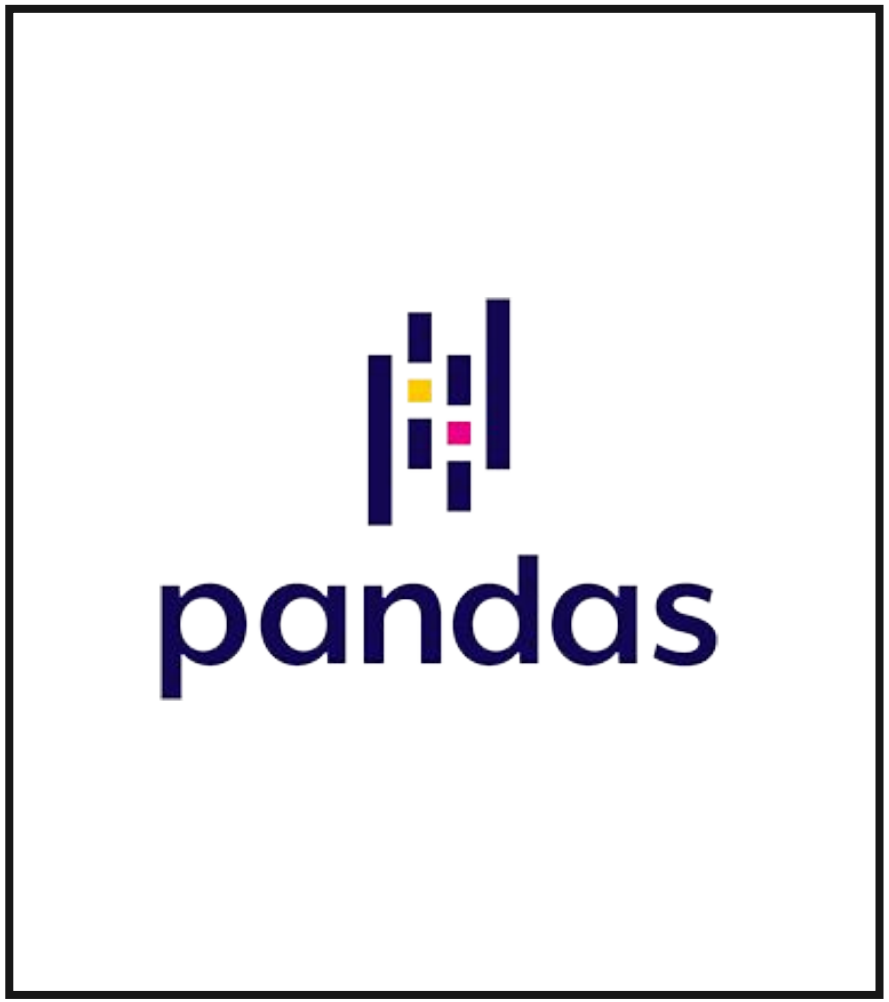

Python Data Processing with Pandas

Mobina Shahbandeh

University of Tehran ACM Summer School 2021

Pandas

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

```
$ pip install pandas
```

Series

First things first

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

Series: an indexed 1D array

In [2]:
```
1 data = pd.Series([0.25, 0.5, 0.75, 1.0])
2 data
```

```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
```

Series

Explicit index

```
In [3]:
1 data.index = ['a', 'b', 'c', 'd']
```

```
In [4]:
1 data
```

```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

Access data

```
In [5]:
1 data['b']
```

```
0.5
```

Series

Can work as a dictionary

```
In [6]:
1 population_dict = {'California': 38332521,
2                    'Texas': 26448193,
3                    'New York': 19651127,
4                    'Florida':19552860,
5                    'Illinois': 12882135}
```

```
In [7]:
1 population = pd.Series(population_dict)
2 population
```

```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
dtype: int64
```

Access and slice data

```
In [8]:
1 population['California']
```

```
38332521
```

```
In [9]:
1 population['California':'Illinois']
```

```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
dtype: int64
```

DataFrame

Generalized two dimensional array with flexible row and column indices

Constructing DataFrame from a dictionary

In [10]:
```
1 d = {'col1':[1,2], 'col2':[3,4]}
```

In [11]:
```
1 df = pd.DataFrame(data=d)
2 df
```

	col1	col2
0	1	3
1	2	4

DataFrame

Constructing DataFrame from a numpy ndarray

In [12]:

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

DataFrame

Constructing DataFrame from pandas Series

In [13]:
```
area_dict={'California':423967,
           'Texas':695662,
           'New York':141297,
           'Florida':170312,
           'Illinois':149995}
```

In [14]:
```
area = pd.Series(area_dict)
area
```

```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
dtype: int64
```

In [6]:
```
population_dict = {'California': 38332521,
                   'Texas': 26448193,
                   'New York': 19651127,
                   'Florida':19552860,
                   'Illinois': 12882135}
```

In [7]:
```
population = pd.Series(population_dict)
population
```

```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
dtype: int64
```

DataFrame

Constructing DataFrame from pandas Series

In [15]:

```
states = pd.DataFrame({'population':population,
                       'area':area})
states
```

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

DataFrame

Another example

In [16]:

```
1 dates= pd.date_range('20130101', periods=6)
2 dates
```

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

DataFrame

Another example

In [17]:

```
df = pd.DataFrame(np.random.randn(6,4),
                  index = dates,
                  columns=['A', 'B', 'C', 'D'])
df
```

	A	B	C	D
2013-01-01	2.758689	0.278113	2.494974	0.010741
2013-01-02	0.325493	-0.970221	-0.881164	1.210782
2013-01-03	1.354842	0.313634	1.224231	-0.235177
2013-01-04	0.885647	-0.297321	-1.628925	0.472148
2013-01-05	-0.883835	0.699636	0.397156	1.072433
2013-01-06	0.888353	2.125925	-1.507256	-1.243995

Viewing Data

View the first or last N rows

In [18]:
```
1 df.head()
```

	A	B	C	D
2013-01-01	2.758689	0.278113	2.494974	0.010741
2013-01-02	0.325493	-0.970221	-0.881164	1.210782
2013-01-03	1.354842	0.313634	1.224231	-0.235177
2013-01-04	0.885647	-0.297321	-1.628925	0.472148
2013-01-05	-0.883835	0.699636	0.397156	1.072433

In [19]:
```
1 df.tail(3)
```

	A	B	C	D
2013-01-04	0.885647	-0.297321	-1.628925	0.472148
2013-01-05	-0.883835	0.699636	0.397156	1.072433
2013-01-06	0.888353	2.125925	-1.507256	-1.243995

Viewing Data

Display the index, columns, and data

In [20]:
```
1 df.index
```

```
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 [21]:
```
1 df.columns
```

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

In [22]:
```
1 df.values
```

```
array([[ 2.7586886 ,  0.27811306,  2.49497409,  0.01074096],
       [ 0.3254933 , -0.97022098, -0.88116387,  1.21078182],
       [ 1.3548424 ,  0.31363401,  1.22423075, -0.23517736],
       [ 0.88564719, -0.29732101, -1.62892456,  0.47214846],
       [-0.88383453,  0.69963593,  0.39715609,  1.07243288],
       [ 0.88835281,  2.1259253 , -1.5072558 , -1.24399463]])
```

Viewing Data

Quick statistics

```
In [23]:
1 df.describe()
```

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	0.888198	0.358294	0.016503	0.214489
std	1.197767	1.043479	1.648114	0.912792
min	-0.883835	-0.970221	-1.628925	-1.243995
25%	0.465532	-0.153462	-1.350733	-0.173698
50%	0.887000	0.295874	-0.242004	0.241445
75%	1.238220	0.603135	1.017462	0.922362
max	2.758689	2.125925	2.494974	1.210782

Viewing Data

Sorting

Sort by the index (i.e., reorder columns or rows), not by the data in the table

```
In [24]:  1  df.sort_index(axis = 1, ascending = False)
```

	D	C	B	A
2013-01-01	0.010741	2.494974	0.278113	2.758689
2013-01-02	1.210782	-0.881164	-0.970221	0.325493
2013-01-03	-0.235177	1.224231	0.313634	1.354842
2013-01-04	0.472148	-1.628925	-0.297321	0.885647
2013-01-05	1.072433	0.397156	0.699636	-0.883835
2013-01-06	-1.243995	-1.507256	2.125925	0.888353

Sort by the data values

```
In [25]:  1  df.sort_values(by='B')
```

	A	B	C	D
2013-01-02	0.325493	-0.970221	-0.881164	1.210782
2013-01-04	0.885647	-0.297321	-1.628925	0.472148
2013-01-01	2.758689	0.278113	2.494974	0.010741
2013-01-03	1.354842	0.313634	1.224231	-0.235177
2013-01-05	-0.883835	0.699636	0.397156	1.072433
2013-01-06	0.888353	2.125925	-1.507256	-1.243995

Selecting Data

Selecting using a label

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

```
A    2.758689
B    0.278113
C    2.494974
D    0.010741
Name: 2013-01-01 00:00:00, dtype: float64
```

Selecting Data

Multi-axis, by label

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

	A	B
2013-01-01	2.758689	0.278113
2013-01-02	0.325493	-0.970221
2013-01-03	1.354842	0.313634
2013-01-04	0.885647	-0.297321
2013-01-05	-0.883835	0.699636
2013-01-06	0.888353	2.125925

Selecting Data

Multi-axis, by label

Slicing: last included

```
In [28]:  1  df.loc['20130102':'20130104', ['A', 'B']]
```

	A	B
2013-01-02	0.325493	-0.970221
2013-01-03	1.354842	0.313634
2013-01-04	0.885647	-0.297321

Selecting Data

By position

In [29]:
```
1 df.iloc[3]
```

```
A    0.885647
B   -0.297321
C   -1.628925
D    0.472148
Name: 2013-01-04 00:00:00, dtype: float64
```

In [30]:
```
1 df.iloc[3:5, 0:2]
```

	A	B
2013-01-04	0.885647	-0.297321
2013-01-05	-0.883835	0.699636

Selecting Data

Boolean indexing

```
In [31]:
1 df[df>0]
```

	A	B	C	D
2013-01-01	2.758689	0.278113	2.494974	0.010741
2013-01-02	0.325493	NaN	NaN	1.210782
2013-01-03	1.354842	0.313634	1.224231	NaN
2013-01-04	0.885647	NaN	NaN	0.472148
2013-01-05	NaN	0.699636	0.397156	1.072433
2013-01-06	0.888353	2.125925	NaN	NaN

Selecting Data

Boolean indexing

In [32]:
```
df2 = df.copy()
```

In [33]:
```
df2['E']= ['one', 'one', 'two', 'three', 'four', 'three']
df2
```

	A	B	C	D	E
2013-01-01	2.758689	0.278113	2.494974	0.010741	one
2013-01-02	0.325493	-0.970221	-0.881164	1.210782	one
2013-01-03	1.354842	0.313634	1.224231	-0.235177	two
2013-01-04	0.885647	-0.297321	-1.628925	0.472148	three
2013-01-05	-0.883835	0.699636	0.397156	1.072433	four
2013-01-06	0.888353	2.125925	-1.507256	-1.243995	three

In [34]:
```
df2[df2['E'].isin(['two', 'four'])]
```

	A	B	C	D	E
2013-01-03	1.354842	0.313634	1.224231	-0.235177	two
2013-01-05	-0.883835	0.699636	0.397156	1.072433	four

Setting Data

Setting a new column aligned by indices

In [35]:
```
s1 = pd.Series([1,2,3,4,5,6], index=pd.date_range('2013010
s1
```

```
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 [36]:
```
df['F']=s1
df
```

	A	B	C	D	F
2013-01-01	2.758689	0.278113	2.494974	0.010741	NaN
2013-01-02	0.325493	-0.970221	-0.881164	1.210782	1.0
2013-01-03	1.354842	0.313634	1.224231	-0.235177	2.0
2013-01-04	0.885647	-0.297321	-1.628925	0.472148	3.0
2013-01-05	-0.883835	0.699636	0.397156	1.072433	4.0
2013-01-06	0.888353	2.125925	-1.507256	-1.243995	5.0

Setting Data

Setting values by label

In [37]:

```
1 df.at[dates[0], 'A']=0
2 df
```

	A	B	C	D	F
2013-01-01	0.000000	0.278113	2.494974	0.010741	NaN
2013-01-02	0.325493	-0.970221	-0.881164	1.210782	1.0
2013-01-03	1.354842	0.313634	1.224231	-0.235177	2.0
2013-01-04	0.885647	-0.297321	-1.628925	0.472148	3.0
2013-01-05	-0.883835	0.699636	0.397156	1.072433	4.0
2013-01-06	0.888353	2.125925	-1.507256	-1.243995	5.0

Setting Data

Setting values by position

In [38]:

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

	A	B	C	D	F
2013-01-01	0.000000	0.000000	2.494974	0.010741	NaN
2013-01-02	0.325493	-0.970221	-0.881164	1.210782	1.0
2013-01-03	1.354842	0.313634	1.224231	-0.235177	2.0
2013-01-04	0.885647	-0.297321	-1.628925	0.472148	3.0
2013-01-05	-0.883835	0.699636	0.397156	1.072433	4.0
2013-01-06	0.888353	2.125925	-1.507256	-1.243995	5.0

Setting Data

Setting by assigning with a numpy array

In [39]:

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

	A	B	C	D	F
2013-01-01	0.000000	0.000000	2.494974	5	NaN
2013-01-02	0.325493	-0.970221	-0.881164	5	1.0
2013-01-03	1.354842	0.313634	1.224231	5	2.0
2013-01-04	0.885647	-0.297321	-1.628925	5	3.0
2013-01-05	-0.883835	0.699636	0.397156	5	4.0
2013-01-06	0.888353	2.125925	-1.507256	5	5.0

Operations

Descriptive statistics

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

In [40]:
```
1 df.mean()
```

```
A    0.428417
B    0.311942
C    0.016503
D    5.000000
F    3.000000
dtype: float64
```

Across axis 1 (columns), i.e., row mean

In [41]:
```
1 df.mean(axis=1)
```

```
2013-01-01    1.873744
2013-01-02    0.894822
2013-01-03    1.978541
2013-01-04    1.391880
2013-01-05    1.842591
2013-01-06    2.301404
Freq: D, dtype: float64
```

Operations

Apply

In [42]:
```
1 df.apply(np.cumsum)
```

	A	B	C	D	F
2013-01-01	0.000000	0.000000	2.494974	5	NaN
2013-01-02	0.325493	-0.970221	1.613810	10	1.0
2013-01-03	1.680336	-0.656587	2.838041	15	3.0
2013-01-04	2.565983	-0.953908	1.209116	20	6.0
2013-01-05	1.682148	-0.254272	1.606272	25	10.0
2013-01-06	2.570501	1.871653	0.099017	30	15.0

In [43]:
```
1 df.apply(lambda x: x.max()-x.min())
```

```
A    2.238677
B    3.096146
C    4.123899
D    0.000000
F    4.000000
dtype: float64
```

Operations

Data distribution

```
In [44]:
1 df['A'].value_counts()
```

```
 0.000000    1
 0.325493    1
 1.354842    1
 0.885647    1
-0.883835    1
 0.888353    1
Name: A, dtype: int64
```

Merge Tables

Join

In [45]:
```
1 left = pd.DataFrame({'key':['foo','bar'], 'lval':[1,2]})
```

In [46]:
```
1 right = pd.DataFrame({'key':['foo', 'bar'], 'rval':[4,5]}
```

In [47]:
```
1 left
```

	key	lval
0	foo	1
1	bar	2

In [48]:
```
1 right
```

	key	rval
0	foo	4
1	bar	5

Merge Tables

Join

In [49]:
```
1 pd.merge(left, right, on='key')
```

	key	lval	rval
0	foo	1	4
1	bar	2	5

Merge Tables

Append

In [50]:
```
df = pd.DataFrame(np.random.randn(8,4),
                  columns=['A', 'B', 'C', 'D'])
df
```

	A	B	C	D
0	0.250835	0.005711	-0.308808	1.321909
1	0.763120	1.994763	-0.921085	-1.089254
2	2.143601	-0.466768	-0.616257	0.413560
3	1.762505	-1.635557	-0.338073	0.347718
4	-0.531461	0.313589	1.760925	-0.193183
5	0.636217	-0.046852	0.357874	-0.505763
6	-1.126569	-1.530734	-1.378126	1.321969
7	1.681342	0.524044	0.613941	-0.392417

In [51]:
```
s = df.iloc[3]
```

Merge Tables

Append

```
In [52]:
1 df.append(s, ignore_index=True)
```

	A	B	C	D
0	0.250835	0.005711	-0.308808	1.321909
1	0.763120	1.994763	-0.921085	-1.089254
2	2.143601	-0.466768	-0.616257	0.413560
3	1.762505	-1.635557	-0.338073	0.347718
4	-0.531461	0.313589	1.760925	-0.193183
5	0.636217	-0.046852	0.357874	-0.505763
6	-1.126569	-1.530734	-1.378126	1.321969
7	1.681342	0.524044	0.613941	-0.392417
8	1.762505	-1.635557	-0.338073	0.347718

Grouping

In [53]:
```
df = pd.DataFrame({'A':['foo','bar',
                        'foo', 'bar',
                        'foo', 'bar',
                        'foo', 'bar'],
                   'B':['one', 'one',
                        'two', 'three',
                        'two', 'two', 'one', 'three'],
                       'C':np.random.randn(8),
                       'D':np.random.randn(8)})
```

In [54]:
```
df
```

	A	B	C	D
0	foo	one	-1.187785	0.095267
1	bar	one	-0.157241	-1.432112
2	foo	two	-0.239852	-0.254292
3	bar	three	-0.437276	-0.131091
4	foo	two	0.520234	0.582490
5	bar	two	-0.014814	-0.890389
6	foo	one	0.659944	0.643184
7	bar	three	-0.917873	0.792226

In [56]:
```
df.groupby('A').sum()
```

A	C	D
bar	-1.527204	-1.661365
foo	-0.247458	1.066650

In [57]:
```
df.groupby(['A','B']).sum()
```

A	B	C	D
bar	one	-0.157241	-1.432112
	three	-1.355149	0.661136
	two	-0.014814	-0.890389
foo	one	-0.527841	0.738452
	two	0.280383	0.328198

CSV

```
In [60]:
1 pd.read_csv('dataset.csv')
```

	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	Year
0	60	8450	7	5	2003	2003
1	20	9600	6	8	1976	1976
2	60	11250	7	5	2001	2002
3	70	9550	7	5	1915	1970
4	60	14260	8	5	2000	2000
...	...	...	...	...	...	...
1455	60	7917	6	5	1999	2000
1456	20	13175	6	6	1978	1988
1457	70	9042	7	9	1941	2006
1458	20	9717	5	6	1950	1996
1459	20	9937	5	6	1965	1965

1460 rows × 17 columns