Data Processing and Analysis in Python
Lecture 14
Tabular Data and pandas

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

DR. ADAM LEE

Pandas

- Panel Data System
- Built for the Python programming language
- An open source, BSD-licensed library (module)
- High-performance, easy-to-use data structures and data analysis tools
- Key components:
  - Series
  - DataFrame

Use Pandas

```
# Import the whole module
>>> import pandas
# Import the whole module using alias
>>> import pandas as pd
# Import the whole module with all libraries
>>> from pandas import *
```

http://pandas.pydata.org/

```
>>> dir(pandas)
>>> dir(pandas.Series)
>>> dir(pandas.DataFrame)
>>> help(pandas)
>>> help(pandas.Series)
>>> help(pandas.DataFrame)
```

UNIVERSITY OF MARYLAND

Data Type: Series

- One-dimensional array like object containing data and labels (or index)
- Lots of ways to build a Series:

```
>>> srs=pd.Series(['a','b','c','d','e','f'])
>>> srs
0    a
1    b
2    c
3    d
4    e
5    f
dtype: object
```

UNIVERSITY OF MARYLAND

Series from Dictionary

- A dict-like container:

```
>>> dct={'a':100,'b':150,'c':200}
>>> srs=pd.Series(dct)
>>> srs
a    100
b    150
c    200
dtype: int64
>>> ses['b']
150
```

UNIVERSITY OF MARYLAND

Series with Index

- A Series index can be specified:
  - Single values can be selected by index
  - Multiple values can be selected with multiple indexes

```
>>> srs=pd.Series([2,4,6],index=['a','b','c'])
>>> srs
a    2
b    4
c    6
dtype: int64
```

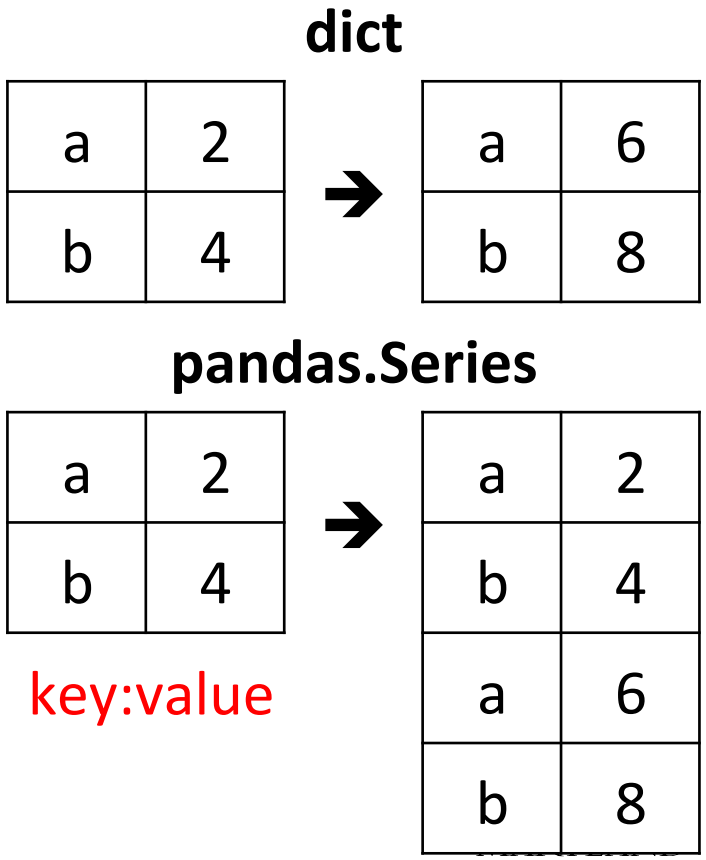
```
>>> srs['b']
4
>>> srs[['b','c']]
b    4
c    6
dtype: int64
```

UNIVERSITY OF MARYLAND

Series versus Dictionary

- Think of a Series as a fixed-length order dictionary
- However, unlike dictionary, index items do not have to be unique:

```
>>> srs=pd.Series([2,4,6,8],
index=['a','b','a','b'])
>>> s['a']
a    2
a    6
dtype: int64
>>> s['b']
b    4
b    8
dtype: int64
```



Data: DataFrame

- Tabular data
- Rectangular data structure, a lot like an array
- Columns (Series) of different types
- Rows and columns act differently
- Can index by (column) labels as well as positions
- Handles missing data (NaN)
- Convenient plotting
- Fast operations with keys
- Lots of facilities for input/output

UNIVERSITY OF MARYLAND

Series to DataFrame

Series

	pop
0	18.8
1	19.1
2	9.7
3	9.7
4	9.8

+

Series

	state
0	FL
1	FL
2	GA
3	GA
4	GA

+

Series

	year
0	2010
1	2011
2	2008
3	2010
4	2011

=

DataFrame

	pop	state	year
0	18.8	FL	2010
1	19.1	FL	2011
2	9.7	GA	2008
3	9.7	GA	2010
4	9.8	GA	2011

index (key)

columns (values)

	A	B	C
1	pop	state	year
2	18.8	FL	2010
3	19.1	FL	2011
4	9.7	GA	2008
5	9.7	GA	2010
6	9.8	GA	2011

Create Data

- Creation with dict of equal-length lists:

```
>>> data={"state":["FL","FL","GA","GA","GA"],
          "year" :[2010,2011,2008,2010,2011],
          "pop"  :[18.8,19.1, 9.7, 9.7, 9.8]}
>>> frame=pd.DataFrame(data)
>>> frame
    pop state  year
0  18.8    FL  2010
1  19.1    FL  2011
2   9.7    GA  2008
3   9.7    GA  2010
4   9.8    GA  2011
```

UNIVERSITY OF MARYLAND

Create Data

- Creation with dict of dicts:

```
>>> pop_data={"FL":{2010:18.8,
                    2011:19.1},
              "GA":{2008: 9.7,
                    2010: 9.7,
                    2011: 9.8}}
>>> pop=pd.DataFrame(pop_data)
>>> pop
        FL   GA
2008   NaN  9.7
2010  18.8  9.7
2011  19.1  9.8
```

UNIVERSITY OF MARYLAND

Create List

- The initial set of baby names and birth rates:

```
>>> names = ["Bob","Jane","Mary","John","Mel"]
>>> births = [968, 155, 77, 578, 973]
>>> BabyDataSet = list(zip(names, births))
>>> BabyDataSet
[('Bob', 968), ('Jane', 155), ('Mary', 77),
('John', 578), ('Mel', 973)]
```

UNIVERSITY OF MARYLAND

Create Data

- Create a Pandas DetaFrame object df:

```
>>> df = pd.DataFrame(data=BabyDataSet,
                      columns=["names","births"])
>>> df
   names  births
0    Bob     968
1   Jane     155
2   Mary      77
3   John     578
4    Mel     973
```

- You can think of df holding the contents of BabyDataSet in a format similar to an SQL table or an Excel spreadsheet

Save Data into File

- `to_csv()` Export DataFrame to a comma-separated values (CSV) file:

```
>>> df.to_csv("filename.csv")
>>> df.to_csv("filename.csv", index=False, header=False)
```

- The file will be saved in the same location as your python file unless specified otherwise

Load Data from File

- `read_csv()` To import from the CSV file:

```
>>> filepath = "filename.csv"
>>> df = pd.read_csv(filepath, header=None)
>>> df
      0    1
0   Bob  968
1  Jane  155
2  Mary   77
3  John  578
4   Mel  973
```

UNIVERSITY OF MARYLAND

Load Data from File

- To give columns specific names, set parameter "names":

```
>>> filepath = "filename.csv"
>>> df = pd.read_csv(filepath, names=["names", "births"])
>>> df
   names  births
0    Bob     968
1   Jane     155
2   Mary      77
3   John     578
4    Mel     973
```

UNIVERSITY OF MARYLAND

Select Data

- `df.head()` returns the first 5 records
- `df.head(n)` returns the first n records
- `df.tail()` returns the last 5 records
- `df.tail(n)` returns the last n records

UNIVERSITY OF MARYLAND

Prepare Data

- To find all unique records of "names" column:

```
>>> df["names"].unique()
array(['Bob', 'Jane', 'Mary', 'John', 'Mel'],
dtype=object)
```

- To obtain all descriptive statistics:

```
>>> df["names"].describe()
count          5
unique         5
top         Jane
freq           1
Name: Names, dtype: object
```

Analyze Data

- To find baby name with the highest birth rate:

```
>>> sorted = df.sort_values(["births"],
ascending=False)
>>> sorted
   names  births
4    Mel     973
0    Bob     968
3   John     578
1   Jane     155
2   Mary      77
>>> sorted.head(1)
   names  births
4    Mel     973
```

Analyze Data

- `df["names"]` returns the entire list of baby names, i.e. the entire "Names" column
- `df["births"]` returns the entire list of births, i.e. the entire "Births" column
- `df["births"].max()` returns the maximum value found in the "Births" column
- `[df["births"] == df["births"].max()]` equals to [Find all records in the "Births" column, where it equals to the maximum, i.e. 973.]
- `df["names"][df["births"] == df["births"].max()]` equals to [Find all records in the "Names" column, where the "Births" column equals to the maximum, i.e. 973.]

UNIVERSITY OF MARYLAND

Column Operations

```
>>> data = range(0,10)
>>> df = pd.DataFrame(list(data))
>>> df.columns = ["num"]
>>> df["new"] = 5
>>> df["new"] = df["new"] + 1
```

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

Column Operations

```
>>> data = range(0,10)
>>> df = pd.DataFrame(list(data))
>>> df.columns = ["num"]
>>> df["three"] = 3
>>> df["col"] = df["num"]
```

	num	three	col
0	0	3	0
1	1	3	1
2	2	3	2
3	3	3	3
4	4	3	4
5	5	3	5
6	6	3	6
7	7	3	7
8	8	3	8
9	9	3	9

Column Operations

```
# ...
>>> ind = ['a','b','c','d','e',
           'f','g','h','i','j']
>>> df.index = ind
```

	num	three	col
a	0	3	0
b	1	3	1
c	2	3	2
d	3	3	3
e	4	3	4
f	5	3	5
g	6	3	6
h	7	3	7
i	8	3	8
j	9	3	9

Column Operations

```
# ...
>>> df.loc['a']
Rev 0
test 3
col 0
Name: a, dtype: int64

# loc[] both inclusive
>>> df.loc['a':'d']

# iloc[] inclusive:exclusive
>>> df.iloc[0:3]
```

	num	three	col
a	0	3	0
b	1	3	1
c	2	3	2
d	3	3	3

	num	three	col
a	0	3	0
b	1	3	1
c	2	3	2

UNIVERSITY OF MARYLAND

Column Operations

```
# ...
>>> df[["num","three"]]


# df.ix[rows, columns]
>>> df.ix[0:3,"num"]
>>> df.ix[5:,"col"]
>>> df.ix[:3,["col","three"]]
```

	num	three
a	0	3
b	1	3
c	2	3

a	0
b	1
c	2

f	5
g	6
h	7
i	8
j	9

	num	three
a	0	3
b	1	3
c	2	3
d	3	3
e	4	3
f	5	3
g	6	3
h	7	3
i	8	3
j	9	3

GroupBy Operations

```
>>> data = {"letter":['a','a','b','b','c'],
            "one":[1,1,1,1,1],
            "two":[2,2,2,2,2]}
>>> df = pd.DataFrame(data)
# create group object
>>> letter = df.groupby("letter")
# apply sum
>>> letter.sum()
```

	one	two
letter		
a	2	4
b	2	4
c	1	2

	letter	one	two
0	a	1	2
1	a	1	2
2	b	1	2
3	b	1	2
4	c	1	2

GroupBy Operations

```
>>> letter_one =
df.groupby(["letter","one"]).sum()
>>> letter_one =
df.groupby(["letter","one"],as_index=False).sum
()
```

	letter	one	two
0	a	1	4
1	b	1	4
2	c	1	2

		two
letter	one	
a	1	4
b	1	4
c	1	2

Missing Values: numpy.nan and pandas.NA

```
>>> import numpy as np
>>> df = pd.DataFrame(np.random.randn(5,3),
        index=['a','c','e','f','h'],
        columns=["one","two","three"])
>>> df.loc['a',"two"] = np.nan
>>> df
        one       two     three
a  0.405857       NaN  2.291062
c -0.491699  1.178464 -1.951439
e  2.649431 -1.938993 -0.827796
f  1.235505 -0.726769 -0.146558
h  2.837826  0.505976 -0.489474
```

UNIVERSITY OF MARYLAND

Detect Missing Values

- `pandas.isnull()` Detect missing values for an array-like object
- `pandas.Series.isnull()` Detect missing values in a Series
- `pandas.DataFrame.isnull()` Detect missing values in a DataFrame

```
>>> df.isnull()
     one    two  three
a  False   True  False
c  False  False  False
e  False  False  False
f  False  False  False
h  False  False  False
```

UNIVERSITY OF MARYLAND

Fill Missing Values

```
>>> df.fillna(0)
          one       two     three
a  0.405857  0.000000  2.291062
c -0.491699  1.178464 -1.951439
e  2.649431 -1.938993 -0.827796
f  1.235505 -0.726769 -0.146558
h  2.837826  0.505976 -0.489474
```

UNIVERSITY OF MARYLAND

Query Data

- Use the **query(expression)** function to embed boolean expressions on columns within quotes:

```
>>> df.query("one>0")
        one       two     three
a  0.405857       NaN  2.291062
e  2.649431 -1.938993 -0.827796
f  1.235505 -0.726769 -0.146558
h  2.837826  0.505976 -0.489474

>>> df.query("one>0 & two>0")
        one       two     three
h  2.837826  0.505976 -0.489474
```

Query Data

- You can apply any function to the columns:

```
>>> df.apply(lambda x: x.max()-x.min())
one      3.329524
two      3.117457
three    4.242501
dtype: float64
```

- You can apply any function to element-wise:

```
>>> df.applymap(np.sqrt)
        one       two     three
a  0.637069       NaN  1.513626
c       NaN  1.085571       NaN
e  1.627707       NaN       NaN
f  1.111533       NaN       NaN
h  1.684585  0.711320       NaN
```

UNIVERSITY OF MARYLAND

Pandas - Visualization

http://pandas.pydata.org/pandas-docs/stable/visualization.html

- Use the standard convention for referencing the matplotlib API
- Provide the basics in pandas to easily create decent looking plots

UNIVERSITY OF MARYLAND