10 Minutes to pands

https://pandas.pydata.org/docs/user_guide/10min.html#object-creation

In [1]:

```
import numpy as np
import pandas as pd
```

Object creation

In [2]:

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

In [3]:

```
s
```

Out[3]:

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

In [4]:

```
dates = pd.date_range("20130101", periods=6)
```

In [5]:

```
dates
```

Out[5]:

```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

In [7]:

```
df = pd.DataFrame(np.random.randn(6,4), index = dates, columns= list("ABCD"))
```

In [8]:

```
df
```

Out[8]:

	A	B	C	D
2013-01-01	-1.398001	1.172644	-0.733226	1.460308
2013-01-02	-0.551749	-2.140217	0.965542	-0.458114
2013-01-03	-1.573659	-0.855627	0.560207	-0.702713
2013-01-04	-0.688304	-2.212931	-0.436499	0.724449
2013-01-05	0.203647	-1.176680	0.859715	0.762530
2013-01-06	-0.677257	-0.992430	-2.394600	-0.949607

In [9]:

```
df2 = pd.DataFrame({
    "A":1.0,
    "B":pd.Timestamp("20130102"),
    "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[9]:

	A	B	C	D	E	F
0	1.0	2013-01-02	1.0	3	test	foo
1	1.0	2013-01-02	1.0	3	train	foo
2	1.0	2013-01-02	1.0	3	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

In [10]:

```
df2.dtypes
```

Out[10]:

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

Viewing data

In [ ]:

```
df.head()
```

In [ ]:

```
df.tail(3)
```

In [ ]:

```
df.index
# 행 축 제목 보기
```

In [ ]:

```
df.columns
# 열 축 제목 보기
# 컬럼 보기
```

In [ ]:

```
df2.to_numpy()
# 출력 시 행 열의 라벨을 포함하지 않는다.
```

In [ ]:

```
df.describe()
```

In [ ]:

```
df.T
# 행 열 전환
```

In [ ]:

```
df.sort_index(axis=1, ascending = True)
```

In [ ]:

```
df.sort_values(by="B")
```

Sslection

Getting

In [ ]:

```
df["A"]
# Series 호출
```

In [ ]:

```
df[0:3]
# 행 슬라이스
```

In [ ]:

```
df["20130102":"20130104"]
```

Selection by Label

In [ ]:

```
df.loc[dates[0]]
```

In [ ]:

```
df.loc[:,["A","B"]]
# 다중 축 호출
```

In [ ]:

```
df.loc["20130102":"20130104",["A","B"]]
```

In [ ]:

```
df.loc["20130102",["A","B"]]
```

In [ ]:

```
df.loc[dates[0],"A"]
```

In [ ]:

```
df.at[dates[0],"A"]
```

Selection by Position

In [ ]:

```
df.iloc[3]
```

In [ ]:

```
df.iloc[2]
```

In [ ]:

```
df.iloc[3:5,0:2]
```

In [ ]:

```
df.iloc[[1,2,4],[0,2]]
```

In [ ]:

```
df.iloc[1:3,:]
```

In [ ]:

```
df.iloc[:,1:3]
```

In [ ]:

```
df.iloc[1,1]
```

In [ ]:

```
df.iat[1,1]
```

Boolean indexing

In [ ]:

```
df[df["A"]>0]
# 해당 컬럼에서 조건을 만족하는 행만 추출
```

In [ ]:

```
df[df>0]
```

In [ ]:

```
df2 = df.copy()
```

In [ ]:

```
df2["E"] = ["one","one","two","three","four","three"]
df2
```

In [ ]:

```
df2[df2["E"].isin(["two","four"])]
# isin()
```

Setting

In [11]:

```
s1 = pd.Series([1,2,3,4,5,6], index=pd.date_range("20130102", periods=6 ))
```

In [12]:

```
s1
```

Out[12]:

```
2013-01-02    1
2013-01-03    2
2013-01-04    3
2013-01-05    4
2013-01-06    5
2013-01-07    6
Freq: D, dtype: int64
```

In [13]:

```
df["F"]= s1
```

In [14]:

```
df.at[dates[0],"A"] = 0
```

In [15]:

```
df.iat[0,1] = 0
```

In [16]:

```
df.loc[:,"D"] = np.array([5] * len(df))
df
```

Out[16]:

	A	B	C	D	F
2013-01-01	0.000000	0.000000	-0.733226	5	NaN
2013-01-02	-0.551749	-2.140217	0.965542	5	1.0

	A	B	C	D	F
2013-01-03	-1.573659	-0.855627	0.560207	5	2.0
2013-01-04	-0.688304	-2.212931	-0.436499	5	3.0
2013-01-05	0.203647	-1.176680	0.859715	5	4.0
2013-01-06	-0.677257	-0.992430	-2.394600	5	5.0

In [17]:

```
df2 = df.copy()
```

In [18]:

```
df2[df2>0] = -df2
df2
```

Out[18]:

	A	B	C	D	F
2013-01-01	0.000000	0.000000	-0.733226	-5	NaN
2013-01-02	-0.551749	-2.140217	-0.965542	-5	-1.0
2013-01-03	-1.573659	-0.855627	-0.560207	-5	-2.0
2013-01-04	-0.688304	-2.212931	-0.436499	-5	-3.0
2013-01-05	-0.203647	-1.176680	-0.859715	-5	-4.0
2013-01-06	-0.677257	-0.992430	-2.394600	-5	-5.0

Missing data

In [19]:

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

Out[19]:

	A	B	C	D	F	E
2013-01-01	0.000000	0.000000	-0.733226	5	NaN	1.0
2013-01-02	-0.551749	-2.140217	0.965542	5	1.0	1.0
2013-01-03	-1.573659	-0.855627	0.560207	5	2.0	NaN
2013-01-04	-0.688304	-2.212931	-0.436499	5	3.0	NaN

In [20]:

```
df1.dropna(how="any")
# 결측치 있는 행 제거
```

Out[20]:

	A	B	C	D	F	E
2013-01-02	-0.551749	-2.140217	0.965542	5	1.0	1.0

In [21]:

```
df1.fillna(value= 5)
# 결측치 채우기
```

Out[21]:

	A	B	C	D	F	E

	A	B	C	D	F	E
2013-01-01	0.000000	0.000000	-0.733226	5	5.0	1.0
2013-01-02	-0.551749	-2.140217	0.965542	5	1.0	1.0
2013-01-03	-1.573659	-0.855627	0.560207	5	2.0	5.0
2013-01-04	-0.688304	-2.212931	-0.436499	5	3.0	5.0

In [22]:

```
pd.isna(df1)
# 결측치 확인
```

Out[22]:

	A	B	C	D	F	E
2013-01-01	False	False	False	False	True	False
2013-01-02	False	False	False	False	False	False
2013-01-03	False	False	False	False	False	True
2013-01-04	False	False	False	False	False	True

Operation

Stats

In [23]:

```
df.mean()
```

Out[23]:

```
A   -0.547887
B   -1.229648
C   -0.196477
D    5.000000
F    3.000000
dtype: float64
```

In [24]:

```
df.mean(1)
```

Out[24]:

```
2013-01-01    1.066694
2013-01-02    0.854715
2013-01-03    1.026184
2013-01-04    0.932453
2013-01-05    1.777336
2013-01-06    1.187142
Freq: D, dtype: float64
```

In [25]:

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

In [26]:

```
s
```

Out[26]:

```
2013-01-01    NaN
2013-01-02    NaN
2013-01-03    1.0
2013-01-04    3.0
2013-01-05    5.0
2013-01-06    NaN
Freq: D, dtype: float64
```

In [27]:

```
df.sub(s, axis="index")
```

Out[27]:

	A	B	C	D	F
2013-01-01	NaN	NaN	NaN	NaN	NaN
2013-01-02	NaN	NaN	NaN	NaN	NaN
2013-01-03	-2.573659	-1.855627	-0.439793	4.0	1.0
2013-01-04	-3.688304	-5.212931	-3.436499	2.0	0.0
2013-01-05	-4.796353	-6.176680	-4.140285	0.0	-1.0
2013-01-06	NaN	NaN	NaN	NaN	NaN

Apply

In [29]:

```
df.apply(np.cumsum)
```

Out[29]:

	A	B	C	D	F
2013-01-01	0.000000	0.000000	-0.733226	5	NaN
2013-01-02	-0.551749	-2.140217	0.232316	10	1.0
2013-01-03	-2.125407	-2.995844	0.792523	15	3.0
2013-01-04	-2.813712	-5.208775	0.356024	20	6.0
2013-01-05	-2.610065	-6.385455	1.215739	25	10.0
2013-01-06	-3.287322	-7.377886	-1.178861	30	15.0

In [32]:

```
df.apply(lambda x: x.max() - x.min())
```

Out[32]:

```
A    1.777306
B    2.212931
C    3.360142
D    0.000000
F    4.000000
dtype: float64
```

Histogramming

https://pandas.pydata.org/docs/user_guide/basics.html#basics-discretization

In [33]:

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

Out[33]:

```
0    3
1    2
2    2
3    3
4    5
5    6
6    0
7    1
8    2
9    0
dtype: int32
```

In [34]:

```
s.value_counts()
```

Out[34]:

```
2    3
0    2
3    2
1    1
5    1
6    1
dtype: int64
```

String Methods

In [35]:

```
s = pd.Series(["A","B","C","Aaba","Baca",np.nan, "CABA","dog","Cat"])
s.str.lower()
```

Out[35]:

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

Merge

https://pandas.pydata.org/docs/user_guide/merging.html#merging

Concat

In [37]:

```
df = pd.DataFrame(np.random.randn(10,4))
df
```

Out[37]:

	0	1	2	3
0	0.112079	-0.909524	-0.082731	0.370589
1	0.825064	0.418057	0.524716	0.913962
2	1.671239	1.200347	1.125526	0.819722
3	0.511291	0.611885	1.797537	-0.805969
4	1.031593	-0.531125	1.705441	1.013867
5	1.031899	-0.162714	-0.817580	0.865577
6	0.125774	-0.531272	-1.615048	-0.081109
7	0.139489	-0.124460	-0.498288	0.607304
8	-0.639346	-0.858044	2.055711	-0.325547
9	-0.057988	-3.002323	-0.104391	0.762115

In [38]:

```
pieces = [df[:3], df[3:7],df[7:]]
pd.concat(pieces)
```

Out[38]:

	0	1	2	3
0	0.112079	-0.909524	-0.082731	0.370589
1	0.825064	0.418057	0.524716	0.913962
2	1.671239	1.200347	1.125526	0.819722
3	0.511291	0.611885	1.797537	-0.805969
4	1.031593	-0.531125	1.705441	1.013867
5	1.031899	-0.162714	-0.817580	0.865577
6	0.125774	-0.531272	-1.615048	-0.081109
7	0.139489	-0.124460	-0.498288	0.607304
8	-0.639346	-0.858044	2.055711	-0.325547
9	-0.057988	-3.002323	-0.104391	0.762115

- 데이터 프레임의 컬럼을 합치는 건 상대적으로 빠르다.
- 하지만 행을 합치는 건 리소스 소모가 심하다.

Join

In [40]:

```
left = pd.DataFrame({"key":["foo","foo"], "lval":[1,2]})
right = pd.DataFrame({"key":["foo","foo"], "rval":[4,5]})
left
```

Out[40]:

	key	lval
0	foo	1
1	foo	2

In [41]:

```
right
```

Out[41]:

	key	rval
0	foo	4
1	foo	5

In [42]:

```
pd.merge(left,right, on ="key")
```

Out[42]:

	key	lval	rval
0	foo	1	4
1	foo	1	5
2	foo	2	4
3	foo	2	5

In [43]:

```
left = pd.DataFrame({"key":["foo","bar"], "lval":[1,2]})
right = pd.DataFrame({"key":["foo","bar"],"rval":[4,5]})
```

```
left
```

Out[43]:

	key	lval
0	foo	1
1	bar	2

In [44]:

```
right
```

Out[44]:

	key	rval
0	foo	4
1	bar	5

In [45]:

```
pd.merge(left,right, on="key")
```

Out[45]:

	key	lval	rval
0	foo	1	4
1	bar	2	5

Grouping

https://pandas.pydata.org/docs/user_guide/groupby.html#groupby

In [46]:

```
df = pd.DataFrame(
        {
            "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],
            "B": ["one", "one", "two", "three", "two", "two", "one", "three"],
            "C": np.random.randn(8),
            "D": np.random.randn(8),
        }
    )
df
```

Out[46]:

	A	B	C	D
0	foo	one	-0.529658	0.498554
1	bar	one	1.965949	0.338279
2	foo	two	-0.211798	-1.796187
3	bar	three	-0.032970	-2.755438
4	foo	two	-1.850905	-0.245048
5	bar	two	0.034017	-0.539988
6	foo	one	-1.089231	2.067523
7	foo	three	1.549236	1.837700

In [47]:

```
df.groupby("A").sum()
# 범주형 데이터는 결과물에서 제외 되었음
```

Out[47]:

	C	D
A		
bar	1.966996	-2.957147
foo	-2.132355	2.362541

In [48]:

```
df.groupby(["A","B"]).sum()
```

Out[48]:

		C	D
A	**B**		
bar	**one**	1.965949	0.338279
	three	-0.032970	-2.755438
	two	0.034017	-0.539988
foo	**one**	-1.618889	2.566077
	three	1.549236	1.837700
	two	-2.062703	-2.041235

Reshaping

https://pandas.pydata.org/docs/user_guide/advanced.html#advanced-hierarchical
https://pandas.pydata.org/docs/user_guide/reshaping.html#reshaping-stacking

Stack

In [51]:

```
tuples = list(
    zip(
        *[
            ["bar","bar","baz","baz","foo","foo","qux","qux"],
            ["one","two","one","two","one","two","one","two"],

        ]
    )
)

index = pd.MultiIndex.from_tuples(tuples, names=["first","second"])
df = pd.DataFrame(np.random.randn(8,2),index=index, columns=["A","B"])
df2 = df[:4]
df2
```

Out[51]:

		A	B
first	**second**		
bar	**one**	1.083845	-0.705348
	two	0.822413	-0.066388
baz	**one**	0.029070	0.293777
	two	1.398663	-0.374066

In [52]:

```
stacked = df2.stack()
```

In [53]:

```
stacked
```

Out[53]:

```
first  second
bar    one     A    1.083845
               B   -0.705348
       two     A    0.822413
               B   -0.066388
baz    one     A    0.029070
               B    0.293777
       two     A    1.398663
               B   -0.374066
dtype: float64
```

In [54]:

```
stacked.unstack()
```

Out[54]:

		A	**B**
first	**second**		
bar	**one**	1.083845	-0.705348
	two	0.822413	-0.066388
baz	**one**	0.029070	0.293777
	two	1.398663	-0.374066

In [55]:

```
stacked.unstack(1)
```

Out[55]:

	second	**one**	**two**
first			
bar	**A**	1.083845	0.822413
	B	-0.705348	-0.066388
baz	**A**	0.029070	1.398663
	B	0.293777	-0.374066

In [56]:

```
stacked.unstack(0)
```

Out[56]:

	first	**bar**	**baz**
second			
one	**A**	1.083845	0.029070
	B	-0.705348	0.293777
two	**A**	0.822413	1.398663
	B	-0.066388	-0.374066

Pivot tables

In [58]:

```
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
```

Out[58]:

	A	B	C	D	E
0	one	A	foo	-0.378944	0.508452
1	one	B	foo	-0.574639	-0.006769
2	two	C	foo	-1.290909	0.028426
3	three	A	bar	0.412040	-1.094314
4	one	B	bar	0.044262	-0.001531
5	one	C	bar	2.971559	1.464310
6	two	A	foo	-0.011837	0.082054
7	three	B	foo	-0.096568	1.146086
8	one	C	foo	-0.209786	2.609733
9	one	A	bar	1.464482	0.027279
10	two	B	bar	1.198594	0.555783
11	three	C	bar	-1.505907	-1.343206

In [59]:

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

Out[59]:

	C	bar	foo
A	**B**		
one	**A**	1.464482	-0.378944
	B	0.044262	-0.574639
	C	2.971559	-0.209786
three	**A**	0.412040	NaN
	B	NaN	-0.096568
	C	-1.505907	NaN
two	**A**	NaN	-0.011837
	B	1.198594	NaN
	C	NaN	-1.290909

Time series

In [61]:

```
rng = pd.date_range("1/1/2012", periods=100, freq="S")
ts = pd.Series(np.random.randint(0,500, len(rng)), index=rng)
ts.resample("5Min").sum()
```

Out[61]:
```
2012-01-01    26519
Freq: 5T, dtype: int32
```

In [62]:
```
rng = pd.date_range("3/6/2012 00:00", periods=5, freq="D")
ts = pd.Series(np.random.randn(len(rng)), rng)
ts
```

Out[62]:
```
2012-03-06   -0.882651
2012-03-07    0.033077
2012-03-08   -0.594378
2012-03-09   -0.438064
2012-03-10    1.221211
Freq: D, dtype: float64
```

In [63]:
```
ts_utc = ts.tz_localize("UTC")
ts_utc
```

Out[63]:
```
2012-03-06 00:00:00+00:00   -0.882651
2012-03-07 00:00:00+00:00    0.033077
2012-03-08 00:00:00+00:00   -0.594378
2012-03-09 00:00:00+00:00   -0.438064
2012-03-10 00:00:00+00:00    1.221211
Freq: D, dtype: float64
```

In [64]:
```
ts_utc.tz_convert("US/Eastern")
```

Out[64]:
```
2012-03-05 19:00:00-05:00   -0.882651
2012-03-06 19:00:00-05:00    0.033077
2012-03-07 19:00:00-05:00   -0.594378
2012-03-08 19:00:00-05:00   -0.438064
2012-03-09 19:00:00-05:00    1.221211
Freq: D, dtype: float64
```

In [65]:
```
rng = pd.date_range("1/1/2012", periods = 5, freq="M")
ts = pd.Series(np.random.randn(len(rng)), index = rng)
ts
```

Out[65]:
```
2012-01-31    0.191414
2012-02-29    0.796353
2012-03-31   -0.173852
2012-04-30    0.274467
2012-05-31   -2.236095
Freq: M, dtype: float64
```

In [66]:
```
ps = ts.to_period()
ps
```

Out[66]:
```
2012-01    0.191414
2012-02    0.796353
2012-03   -0.173852
2012-04    0.274467
2012-05   -2.236095
Freq: M, dtype: float64
```

In [67]:
```
ps.to_timestamp()
```

Out[67]:
```
2012-01-01    0.191414
2012-02-01    0.796353
2012-03-01   -0.173852
2012-04-01    0.274467
```

```
2012-05-01   -2.236095
Freq: MS, dtype: float64
```

In [68]:

```
prng = pd.period_range("1990Q1", "2000Q4", freq = "Q-NOV")
ts = pd.Series(np.random.randn(len(prng)), prng)
ts.index = (prng.asfreq("M","e") + 1).asfreq("H","s") + 9
ts.head()
```

Out[68]:

```
1990-03-01 09:00    1.463594
1990-06-01 09:00   -0.521600
1990-09-01 09:00    0.057676
1990-12-01 09:00   -0.095379
1991-03-01 09:00    0.424294
Freq: H, dtype: float64
```

Categoricals

In [69]:

```
df = pd.DataFrame({
    "id": [1,2,3,4,5,6] , "raw_grade" : ["a","b","b","a","a","e"]
})

df["grade"] = df["raw_grade"].astype("category")
df["grade"]

## 범주화
```

Out[69]:

```
0    a
1    b
2    b
3    a
4    a
5    e
Name: grade, dtype: category
Categories (3, object): ['a', 'b', 'e']
```

In [70]:

```
df["grade"].cat.categories = ["Very Good", "Good", "Very Bad"]
df["grade"] = df["grade"].cat.set_categories(
    ["Very Bad","Bad","Medium","Good","Very Good"])

df["grade"]
```

Out[70]:

```
0    Very Good
1         Good
2         Good
3    Very Good
4    Very Good
5     Very Bad
Name: grade, dtype: category
Categories (5, object): ['Very Bad', 'Bad', 'Medium', 'Good', 'Very Good']
```

In [72]:

```
df.sort_values(by="grade")
```

Out[72]:

	id	raw_grade	grade
5	6	e	Very Bad
1	2	b	Good
2	3	b	Good
0	1	a	Very Good
3	4	a	Very Good

	id	raw_grade	grade
4	5	a	Very Good

In [73]:

```
df.groupby("grade").size()
```

Out[73]:

```
grade
Very Bad     1
Bad          0
Medium       0
Good         2
Very Good    3
dtype: int64
```

Ploting

In [74]:

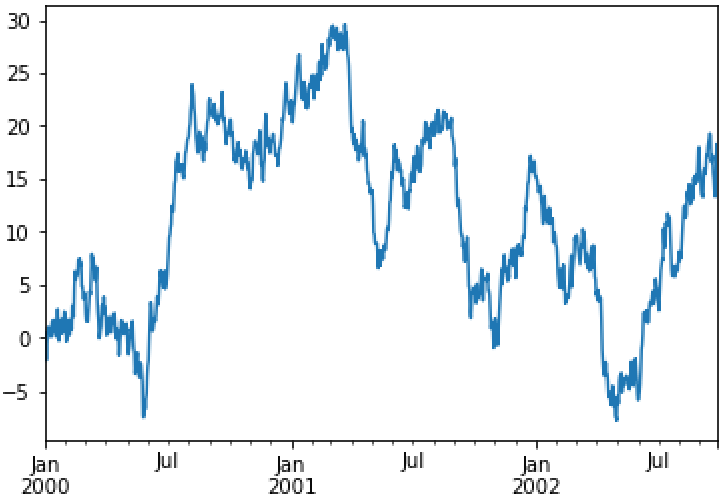
```
import matplotlib.pyplot as plt
plt.close("all")
```

In [75]:

```
ts = pd.Series(np.random.randn(1000), index=pd.date_range("1/1/2000", periods = 1000)
ts = ts.cumsum()
ts.plot()
```

Out[75]:

```
<AxesSubplot:>
```



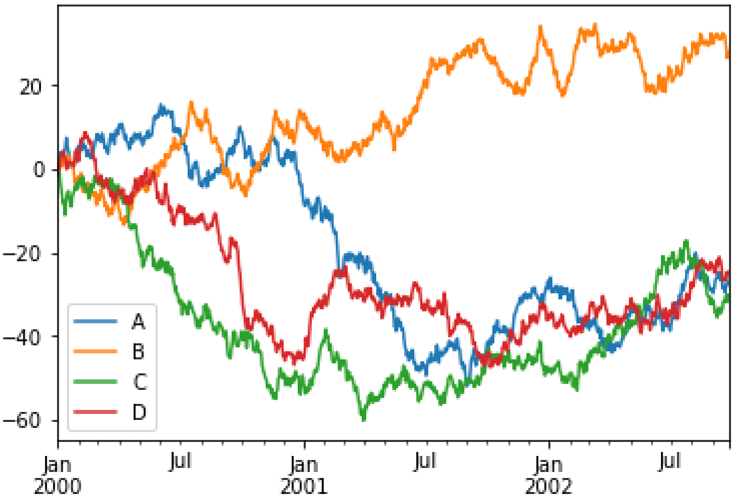
In [76]:

```
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[76]:

```
<matplotlib.legend.Legend at 0x1b7240d74f0>
```

```
<Figure size 432x288 with 0 Axes>
```



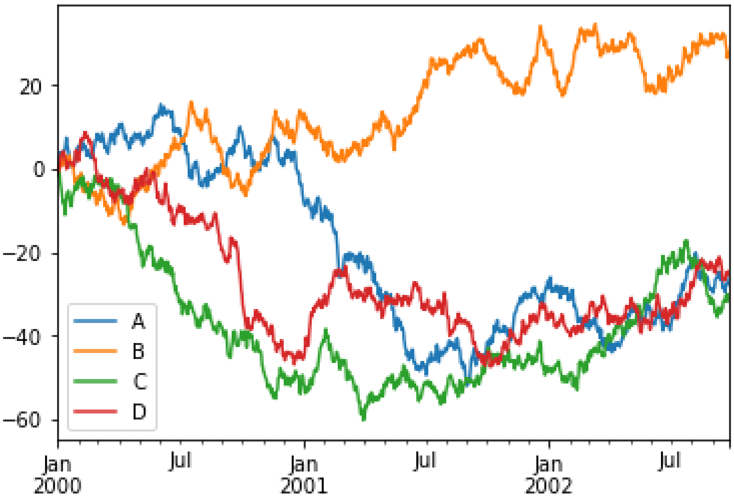
In [79]:

```
plt.figure()
df.plot()
plt.legend(loc='best')
```

Out[79]:

```
<matplotlib.legend.Legend at 0x1b724305370>
```

```
<Figure size 432x288 with 0 Axes>
```



Getting data in/out

CSV

In [ ]:

```
df.to_csv("fadfa.cs")
```