])	Provide exponential weighted functions.

- **pandas.DataFrame.groupby** (Python method, in pandas.DataFrame.groupby)
- **pandas.Index.groupby** (Python method, in pandas.Index.groupby)
- **pandas.Series.groupby** (Python method, in pandas.Series.groupby)

GroupBy ---后面可以自带各种功能
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html?highlight=groupby#pandas.DataFrame.groupby
df.groupby('keyname').sum() ------ type: groupby
df.groupby('keyname').get_group('A')
for groupby(['','','','','']).sum()
df.groupby(['country','devise']).sum() ----- -避免不同货币单位加在一起
df.country.unique() ------ 列出不同国家
country_map = {'France':'south' , Germany : North}
.agg([np.sum,np.mean])
.agg(['sum','mean','median']) ------ pandas自动猜出函数
。agg (自定义函数) ---- 降低数据到一维
。apply (自定义函数) ---
.mean ( ) ---计算均值
.cumsum ---------Cumulative sum for each group.

GroupBy

GroupBy objects are returned by groupby calls: **pandas.DataFrame.groupby()**, **pandas.Series.groupby()**, etc.

Indexing, iteration

GroupBy.__iter__(self)	Groupby iterator.
GroupBy.groups	Dict {group name -> group labels}.
GroupBy.indices	Dict {group name -> group indices}.
GroupBy.get_group(self, name[, obj])	Construct DataFrame from group with provided name.

Grouper([key, level, freq, axis, sort])	A Grouper allows the user to specify a groupby instruction for a target object

Function application

GroupBy.apply(self, func, *args, **kwargs)	Apply function func group-wise and

GroupBy.apply(self, func, *args, **kwargs)	combine the results together.
GroupBy.agg(self, func, *args, **kwargs)	
GroupBy.aggregate(self, func, *args, **kwargs)	
GroupBy.transform(self, func, *args, **kwargs)	
GroupBy.pipe(self, func, *args, **kwargs)	Apply a function func with arguments to this GroupBy object and return the function's result.

Computations / descriptive stats

GroupBy.all(self[, skipna])	Return True if all values in the group are truthful, else False.
GroupBy.any(self[, skipna])	Return True if any value in the group is truthful, else False.
GroupBy.bfill(self[, limit])	Backward fill the values.
GroupBy.count(self)	Compute count of group, excluding missing values.
GroupBy.cumcount(self[, ascending])	Number each item in each group from 0 to the length of that group - 1.
GroupBy.cummax(self[, axis])	Cumulative max for each group.
GroupBy.cummin(self[, axis])	Cumulative min for each group.
GroupBy.cumprod(self[, axis])	Cumulative product for each group.
GroupBy.cumsum(self[, axis])	Cumulative sum for each group.
GroupBy.ffill(self[, limit])	Forward fill the values.
GroupBy.first(self, **kwargs)	Compute first of group values.
GroupBy.head(self[, n])	Return first n rows of each group.
GroupBy.last(self, **kwargs)	Compute last of group values.
GroupBy.max(self, **kwargs)	Compute max of group values.
GroupBy.mean(self, *args, **kwargs)	Compute mean of groups, excluding missing values.
GroupBy.median(self, **kwargs)	Compute median of groups, excluding missing values.
GroupBy.min(self, **kwargs)	Compute min of group values.
GroupBy.ngroup(self[, ascending])	Number each group from 0 to the number of groups - 1.
GroupBy.nth(self, n, List[int]], dropna, …)	Take the nth row from each group if n is an int, or a subset of rows if n is a list of ints.
GroupBy.ohlc(self)	Compute sum of values, excluding missing values.
GroupBy.prod(self, **kwargs)	Compute prod of group values.
GroupBy.rank(self[, method, ascending, …])	Provide the rank of values within each group.
GroupBy.pct_change(self[, periods, …])	Calculate pct_change of each value to previous entry in group.
GroupBy.size(self)	Compute group sizes.
GroupBy.sem(self[, ddof])	Compute standard error of the mean of groups, excluding missing values.
GroupBy.std(self[, ddof])	Compute standard deviation of groups, excluding missing values.
GroupBy.sum(self, **kwargs)	Compute sum of group values.
GroupBy.var(self[, ddof])	Compute variance of groups, excluding

GroupBy.var(self[, ddof])	missing values.
GroupBy.tail(self[, n])	Return last n rows of each group.

The following methods are available in both SeriesGroupBy and DataFrameGroupBy objects, but may differ slightly, usually in that the DataFrameGroupBy version usually permits the specification of an axis argument, and often an argument indicating whether to restrict application to columns of a specific data type.

DataFrameGroupBy.all(self[, skipna])	Return True if all values in the group are truthful, else False.
DataFrameGroupBy.any(self[, skipna])	Return True if any value in the group is truthful, else False.
DataFrameGroupBy.bfill(self[, limit])	Backward fill the values.
DataFrameGroupBy.corr	Compute pairwise correlation of columns, excluding NA/null values.
DataFrameGroupBy.count(self)	Compute count of group, excluding missing values.
DataFrameGroupBy.cov	Compute pairwise covariance of columns, excluding NA/null values.
DataFrameGroupBy.cummax(self[, axis])	Cumulative max for each group.
DataFrameGroupBy.cummin(self[, axis])	Cumulative min for each group.
DataFrameGroupBy.cumprod(self[, axis])	Cumulative product for each group.
DataFrameGroupBy.cumsum(self[, axis])	Cumulative sum for each group.
DataFrameGroupBy.describe(self, **kwargs)	Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.
DataFrameGroupBy.diff	First discrete difference of element.
DataFrameGroupBy.ffill(self[, limit])	Forward fill the values.
DataFrameGroupBy.fillna	Fill NA/NaN values using the specified method.
DataFrameGroupBy.filter(self, func[, dropna])	Return a copy of a DataFrame excluding elements from groups that do not satisfy the boolean criterion specified by func.
DataFrameGroupBy.hist	Make a histogram of the DataFrame's.
DataFrameGroupBy.idxmax	Return index of first occurrence of maximum over requested axis.
DataFrameGroupBy.idxmin	Return index of first occurrence of minimum over requested axis.
DataFrameGroupBy.mad	Return the mean absolute deviation of the values for the requested axis.
DataFrameGroupBy.nunique(self[, dropna])	Return DataFrame with number of distinct observations per group for each column.
DataFrameGroupBy.pct_change(self[, periods, …])	Calculate pct_change of each value to previous entry in group.
DataFrameGroupBy.plot	Class implementing the .plot attribute for groupby objects.

DataFrameGroupBy.quantile(self[, q, …])	Return group values at the given quantile, a la numpy.percentile.
DataFrameGroupBy.rank(self[, method, …])	Provide the rank of values within each group.
DataFrameGroupBy.resample(self, rule, …)	Provide resampling when using a TimeGrouper.
DataFrameGroupBy.shift(self[, periods, …])	Shift each group by periods observations.
DataFrameGroupBy.size(self)	Compute group sizes.
DataFrameGroupBy.skew	Return unbiased skew over requested axis Normalized by N-1.
DataFrameGroupBy.take	Return the elements in the given *positional* indices along an axis.
DataFrameGroupBy.tshift	Shift the time index, using the index's frequency if available.

Transform(sum())

Stack/unstack —- MultiIndex —- Multidimensional datas

Time Series:
生成日期范围使用date_range函数

```
index = pd.date_range('4/1/2012','6/1/2012') index #输出 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', "2012-05-29', '2012-05-30', '2012-05-31', '2012-06-01'], dtype='datetime64[ns]', freq='D')
```

默认情况下，date_range会产生按天计算的时间点，如果只传入起始或结束日期，那就还得传入一个表示一段时间的数字:

pd.date_range(start='4/1/2012',periods=20)

如果你不想按天生成数据，想要按照一定的频率生成，我们传入freq参数即可.如想按5小时生成数据:

```
pd.date_range(end='4/1/2012',periods=20,freq='5H')
```

如果你想生成一个由每月最后一个工作日组成的日期索引，可以使用BM频率:

```
pd.date_range('1/1/2000','12/1/2000',freq='BM')
```

pandas中的时间序列一般被认为是不规则的，也就是说，没有固定的频率，对于大部分程序而言，这是无所谓的，但是，他常常需要以某种相对固定的频率进行分析，比如每月，每日，每15min等。pandas有一套标准时间序列频率以及用于重采样，频率推断，生成固定频率日期范围的工具.

例如，我们可以将之前的时间序列转换为一个具有固定频率（每日）的时间序列，只需调用resample即可.返回DatetimeIndexResampler，获取值使用asfreq():

```
ts1 = ts.resample('D').asfreq() / .mean() /agg(mean(),std())
ts1
#输出
2011-01-02   -0.881964
2011-01-03         NaN
2011-01-04         NaN
2011-01-05   -0.554943
2011-01-06         NaN
2011-01-07   -1.111905
```

2011-01-07 -1.111905

df.reindex(pd.date_range(‘’)

别名	偏移量类型	说明
D	Day	每日历日
B	BusinessDay	每工作日
H	Hour	每小时
T/min	Minute	每分
S	Second	每秒
L/ms	Million	每毫秒
U	Micro	每微妙
M	MonthEnd	每月最后一个日历日
BM	BusinessMonthEnd	每月最后一个工作日
MS	MonthBegin	每月第一个日历日
BMS	BusinessMonthBegin	每月第一个工作日
W-MON、W-TUE…	Week	从指定的星期几开始算起，每周
WOM-1MON、WOM-2MON…	WeekOfMonth	产生每月第一、二、三、四周的星期几，例如WOM-1MON表示每月的第一个星期一
Q-JAN、Q-FEB…	QuarterEnd	对于以指定月份（JAN、FEB、…、DEC）结束的年度，每季度的最后一月的最后一个日历日
BQ-JAN、BQ-FEB…	BusinessQuarterEnd	对于以指定月份（JAN、FEB、…、DEC）结束的年度，每季度的最后一月的最后一个工作日
QS-JAN、QS-FEB…	QuarterBegin	对于以指定月份（JAN、FEB、…、DEC）结束的年度，每季度的最后一月的第一个日历日
BQS-JAN、BQS-FEB…	BusinessQuarterBegin	对于以指定月份（JAN、FEB、…、DEC）结束的年度，每季度的最后一月的第一个工作日
A-JAN、A-FEB…	YearEnd	每年指定月份最后一个日历日
BA-JAN、BA-FEB…	BusinessYearEnd	每年指定月份最后一个工作日
AS-JAN、AS-FEB…	YearBegin	每月指定月份第一个日历日
BAS-JAN、BAS-FEB…	BusinessYearBegin	每月指定月份第一个工作日

————————————————

移动数据shift()

shift()是指在索引不变的条件下，沿着时间轴将数据迁移或者后移。通常用于计算时间序列中百分比的变化，即：ts/ts.shift(1)-1

i）索引保持不动，将数据前移或后移————shift

如果在shift()方法中不指定频率freq，索引会保持不动，数据则会前移或后移

```
date_range(start=None, end=None, periods=None, freq=None, tz=None(Time Zone), normalize=False,
           name=None, closed=None, **kwargs)
```

i）Period对象转换成别的频率————asfreq

把低频率转化为高频率需要指定how

```
In [2]: p = pd.Period('2018',freq='A-DEC')

In [3]: p
Out[3]: Period('2018', 'A-DEC')

In [4]: p.asfreq('M',how='start')
Out[4]: Period('2018-01', 'M')

In [5]: p.asfreq('M',how='end')
```

```
Out[5]: Period('2018-12', 'M')
```

把年转化为月

VIII. 重采样（resampling）

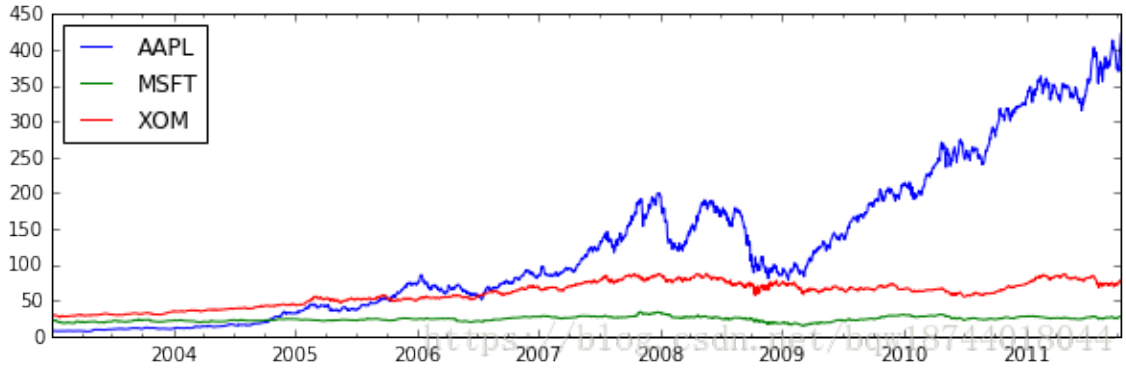
(下面这段重采样的定义文字来自 Wes McKinney 《利用Python进行数据分析》)

重采样（resampling）指的是将时间序列从一个频率转换到另一个频率的处理过程。将高频率数据聚合到低频率称为降采样（downsampling），而将低频率数据转换到高频率则称为升采样（upsampling）。并不是所有的重采样都能被划分到这两个大类中。例如，将W-WED（每周三）转换到W-FRI（每周五）既不是降采样也不是升采样。

Visualisr Time Series

一、直接使用plot进行绘图

```
df.loc['2011-01':'2011-03'].plot(figsize=(10,3)) # 整个DataFrame
```




```
appl_q = close_px['AAPL'].resample('Q-DEC').ffill() # 按季度进行重采样(聚集)
```

df.agg.plot[y='mean']

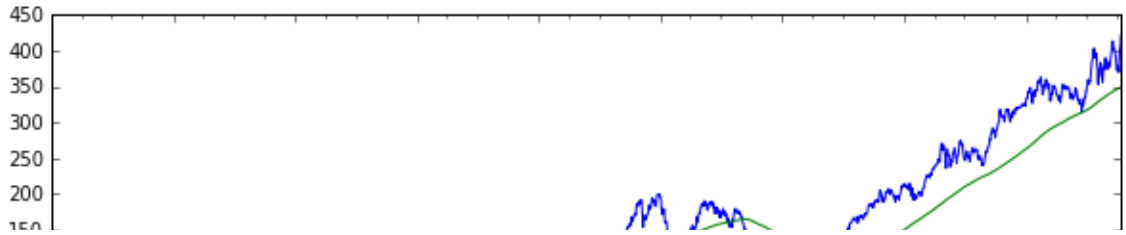
fill_between (fill with color)

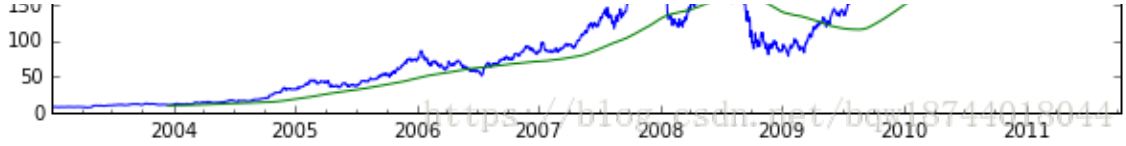
二、移动窗口函数

- **rolling方法，移动窗口函数，其中参数window指定窗口的大小；**
- **rolling方法后，可以接mean、count、sum、max、min、median、std等聚合函数；**

close_px['AAPL'].plot(figsize=(10,3))

close_px['AAPL'].rolling(window=250).mean().plot()






```
close_px['AAPL'].plot(figsize=(10,3))
# min_periods指窗口中非NA值至少要有10个
close_px['AAPL'].rolling(window=250,min_periods=10).std().plot()
```

Maplotlib — 设置不同的颜色都困难
%Maplotlib
Seanborn