

# Python Data Processing with Pandas

CSE 5542

Introduction to Data Visualization

# Pandas

- A very powerful package of Python for manipulating tables
- Built on top of numpy, so is efficient
- Save you a lot of effort from writing lower python code for manipulating, extracting, and deriving tables related information
- Easy visualization with Matplotlib
- Main data structures – Series and DataFrame

- First thing first

```
In [1]: import pandas as pd  
In [2]: import numpy as np  
In [3]: import matplotlib.pyplot as plt
```

- Series: an indexed 1D array

```
In[2]: data = pd.Series([0.25, 0.5, 0.75, 1.0])  
data  
Out[2]: 0    0.25  
       1    0.50  
       2    0.75  
       3    1.00  
dtype: float64
```

- Explicit index

```
In[7]: data = pd.Series([0.25, 0.5, 0.75, 1.0],  
                        index=['a', 'b', 'c', 'd'])
```

```
data  
Out[7]: a    0.25  
       b    0.50  
       c    0.75  
       d    1.00  
       dtype: float64
```

- Access data

```
In[8]: data['b']  
Out[8]: 0.5
```

- Can work as a dictionary

```
In[11]: population_dict = {'California': 38332521,
                             'Texas': 26448193,
                             'New York': 19651127,
                             'Florida': 19552860,
                             'Illinois': 12882135}

population = pd.Series(population_dict)
population
Out[11]: California    38332521
         Florida      19552860
         Illinois     12882135
         New York     19651127
         Texas        26448193
         dtype: int64
```

- Access and slice data

```
In[12]: population['California']
Out[12]: 38332521
```

```
In[13]: population['California':'Illinois']
Out[13]: California    38332521
         Florida      19552860
         Illinois     12882135
         dtype: int64
```

# DataFrame Object

- Generalized two dimensional array with flexible row and column indices

Constructing DataFrame from a dictionary.

```
>>> d = {'col1': [1, 2], 'col2': [3, 4]}
>>> df = pd.DataFrame(data=d)
>>> df
```

	col1	col2
0	1	3
1	2	4

# DataFrame Object

- Generalized two dimensional array with flexible row and column indices

Constructing DataFrame from numpy ndarray:

```
>>> df2 = pd.DataFrame(np.random.randint(low=0, high=10, size=(5, 5)),  
...                      columns=['a', 'b', 'c', 'd', 'e'])  
>>> df2
```

	a	b	c	d	e
0	2	8	8	3	4
1	4	2	9	0	9
2	1	0	7	8	0
3	5	1	7	1	3
4	6	0	2	4	2

# DataFrame Object

- From Pandas Series

```
In[11]: population_dict = {'California': 38332521,  
                           'Texas': 26448193,  
                           'New York': 19651127,  
                           'Florida': 19552860,  
                           'Illinois': 12882135}
```

```
population = pd.Series(population_dict)  
population
```

```
Out[11]: California    38332521  
         Florida       19552860  
         Illinois      12882135  
         New York      19651127  
         Texas         26448193  
         dtype: int64
```

```
In[18]:
```

```
area_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297,  
             'Florida': 170312, 'Illinois': 149995}
```

```
area = pd.Series(area_dict)  
area
```

```
Out[18]: California    423967  
         Florida       170312  
         Illinois      149995  
         New York      141297  
         Texas         695662  
         dtype: int64
```



# DataFrame Object

- From Pandas Series

```
In[19]: states = pd.DataFrame({'population': population,  
                                'area': area})
```

```
states  
Out[19]:
```

	area	population
California	423967	38332521
Florida	170312	19552860
Illinois	149995	12882135
New York	141297	19651127
Texas	695662	26448193

# DataFrame Object

- Another example

```
In [6]: dates = pd.date_range('20130101', periods=6)

In [7]: dates
Out[7]:
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 [8]: df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=['A', 'B', 'C', 'D'])

In [9]: df
Out[9]:
```

	A	B	C	D
2013-01-01	0.469112	-0.282863	-1.509059	-1.135632
2013-01-02	1.212112	-0.173215	0.119209	-1.044236
2013-01-03	-0.861849	-2.104569	-0.494929	1.071804
2013-01-04	0.721555	-0.706771	-1.039575	0.271860
2013-01-05	-0.424972	0.567020	0.276232	-1.087401
2013-01-06	-0.673690	0.113648	-1.478427	0.524988

# Viewing Data

- View the first or last N rows

```
In [14]: df.head()
```

```
Out[14]:
```

	A	B	C	D
2013-01-01	0.469112	-0.282863	-1.509059	-1.135632
2013-01-02	1.212112	-0.173215	0.119209	-1.044236
2013-01-03	-0.861849	-2.104569	-0.494929	1.071804
2013-01-04	0.721555	-0.706771	-1.039575	0.271860
2013-01-05	-0.424972	0.567020	0.276232	-1.087401

```
In [15]: df.tail(3)
```

```
Out[15]:
```

	A	B	C	D
2013-01-04	0.721555	-0.706771	-1.039575	0.271860
2013-01-05	-0.424972	0.567020	0.276232	-1.087401
2013-01-06	-0.673690	0.113648	-1.478427	0.524988

# Viewing Data

- Display the index, columns, and data

```
In [16]: df.index
```

```
Out[16]:
```

```
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 [17]: df.columns
```

```
Out[17]: Index(['A', 'B', 'C', 'D'], dtype='object')
```

```
In [18]: df.values
```

```
Out[18]:
```

```
array([[ 0.4691, -0.2829, -1.5091, -1.1356],  
       [ 1.2121, -0.1732,  0.1192, -1.0442],  
       [-0.8618, -2.1046, -0.4949,  1.0718],  
       [ 0.7216, -0.7068, -1.0396,  0.2719],  
       [-0.425 ,  0.567 ,  0.2762, -1.0874],  
       [-0.6737,  0.1136, -1.4784,  0.525 ]])
```

# Viewing Data

- Quick statistics (for columns A B C D in this case)

```
In [19]: df.describe()
```

```
Out[19]:
```

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	0.073711	-0.431125	-0.687758	-0.233103
std	0.843157	0.922818	0.779887	0.973118
min	-0.861849	-2.104569	-1.509059	-1.135632
25%	-0.611510	-0.600794	-1.368714	-1.076610
50%	0.022070	-0.228039	-0.767252	-0.386188
75%	0.658444	0.041933	-0.034326	0.461706
max	1.212112	0.567020	0.276232	1.071804

# Viewing Data

- Sorting: sort by the index (i.e., reorder columns or rows), not by the data in the table

column



```
In [21]: df.sort_index(axis=1, ascending=False)
```

```
Out[21]:
```

	D	C	B	A
2013-01-01	-1.135632	-1.509059	-0.282863	0.469112
2013-01-02	-1.044236	0.119209	-0.173215	1.212112
2013-01-03	1.071804	-0.494929	-2.104569	-0.861849
2013-01-04	0.271860	-1.039575	-0.706771	0.721555
2013-01-05	-1.087401	0.276232	0.567020	-0.424972
2013-01-06	0.524988	-1.478427	0.113648	-0.673690

# Viewing Data

- Sorting: sort by the data values

```
In [22]: df.sort_values(by='B')
```

```
Out[22]:
```

	A	B	C	D
2013-01-03	-0.861849	-2.104569	-0.494929	1.071804
2013-01-04	0.721555	-0.706771	-1.039575	0.271860
2013-01-01	0.469112	-0.282863	-1.509059	-1.135632
2013-01-02	1.212112	-0.173215	0.119209	-1.044236
2013-01-06	-0.673690	0.113648	-1.478427	0.524988
2013-01-05	-0.424972	0.567020	0.276232	-1.087401

# Selecting Data

- Selecting using a label

```
In [26]: df.loc[dates[0]]
```

```
Out[26]:
```

```
A    0.469112
```

```
B   -0.282863
```

```
C   -1.509059
```

```
D   -1.135632
```

```
Name: 2013-01-01 00:00:00, dtype: float64
```



# Selecting Data

- Multi-axis, by label

```
In [27]: df.loc[:,['A','B']]
```

```
Out[27]:
```

	A	B
2013-01-01	0.469112	-0.282863
2013-01-02	1.212112	-0.173215
2013-01-03	-0.861849	-2.104569
2013-01-04	0.721555	-0.706771
2013-01-05	-0.424972	0.567020
2013-01-06	-0.673690	0.113648

# Selecting Data

- Multi-axis, by label

Slicing: last included



```
In [28]: df.loc['20130102':'20130104',['A','B']]
```

```
Out[28]:
```

	A	B
2013-01-02	1.212112	-0.173215
2013-01-03	-0.861849	-2.104569
2013-01-04	0.721555	-0.706771

# Selecting Data

- Select by position

```
In [32]: df.iloc[3]
```

```
Out[32]:
```

```
A    0.721555
```

```
B   -0.706771
```

```
C   -1.039575
```

```
D    0.271860
```

```
Name: 2013-01-04 00:00:00, dtype: float64
```

```
In [33]: df.iloc[3:5,0:2]
```

```
Out[33]:
```

	A	B
2013-01-04	0.721555	-0.706771
2013-01-05	-0.424972	0.567020

# Selecting Data

- Boolean indexing

```
In [40]: df[df > 0]
```

```
Out[40]:
```

	A	B	C	D
2013-01-01	0.469112	NaN	NaN	NaN
2013-01-02	1.212112	NaN	0.119209	NaN
2013-01-03	NaN	NaN	NaN	1.071804
2013-01-04	0.721555	NaN	NaN	0.271860
2013-01-05	NaN	0.567020	0.276232	NaN
2013-01-06	NaN	0.113648	NaN	0.524988

# Selecting Data

- Boolean indexing

```
In [41]: df2 = df.copy()
```

```
In [42]: df2['E'] = ['one', 'one', 'two', 'three', 'four', 'three']
```

```
In [43]: df2
```

```
Out[43]:
```

	A	B	C	D	E
2013-01-01	0.469112	-0.282863	-1.509059	-1.135632	one
2013-01-02	1.212112	-0.173215	0.119209	-1.044236	one
2013-01-03	-0.861849	-2.104569	-0.494929	1.071804	two
2013-01-04	0.721555	-0.706771	-1.039575	0.271860	three
2013-01-05	-0.424972	0.567020	0.276232	-1.087401	four
2013-01-06	-0.673690	0.113648	-1.478427	0.524988	three

```
In [44]: df2[df2['E'].isin(['two', 'four'])]
```

```
Out[44]:
```

	A	B	C	D	E
2013-01-03	-0.861849	-2.104569	-0.494929	1.071804	two
2013-01-05	-0.424972	0.567020	0.276232	-1.087401	four

# Setting Data

- Setting a new column aligned by indexes

```
In [45]: s1 = pd.Series([1,2,3,4,5,6], index=pd.date_range('20130102', periods=6))
```

```
In [46]: s1
```

```
Out[46]:
```

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

# Setting Data

Setting values by label

```
In [48]: df.at[dates[0], 'A'] = 0
```

Setting values by position

```
In [49]: df.iat[0,1] = 0
```

Setting by assigning with a numpy array

```
In [50]: df.loc[:, 'D'] = np.array([5] * len(df))
```

The result of the prior setting operations

```
In [51]: df
```

```
Out[51]:
```

	A	B	C	D	F
2013-01-01	0.000000	0.000000	-1.509059	5	NaN
2013-01-02	1.212112	-0.173215	0.119209	5	1.0
2013-01-03	-0.861849	-2.104569	-0.494929	5	2.0
2013-01-04	0.721555	-0.706771	-1.039575	5	3.0
2013-01-05	-0.424972	0.567020	0.276232	5	4.0
2013-01-06	-0.673690	0.113648	-1.478427	5	5.0

# Operations

- Descriptive statistics
  - Across axis 0 (rows), i.e., column mean

```
In [61]: df.mean()  
Out[61]:  
A    -0.004474  
B    -0.383981  
C    -0.687758  
D     5.000000  
F     3.000000  
dtype: float64
```

- Across axis 1 (column), i.e., row mean

```
In [62]: df.mean(1)  
Out[62]:  
2013-01-01    0.872735  
2013-01-02    1.431621  
2013-01-03    0.707731  
2013-01-04    1.395042  
2013-01-05    1.883656  
2013-01-06    1.592306  
Freq: D, dtype: float64
```



# Operations

- Apply

```
In [66]: df.apply(np.cumsum)
Out[66]:
```

	A	B	C	D	F
2013-01-01	0.000000	0.000000	-1.509059	5	NaN
2013-01-02	1.212112	-0.173215	-1.389850	10	1.0
2013-01-03	0.350263	-2.277784	-1.884779	15	3.0
2013-01-04	1.071818	-2.984555	-2.924354	20	6.0
2013-01-05	0.646846	-2.417535	-2.648122	25	10.0
2013-01-06	-0.026844	-2.303886	-4.126549	30	15.0

```
In [67]: df.apply(lambda x: x.max() - x.min())
Out[67]:
```

A	2.073961
B	2.671590
C	1.785291
D	0.000000
F	4.000000

dtype: float64

- Histogram

```
In [68]: s = pd.Series(np.random.randint(0, 7, size=10))

In [69]: s
Out[69]:
```

0	4
1	2
2	1
3	2
4	6
5	4
6	4
7	6
8	4
9	4

dtype: int64

```
In [70]: s.value_counts()
Out[70]:
```

4	5
6	2
2	2
1	1

dtype: int64

# Merge Tables

- Join

```
In [82]: left = pd.DataFrame({'key': ['foo', 'bar'], 'lval': [1, 2]})  
  
In [83]: right = pd.DataFrame({'key': ['foo', 'bar'], 'rval': [4, 5]})  
  
In [84]: left  
Out[84]:  
   key  lval  
0  foo     1  
1  bar     2  
  
In [85]: right  
Out[85]:  
   key  rval  
0  foo     4  
1  bar     5  
  
In [86]: pd.merge(left, right, on='key')  
Out[86]:  
   key  lval  rval  
0  foo     1     4  
1  bar     2     5
```

# Merge Tables

- Append

```
In [87]: df = pd.DataFrame(np.random.randn(8, 4), columns=['A', 'B', 'C', 'D'])
```

```
In [88]: df
```

```
Out[88]:
```

	A	B	C	D
0	1.346061	1.511763	1.627081	-0.990582
1	-0.441652	1.211526	0.268520	0.024580
2	-1.577585	0.396823	-0.105381	-0.532532
3	1.453749	1.208843	-0.080952	-0.264610
4	-0.727965	-0.589346	0.339969	-0.693205
5	-0.339355	0.593616	0.884345	1.591431
6	0.141809	0.220390	0.435589	0.192451
7	-0.096701	0.803351	1.715071	-0.708758

```
In [89]: s = df.iloc[3]
```

```
In [90]: df.append(s, ignore_index=True)
```

```
Out[90]:
```

	A	B	C	D
0	1.346061	1.511763	1.627081	-0.990582
1	-0.441652	1.211526	0.268520	0.024580
2	-1.577585	0.396823	-0.105381	-0.532532
3	1.453749	1.208843	-0.080952	-0.264610
4	-0.727965	-0.589346	0.339969	-0.693205
5	-0.339355	0.593616	0.884345	1.591431
6	0.141809	0.220390	0.435589	0.192451
7	-0.096701	0.803351	1.715071	-0.708758
8	1.453749	1.208843	-0.080952	-0.264610

# Grouping

```
In [91]: 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)})  
.....:
```

```
In [92]: df
```

```
Out[92]:
```

	A	B	C	D
0	foo	one	-1.202872	-0.055224
1	bar	one	-1.814470	2.395985
2	foo	two	1.018601	1.552825
3	bar	three	-0.595447	0.166599
4	foo	two	1.395433	0.047609
5	bar	two	-0.392670	-0.136473
6	foo	one	0.007207	-0.561757
7	foo	three	1.928123	-1.623033

```
In [93]: df.groupby('A').sum()
```

```
Out[93]:
```

	C	D
A		
bar	-2.802588	2.42611
foo	3.146492	-0.63958

```
In [94]: df.groupby(['A', 'B']).sum()
```

```
Out[94]:
```

		C	D
A	B		
bar	one	-1.814470	2.395985
	three	-0.595447	0.166599
	two	-0.392670	-0.136473
foo	one	-1.195665	-0.616981
	three	1.928123	-1.623033
	two	2.414034	1.600434

# File I/O

- CSV

```
In [142]: pd.read_csv('foo.csv')
Out[142]:
```

	Unnamed: 0	A	B	C	D
0	2000-01-01	0.266457	-0.399641	-0.219582	1.186860
1	2000-01-02	-1.170732	-0.345873	1.653061	-0.282953
2	2000-01-03	-1.734933	0.530468	2.060811	-0.515536
3	2000-01-04	-1.555121	1.452620	0.239859	-1.156896
4	2000-01-05	0.578117	0.511371	0.103552	-2.428202
5	2000-01-06	0.478344	0.449933	-0.741620	-1.962409
6	2000-01-07	1.235339	-0.091757	-1.543861	-1.084753
..	...	...	...	...	...
993	2002-09-20	-10.628548	-9.153563	-7.883146	28.313940
994	2002-09-21	-10.390377	-8.727491	-6.399645	30.914107
995	2002-09-22	-8.985362	-8.485624	-4.669462	31.367740
996	2002-09-23	-9.558560	-8.781216	-4.499815	30.518439
997	2002-09-24	-9.902058	-9.340490	-4.386639	30.105593
998	2002-09-25	-10.216020	-9.480682	-3.933802	29.758560
999	2002-09-26	-11.856774	-10.671012	-3.216025	29.369368

[1000 rows x 5 columns]

# File I/O

- Excel

```
In [146]: pd.read_excel('foo.xlsx', 'Sheet1', index_col=None, na_values=['NA'])
```

```
Out[146]:
```

	A	B	C	D
2000-01-01	0.266457	-0.399641	-0.219582	1.186860
2000-01-02	-1.170732	-0.345873	1.653061	-0.282953
2000-01-03	-1.734933	0.530468	2.060811	-0.515536
2000-01-04	-1.555121	1.452620	0.239859	-1.156896
2000-01-05	0.578117	0.511371	0.103552	-2.428202
2000-01-06	0.478344	0.449933	-0.741620	-1.962409
2000-01-07	1.235339	-0.091757	-1.543861	-1.084753
...	...	...	...	...
2002-09-20	-10.628548	-9.153563	-7.883146	28.313940
2002-09-21	-10.390377	-8.727491	-6.399645	30.914107
2002-09-22	-8.985362	-8.485624	-4.669462	31.367740
2002-09-23	-9.558560	-8.781216	-4.499815	30.518439
2002-09-24	-9.902058	-9.340490	-4.386639	30.105593
2002-09-25	-10.216020	-9.480682	-3.933802	29.758560
2002-09-26	-11.856774	-10.671012	-3.216025	29.369368

```
[1000 rows x 4 columns]
```