十分钟入门pandas

本节代码来自: **黄海广-机器学习** https://github.com/fengdu78/WZU-machine-learning-course 推荐自学

Pandas是基于Numpy构建的，让Numpy为中心的应用变得更加简单。

In [1]:
```
#coding:utf8
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

这个一篇针对pandas新手的简短入门，想要了解更多复杂的内容，参阅*Cookbook*

通常，我们首先要导入以下几个库：

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

创建对象

通过传递一个list来创建**Series**，pandas会默认创建整型索引：

In [3]:
```
s = pd.Series([1,3,5,np.nan,6,8])
s
```

Out[3]:
```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

通过传递一个numpy array，日期索引以及列标签来创建一个**DataFrame**：

In [4]:
```
dates = pd.date_range('20230101', periods=6)
dates
```

Out[4]:
```
DatetimeIndex(['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04',
               '2023-01-05', '2023-01-06'],
              dtype='datetime64[ns]', freq='D')
```

In [5]:
```
df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
df
```

Out[5]:

	A	B	C	D
2023-01-01	-0.474649	-1.033131	-0.221292	1.123945
2023-01-02	-1.862598	-0.608146	-0.382219	2.711038
2023-01-03	0.430390	0.001635	-0.626828	0.647631
2023-01-04	2.206153	-0.095317	0.193339	-1.389618
2023-01-05	-0.661550	0.019897	1.039454	0.139001
2023-01-06	-1.651088	0.304878	0.532442	0.649129

通过传递一个能够被转换为类似series的dict对象来创建一个**DataFrame**:

In [6]:
```
df2 = pd.DataFrame({ 'A' : 1.,
                     'B' : pd.Timestamp('20230102'),
                     'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
                     'D' : np.array([3]*4,dtype='int32'),
                     'E' : pd.Categorical(["test","train","test","train"]),
                     'F' : 'foo' })
df2
```

Out[6]:

	A	B	C	D	E	F
0	1.0	2023-01-02	1.0	3	test	foo
1	1.0	2023-01-02	1.0	3	train	foo
2	1.0	2023-01-02	1.0	3	test	foo
3	1.0	2023-01-02	1.0	3	train	foo

可以看到各列的数据类型为：

In [7]:
```
df2.dtypes
```

Out[7]:
```
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

查看数据

查看frame中头部和尾部的几行：

In [8]:
```
df.head()
```

Out[8]:

	A	B	C	D
2023-01-01	-0.474649	-1.033131	-0.221292	1.123945
2023-01-02	-1.862598	-0.608146	-0.382219	2.711038
2023-01-03	0.430390	0.001635	-0.626828	0.647631
2023-01-04	2.206153	-0.095317	0.193339	-1.389618
2023-01-05	-0.661550	0.019897	1.039454	0.139001

In [9]:
```
df.tail(3)
```

Out[9]:

	A	B	C	D
2023-01-04	2.206153	-0.095317	0.193339	-1.389618
2023-01-05	-0.661550	0.019897	1.039454	0.139001
2023-01-06	-1.651088	0.304878	0.532442	0.649129

显示索引、列名以及底层的numpy数据

In [10]:
```
df.index
```

Out[10]:
```
DatetimeIndex(['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04',
               '2023-01-05', '2023-01-06'],
              dtype='datetime64[ns]', freq='D')
```

In [11]:
```
df.columns
```

Out[11]:
```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

In [ ]:
```
df.values
```

describe()能对数据做一个快速统计汇总

In [ ]:
```
df.describe()
```

对数据做转置:

In [ ]:
```
df.T
```

按轴进行排序:

In [ ]:
```
df.sort_index(axis=1, ascending=False)
```

按值进行排序 :

In [ ]:
```
df.sort_values(by='B')
```

数据选择

标准的Python/Numpy的表达式能完成选择与赋值等功能,

不推荐使用优化过的pandas数据访问方法：.at，.iat，.loc，.iloc和.i

尽管有性能上的优化, 但是对初学者不友好.

选取

选择某一列数据，它会返回一个**Series**，等同于**df.A**：

In [17]:
```
df['A']
```

Out[17]:
```
2023-01-01   -0.474649
2023-01-02   -1.862598
2023-01-03    0.430390
2023-01-04    2.206153
2023-01-05   -0.661550
2023-01-06   -1.651088
Freq: D, Name: A, dtype: float64
```

通过使用**[ ]**进行切片选取：

In [18]:
```
df[0:3]
```

Out[18]:

	A	B	C	D
2023-01-01	-0.474649	-1.033131	-0.221292	1.123945
2023-01-02	-1.862598	-0.608146	-0.382219	2.711038
2023-01-03	0.430390	0.001635	-0.626828	0.647631

In [19]:
```
df['20230102':'20230104']
```

Out[19]:

	A	B	C	D
2023-01-02	-1.862598	-0.608146	-0.382219	2.711038
2023-01-03	0.430390	0.001635	-0.626828	0.647631
2023-01-04	2.206153	-0.095317	0.193339	-1.389618

布尔索引

用某列的值来选取数据

In [20]:
```
df[df.A > 0]
```

Out[20]:

	A	B	C	D
2023-01-03	0.430390	0.001635	-0.626828	0.647631
2023-01-04	2.206153	-0.095317	0.193339	-1.389618

用**where**操作来选取数据

In [ ]:
```
df[df > 0]
```

用**isin()**方法来过滤数据

In [22]:
```
df2 = df.copy()
```

In [ ]:
```
df2['E'] = ['one', 'one', 'two', 'three', 'four', 'three']
df2
```

In [24]:
```
df2[df2['E'].isin(['two', 'four'])]
```

Out[24]:

	A	B	C	D	E
2023-01-03	0.43039	0.001635	-0.626828	0.647631	two
2023-01-05	-0.66155	0.019897	1.039454	0.139001	four

赋值

赋值一个新的列，通过索引来自动对齐数据

In [25]:
```
s1 = pd.Series([1,2,3,4,5,6], index=pd.date_range('20230102',periods=6))
s1
```

Out[25]:
```
2023-01-02    1
2023-01-03    2
2023-01-04    3
2023-01-05    4
2023-01-06    5
2023-01-07    6
Freq: D, dtype: int64
```

In [ ]:
```
df['F'] = s1
df
```

通过标签赋值

In [ ]:
```
df['A'] = 0
df
```

通过位置赋值

In [ ]:
```
df["B"][0] = 0
df
```

通过传递numpy array赋值

In [ ]:
```
df['D'] = np.array([5] * len(df))
df
```

通过**where**操作来赋值

In [ ]:
```
df2 = df.copy()
df2[df2 > 0] = -df2
df2
```

缺失值处理

在pandas中，用**np.nan**来代表缺失值，这些值默认不会参与运算。

reindex()允许你修改、增加、删除指定轴上的索引，并返回一个数据副本。

In [ ]:
```
df1 = df.reindex(index=dates[0:4], columns=list(df.columns)+['E'])
df1["E"][0:2] = 1
df1
```

剔除所有包含缺失值的行数据

In [32]:
```
df1.dropna(how='any')
```

Out[32]:

	A	B	C	D	F	E
2023-01-02	0	-0.608146	-0.382219	5	1.0	1.0

填充缺失值

In [ ]:
```
df1.fillna(value=5)
```

获取值是否为**nan**的布尔标记

In [ ]:
```
pd.isnull(df1)
```

运算

统计

运算过程中，通常不包含缺失值。

进行描述性统计

In [35]:
```
df.mean()
```

Out[35]:
```
A    0.000000
B   -0.062842
C    0.089149
D    5.000000
F    3.000000
dtype: float64
```

对其他轴进行同样的运算

In [ ]:
```
df.mean(1)
```

对于拥有不同维度的对象进行运算时需要对齐。除此之外，pandas会自动沿着指定维度计算。

In [ ]:
```
s = pd.Series([1,3,5,np.nan,6,8], index=dates).shift(2)
s,df
```

In [ ]:
```
df.sub(s, axis='index')
```

Apply 函数作用

通过apply()对函数作用

In [ ]:
```
df.apply(np.cumsum)
```

In [40]:
```
df.apply(lambda x:x.max()-x.min())
```

Out[40]:
```
A    0.000000
B    0.913023
C    1.666282
D    0.000000
F    4.000000
dtype: float64
```

频数统计

In [ ]:
```
s = pd.Series(np.random.randint(0, 7, size=10))
s
```

In [ ]:
```
s.value_counts()
```

字符串方法

对于Series对象，在其str属性中有着一系列的字符串处理方法。就如同下段代码一样，能很方便的对array中各个元素进行运算。值得注

意的是，在str属性中的模式匹配默认使用正则表达式。

In [43]:
```
s = pd.Series(['A', 'B', 'C', 'Aaba', 'Baca', np.nan, 'CABA', 'dog', 'cat'])
s.str.lower()
```

Out[43]:
```
0       a
1       b
2       c
3    aaba
4    baca
5     NaN
6    caba
7     dog
8     cat
dtype: object
```

合并

Concat 连接

pandas中提供了大量的方法能够轻松对Series，DataFrame和Panel对象进行不同满足逻辑关系的合并操作

通过**concat()**来连接pandas对象

In [ ]:
```
df = pd.DataFrame(np.random.randn(10,4))
df
```

In [ ]:
```
#break it into pieces
pieces = [df[:3], df[3:7], df[7:]]
pieces
```

In [ ]:
```
pd.concat(pieces)
```

Join 合并

类似于SQL中的合并(merge)

In [47]:
```
left = pd.DataFrame({'key':['foo', 'foo'], 'lval':[1,2]})
left
```

Out[47]:

	key	lval
0	foo	1
1	foo	2

In [48]:
```
right = pd.DataFrame({'key':['foo', 'foo'], 'lval':[4,5]})
right
```

Out[48]:

	key	lval
0	foo	4
1	foo	5

In [ ]:
```
pd.merge(left, right, on='key')
```

Append 添加

将若干行添加到dataFrame后面

In [ ]:
```
df = pd.DataFrame(np.random.randn(8, 4), columns=['A', 'B', 'C', 'D'])
df
```

In [51]:
```
s = df[2:3]
s
```

Out[51]:

	A	B	C	D
2	-0.136514	1.104717	0.290601	0.429694

In [ ]:
```
df.append(s, ignore_index=True)
```

数据透视表

In [ ]:
```
df = pd.DataFrame({'A' : ['one', 'one', 'two', 'three'] * 3,
                   'B' : ['A', 'B', 'C'] * 4,
                   'C' : ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] * 2,
                   'D' : np.random.randn(12),
                   'E' : np.random.randn(12)})
df
```

我们可以轻松地从这个数据得到透视表

In [ ]:
```
pd.pivot_table(df, values='D', index=['A', 'B'], columns=['C'])
```

时间序列

pandas在对频率转换进行重新采样时拥有着简单，强大而且高效的功能（例如把按秒采样的数据转换为按5分钟采样的数据）。这在金融领域很常见，但又不限于此。

In [ ]:
```
rng = pd.date_range('1/1/2023', periods=10, freq='S')
rng
```

In [ ]:
```
ts = pd.Series(np.random.randint(0,100,len(rng)), index=rng)
ts
```

In [57]:
```
rng = pd.date_range('1/1/2023', periods=5, freq='D')
rng
```

Out[57]:
```
DatetimeIndex(['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04',
               '2023-01-05'],
              dtype='datetime64[ns]', freq='D')
```

In [58]:
```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts
```

Out[58]:
```
2023-01-01   -1.839138
2023-01-02   -0.173644
2023-01-03    0.629050
2023-01-04    0.008660
2023-01-05   -0.270236
Freq: D, dtype: float64
```

时间跨度转换

In [59]:
```
rng = pd.date_range('1/1/2023', periods=5, freq='M')
rng
```

Out[59]:
```
DatetimeIndex(['2023-01-31', '2023-02-28', '2023-03-31', '2023-04-30',
               '2023-05-31'],
              dtype='datetime64[ns]', freq='M')
```

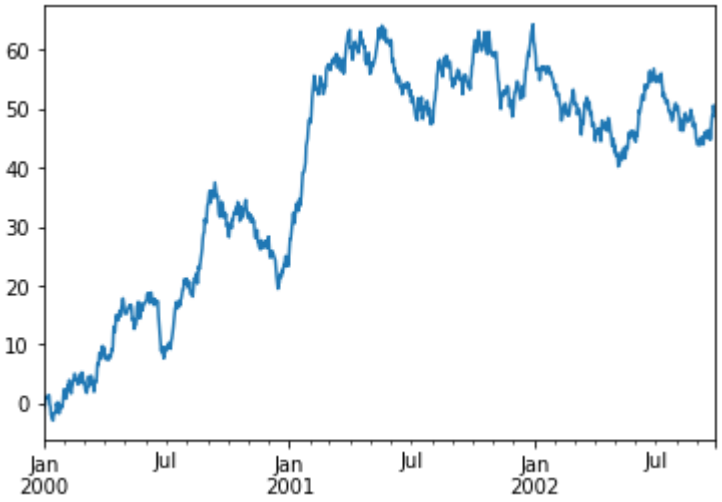
In [60]:
```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts
```

Out[60]:
```
2023-01-31   -0.000015
2023-02-28   -0.433859
2023-03-31    1.607081
2023-04-30    0.007694
2023-05-31    0.101212
Freq: M, dtype: float64
```

绘图

In [61]:
```
ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=1000))
ts = ts.cumsum()
ts.plot()
```

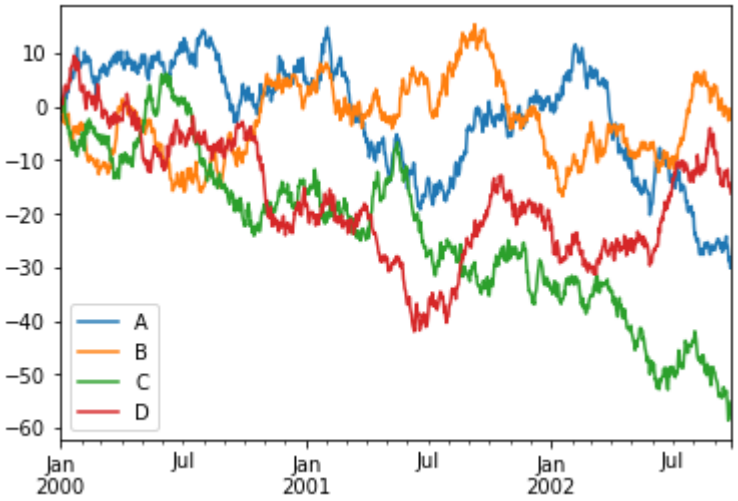
Out[61]:
```
<AxesSubplot:>
```



对于DataFrame类型，**plot()**能很方便地画出所有列及其标签

In [62]:
```
df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index, columns=['A', 'B', 'C', 'D'])
df = df.cumsum()
plt.figure(); df.plot(); plt.legend(loc='best')
```

Out[62]:
```
<matplotlib.legend.Legend at 0x15aff8de850>
<Figure size 432x288 with 0 Axes>
```



获取数据的I/O

CSV

写入一个csv文件

In [64]:
```
df.to_csv('data/foo.csv')
```

从一个csv文件读入

In [ ]:
```
pd.read_csv('data/foo.csv')
```

Excel

MS Excel的读写

写入一个Excel文件

In [66]:
```
df.to_excel('./foo.xlsx', sheet_name='Sheet1')
```

从一个excel文件读入

In [ ]:
```
pd.read_excel('./foo.xlsx', 'Sheet1', index_col=None, na_values=['NA'])
```