



Pandas

CHULA Σ ENGINEERING
Foundation toward Innovation

COMPUTER

Chula Big Data and IoT
Center of Excellence
(CUBIC)



Pandas Python for Data Analytics

Peerapon Vateekul, Ph.D.

Department of Computer Engineering,
Faculty of Engineering, Chulalongkorn University

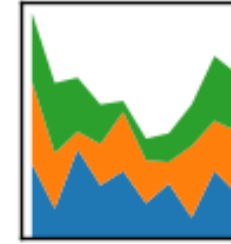
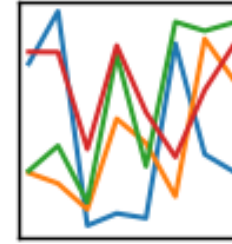
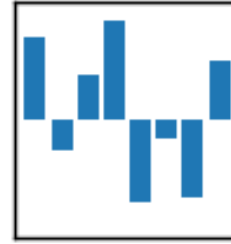
Peerapon.v@chula.ac.th



Outlines

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



- Python Data Analysis Library
- Viewing and Inspecting Data
- Selection of Data
- Filter, Sort and Groupby
- Data Cleaning
- Join/Combine
- Series
- DataFrame

- Axis indexing, the special pandas-flavored sauce
- Data alignment
- GroupBy
- Hierarchical indexes
- pandas.core
- The pandas roadmap
- pandas for “Big Data”
- Summary

Reference:

(1)<http://pandas.pydata.org>,

(2) <https://medium.com/@adi.bronshtein/a-quick-introduction-to-the-pandas-python-library-flb678f34673>

(3) Wes McKinney Lecture, pandas: Powerful data analysis tools for Python

+ Python Data Analysis Library

3

- *pandas* is an open source, BSD-licensed library providing high-performance, [easy-to-use data structures](#) and data analysis tools for the [Python](#) programming language.
- *pandas* is a [NumFOCUS](#) sponsored project. This will help ensure the success of development of *pandas* as a world-class open-source project, and makes it possible to [donate](#) to the project



VERSIONS

Latest version

Release

0.23.1 - June 2018

[download](#) // [docs](#) // [pdf](#)

Development

0.24.0 - 2018

[github](#) // [docs](#)



pandas.core

- Data structures
 - Series (1D)
 - DataFrame (2D)
 - Panel (3D)
- NA-friendly statistics
- Index implementations / label-indexing
- GroupBy engine
- Time series tools
- Date range generation
- Extensible date offsets
- Hierarchical indexing stuff
- Join / concatenation algorithms
- Sparse versions of Series, DataFrame...
- IO tools: CSV files, HDF5, Excel 2003/2007
- Moving window statistics (rolling mean, ...)
- Pivot tables
- High level matplotlib interface



Loading and Saving Data with Pandas

- When you want to use Pandas for data analysis, you'll usually use it in one of three different ways:
- Convert a Python's list, dictionary or Numpy array to a Pandas data frame
- Open a local file using Pandas, usually a CSV file, but could also be a delimited text file (like TSV), Excel, etc
- Open a remote file or database like a CSV or a JSON on a website through a URL or read from a SQL table/database
- There are different commands to each of these options, but when you open a file, they would look like this:

```
pd.read_filetype()
```

```
pd.read_csv()
```

```
pd.read_excel()
```



Viewing and Inspecting Data

- Now that you've loaded your data, it's time to take a look.
- How does the data frame look? Running the name of the data frame would give you the entire table, but you can also get the first n rows with **df.head(n)** or the last n rows with **df.tail(n)**.
- **df.shape** would give you the number of rows and columns.
- **df.info()** would give you the index, datatype and memory information.
- The command **s.value_counts(dropna=False)** would allow you to view unique values and counts for a series (like a column or a few columns).
- A very useful command is **df.describe()** which inputs summary statistics for numerical columns.



Viewing and Inspecting Data (cont.)

- It is also possible to get **statistics** on the entire data frame or a series (a column, etc.):
- **df.mean()** -- Returns the mean of all columns
- **df.corr()** -- Returns the correlation between columns in a data frame
- **df.count()** -- Returns the number of non-null values in each data frame column
- **df.max()** -- Returns the highest value in each column
- **df.min()** -- Returns the lowest value in each column
- **df.median()** -- Returns the median of each column
- **df.std()** -- Returns the standard deviation of each column



Selection of Data

- One of the things that is so much easier in Pandas is selecting the data you want in comparison to selecting a value from a list or a dictionary.
- You can select a column (**df[col]**) and return column with label col as Series or a few columns (**df[[col1, col2]]**) and returns columns as a new DataFrame.
- You can select by position (**s.iloc[0]**), or by index (**s.loc['index_one']**).
- In order to select the first row you can use **df.iloc[0,:]** and in order to select the first element of the first column you would run **df.iloc[0,0]**.
- These can also be used in different combinations, so I hope it gives you an idea of the different selection and indexing you can perform in Pandas.



Filter, Sort and Group by

- You can use different conditions to filter columns. For example, `df[df[year] > 1984]` would give you only the column year is greater than 1984.
- You can use `&` (and) or `|` (or) to add different conditions to your filtering.
 - These is also called *boolean filtering*.
- It is possible to sort values in a certain column in an ascending order using **`df.sort_values(coll)`** ; and also in a descending order using **`df.sort_values(col2,ascending=False)`**.
- Furthermore, it's possible to sort values by `coll` in ascending order then `col2` in descending order by using **`df.sort_values([coll,col2],ascending=[True,False])`**.



Filter, Sort and Group by (cont.)

- The last command in this section is groupby.
- It involves splitting the data into groups based on some criteria, applying a function to each group independently and combining the results into a data structure.
- `df.groupby(col)` returns a groupby object for values from one column.
- While **`df.groupby([col1,col2])`** returns a groupby object for values from multiple columns.



Data Cleansing

- Data cleansing is a very important step in data analysis.
- For example, we always check for missing values in the data by running **pd.isnull()** which checks for null Values, and returns a boolean array (an array of *true* for missing values and *false* for non-missing values).
- In order to get a sum of null/missing values, run **pd.isnull().sum()**. **pd.notnull()** is the opposite of **pd.isnull()**.
- After you get a list of missing values you can get rid of them, or drop them by using **df.dropna()** to drop the rows or **df.dropna(axis=1)** to drop the columns.

+ Data Cleansing (cont.)

- A different approach would be to fill the missing values with other values by using **df.fillna(x)** which fills the missing values with x
- you can put there whatever you want or **s.fillna(s.mean())** to replace all null values with the mean
 - mean can be replaced with almost any function from the statistics section.
- It is sometimes necessary to replace values with different values.
- For example, **s.replace(1,'one')** would replace all values equal to 1 with 'one'.
 - It's possible to do it for multiple values: **s.replace([1,3],['one','three'])** would replace all 1 with 'one' and 3 with 'three'.
- You can also rename specific columns by running: **df.rename(columns={'old_name': 'new_name'})** or use **df.set_index('student_id')** to change the index of the data frame (PK).



Join/Combine

- The last set of basic Pandas commands are for joining or combining data frames or rows/columns. The three commands are:
- **df1.append(df2);** add the rows in df1 to the end of df2 (columns should be identical)
- **df.concat([df1, df2],axis=1);** add the columns in df1 to the end of df2 (rows should be identical)
- **df1.join(df2,on=col,how='inner');** SQL-style join the columns in df1 with the columns on df2 where the rows for col have identical values. how can be equal to one of: 'left', 'right', 'outer', 'inner'

+ Data Structures in Pandas

- *1) Series (1D)
- *2) DataFrame (2D)
- 3) Panel (3D)



Series

- Subclass of `numpy.ndarray`
- Data: any type
- Index labels need not be ordered
- Duplicates are possible (but result in reduced functionality)

index **values**

A	→	5
B	→	6
C	→	12
D	→	-5
E	→	6.7



Series (cont.)

CODE

```
import pandas as pd  
sr = pd.Series([5,6,12,-5,6.7], index=['A', 'B', 'C', 'D', 'E'])
```

sr

A 5.0

B 6.0

C 12.0

D -5.0

E 6.7

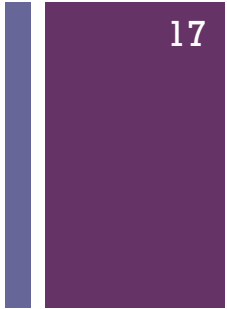
dtype: float64



DataFrame

- NumPy array-like
- Each column can have a different type
- Row and column index
- Size mutable: insert and delete columns

Column				
Index	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509



+ DataFrame (cont.) code

```
df = pd.DataFrame(randn(5,4),index='A B C D E'.split()\  
                  ,columns='W X Y Z'.split())
```

Index

Column

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

+ Data alignment

- Binary operations are joins!

DF: left

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

DF: right

	C	D
K2	C2	D2
K3	C3	D3

Code: 1

```
left.join(right)
```

	A	B	C	D
K0	A0	B0	NaN	NaN
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2

Code: 2

```
left.join(right, how='outer')
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2
K3	NaN	NaN	C3	D3



Group by

DF Example

	Company	Person	Sales
0	GOOG	Sam	200
1	GOOG	Charlie	120
2	MSFT	Amy	340
3	MSFT	Vanessa	124
4	FB	Carl	243
5	FB	Sarah	350



Code

```
df.groupby('Company')
```

```
<pandas.core.groupby.DataFrameGroupBy object at 0x7f5d8a495400>
```

```
by_comp = df.groupby("Company")
```

```
by_comp.mean()
```



Sales	
Company	
FB	296.5
GOOG	160.0
MSFT	232.0

Output



Pandas Summary

- A fast and efficient **DataFrame** object for data manipulation with integrated indexing;
- Tools for **reading and writing data** between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format;
- Intelligent **data alignment** and integrated handling of **missing data**: gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form;
- Flexible **reshaping** and pivoting of data sets;



Summary: Library Highlights (cont.)

- Intelligent label-based **slicing**, **fancy indexing**, and **subsetting** of large data sets;
- Columns can be inserted and deleted from data structures for **size mutability**;
- Aggregating or transforming data with a powerful **group by** engine allowing split-apply-combine operations on data sets;
- High performance **merging and joining** of data sets;
- **Hierarchical axis indexing** provides an intuitive way of working with high-dimensional data in a lower-dimensional data structure;

Summary: Library Highlights (cont.)

- **Time series**-functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data;
- Highly **optimized for performance**, with critical code paths written in [Cython](#) or C.
- Python with *pandas* is in use in a wide variety of **academic and commercial** domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.



Lab: Pandas

Data Preparation Full Package

Part 1: Data Input and Output

CSV

CSV Output

Excel

Excel Input

Excel Output

Part 2: DataFrames

Selection and Indexing

Conditional Selection

More Index Details

Multi-Index and Index Hierarchy

Part3: Missing Data

Part4: Group by

Part5: Merging, Joining, and Concatenating

Concatenation

Merging

Joining

Part6: Operations

Info on Unique Values

Selecting Data

Applying Functions

+



Any Questions?