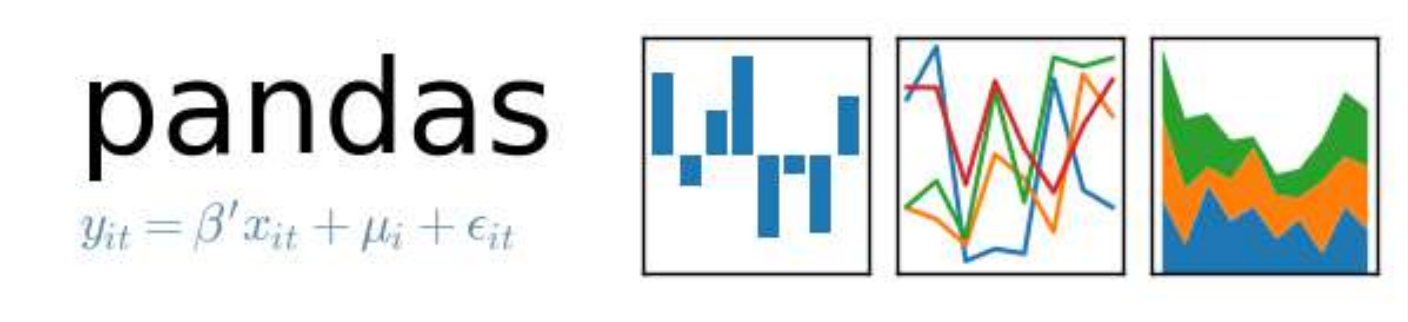
Pandas

Peerapon Vateekul, Ph.D.

peerapon.v@4amconsult.com

+ Outlines



- **Pandas : Python Data Analysis Library**
- **Pandas Structure**
  - Series
  - DataFrame
- **Pandas Functions**
  - Creating Pandas
  - Viewing and Inspecting Data
  - Filtering and Sorting
  - Basic Statistic
  - Grouping
  - Pivoting and Melting
  - Data Cleansing
  - Appending and Merging
- **SQL Query in Pandas**
- **Pandas Summary**

Reference:
(1)http://pandas.pydata.org,
(2) https://medium.com/@adi.bronshtein/a-quick-introduction-to-the-pandas-python-library-f1b678f34673
(3) Wes McKinney Lecture, pandas: Powerful data analysis tools for Python

+ Outlines

- **Pandas : Python Data Analysis Library**
- **Pandas Structure**
  - Series
  - DataFrame
- **Pandas Functions**
  - Creating Pandas
  - Viewing and Inspecting Data
  - Filtering and Sorting
  - Basic Statistic
  - Grouping
  - Pivoting and Melting
  - Data Cleansing
  - Appending and Merging
- **SQL Query in Pandas**
- **Pandas Summary**

Reference:
(1)http://pandas.pydata.org,
(2) https://medium.com/@adi.bronshtein/a-quick-introduction-to-the-pandas-python-library-f1b678f34673
(3) Wes McKinney Lecture, pandas: Powerful data analysis tools for Python

Pandas : Python Data Analysis Library

- *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

A Fiscally Sponsored Project of
NUMFOCUS
OPEN CODE = BETTER SCIENCE

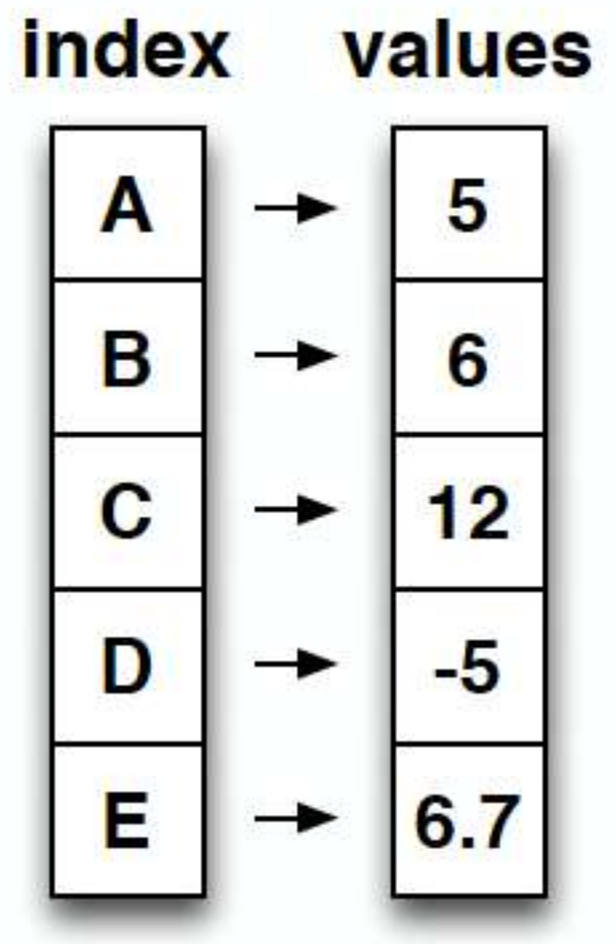
+ Outlines

- **Pandas : Python Data Analysis Library**
- **Pandas Structure**
  - Series
  - DataFrame
- **Pandas Functions**
  - Creating Pandas
  - Viewing and Inspecting Data
  - Filtering and Sorting
  - Basic Statistic
  - Grouping
  - Pivoting and Melting
  - Data Cleansing
  - Appending and Merging
- **SQL Query in Pandas**
- **Pandas Summary**

Reference:
(1)http://pandas.pydata.org,
(2) https://medium.com/@adi.bronshtein/a-quick-introduction-to-the-pandas-python-library-f1b678f34673
(3) Wes McKinney Lecture, pandas: Powerful data analysis tools for Python

Pandas Structure - Series

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



CODE

```
# create Series from a list
s1 = pd.Series([14, -8, 0, 3, 9])
s1
```

```
0    14
1    -8
2     0
3     3
4     9
dtype: int64
```

Pandas Structure - 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

Pandas Structure - DataFrame

```
# Create From a Dict
data = {
    'province': ['Chiang Mai', 'Chiang Mai', 'Chiang Mai', 'Phrae', 'Phrae', 'Phrae'],
    'year': [2016, 2017, 2018, 2016, 2017, 2018],
    'population': [1630428, 1664012, 1687971, 398936, 410382, 421653]
}
df = pd.DataFrame(data)
df
```

Column

	province	year	population
0	Chiang Mai	2016	1630428
1	Chiang Mai	2017	1664012
2	Chiang Mai	2018	1687971
3	Phrae	2016	398936
4	Phrae	2017	410382
5	Phrae	2018	421653

Index

+ Outlines

- **Pandas : Python Data Analysis Library**
- **Pandas Structure**
  - Series
  - DataFrame
- **Pandas Functions**
  - Creating Pandas
  - Viewing and Inspecting Data
  - Filtering and Sorting
  - Basic Statistic
  - Grouping
  - Pivoting and Melting
  - Data Cleansing
  - Appending and Merging
- **SQL Query in Pandas**
- **Pandas Summary**

Reference:
(1)http://pandas.pydata.org,
(2) https://medium.com/@adi.bronshtein/a-quick-introduction-to-the-pandas-python-library-f1b678f34673
(3) Wes McKinney Lecture, pandas: Powerful data analysis tools for Python

Pandas Functions

- Creating Pandas
  - Import/Export with list
  - Import/Export with file csv, excel, sas
  - Import/Export with database
- Viewing and Inspecting Data
- Filtering and Sorting
- Basic Statistic
- Grouping
- Pivoting and Melting
- Data Cleansing
- Appending and Merging

Creating Pandas - Pandas <-> List

- Create Pandas from List
  - pd.dataFrame(data, columns)
- Convert Pandas to List
  - df.values.tolist()

```
# Create From a List
data = [
    ['Chiang Mai', 2016, 1630428],
    ['Chiang Mai', 2017, 1664012],
    ['Chiang Mai', 2018, 1687971],
    ['Phrae', 2016, 398936],
    ['Phrae', 2017, 410382],
    ['Phrae', 2018, 421653]
]
df = pd.DataFrame(
    data=data,
    columns=['province','year','population']
)
df
```

	province	year	population
0	Chiang Mai	2016	1630428
1	Chiang Mai	2017	1664012
2	Chiang Mai	2018	1687971
3	Phrae	2016	398936
4	Phrae	2017	410382
5	Phrae	2018	421653

```
# Pandas to List
df.values.tolist()
```

```
[['Chiang Mai', 2016, 1630428.0],
 ['Chiang Mai', 2017, 1664012.0],
 ['Chiang Mai', 2018, 1687971.0],
 ['Phrae', 2016, nan],
 ['Phrae', 2017, 410382.0],
 [None, 2018, 421653.0]]
```

Creating Pandas
- Pandas <->File

- 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 JSONon 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()

Creating Pandas - Pandas <-> File

- **Read/Write CSV**
  - pd.read_csv(filepath, sep=‘,’)
  - df.to_csv(filepath, sep=‘,’)
- **Read/Write Excel**
  - pd.read_excel(filepath, sheet_name)
  - df.to_excel(filepath, sheet_name)
- **Write Multiple DF to same Excel**
  - with pd.ExcelWriter('output.xlsx') as writer:

    df1.to_excel(writer, sheet_name='Sheet_name_1’)

    df2.to_excel(writer, sheet_name='Sheet_name_2')
- **Read/Write SAS**
  - pd.read_sas(filepath)
  - SAS7BDAT is a closed file format, and not intended to be read/written to by other languages

```
df = pd.read_csv('sample.csv')
df
```

	province	year	population
0	Chiang Mai	2016	1630428
1	Chiang Mai	2017	1664012
2	Chiang Mai	2018	1687971
3	Phrae	2016	398936
4	Phrae	2017	410382
5	Phrae	2018	421653

```
df = pd.read_excel('sample.xlsx', sheet_name='sheet1')
df
```

	province	year	population
0	Chiang Mai	2016	1630428
1	Chiang Mai	2017	1664012
2	Chiang Mai	2018	1687971
3	Phrae	2016	398936
4	Phrae	2017	410382
5	Phrae	2018	421653

Creating Pandas - Pandas <-> Database

- **Read Database**
  - pd.read_sql(sql, con)
  - pd.read_sql_table(table_name, con)
- **Write Database**
  - df.to_sql(table_name, con, if_exists})
    - if_exists={'fail', 'replace', 'append'}
      - default 'fail'
      - How to behave if the table already exists.
      - fail: Raise a ValueError.
      - replace: Drop the table before inserting new values.
      - append: Insert new values to the existing table.
- *** con = SQL Alchemy Connection or Database URI

Creating Pandas - Pandas <-> Database

- Example

```
import pandas as pd
import mysql.connector
from sqlalchemy import create_engine

#Create con
engine = create_engine(
    'mysql+mysqlconnector://[user]:[pass]@[host]:[port]/[schema]’,
    echo=False
)

#Insert
data.to_sql(
    name='sample_table’,
    con=engine,
    if_exists = 'append’,
    index=False
)
```

Viewing and Inspecting Data - Row

- View head and tail
  - df.head(n)
  - df.tail(n)
- Sample
  - df.sample(n, replace=False, random_state)
  - df.sample(fraction, replace=False, random_state)
- Get some rows
  - df.iloc[position]
  - df.loc[index_name]

```
df.head(3)
```

	province	year	population
0	Chiang Mai	2016	1630428.0
1	Chiang Mai	2017	1664012.0
2	Chiang Mai	2018	1687971.0

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

	province	year	population
3	Phrae	2016	NaN
4	Phrae	2017	410382.0
5	None	2018	421653.0

Viewing and Inspecting Data - Column

- Get column name
  - df.columns
- Select columns
  - df[columns_list]
- Drop columns
  - df.drop(columns=colums_list)
- Rename columns
  - df.rename(columns=mapper)
    - Mapper : Dict-like or functions transformations to apply to that axis' values
    - Example : {'c1': 'col1', 'c2: 'col2'}

```
df.columns
```

```
Index(['province', 'year', 'population'], dtype='object')
```

```
columns_list = ['year','population']
df[columns_list]
```

	year	population
0	2016	1630428
1	2017	1664012
2	2018	1687971
3	2016	398936
4	2017	410382
5	2018	421653

Viewing and Inspecting Data - Apply

- Create/Replace column with fix value or array
  - df['new_column'] = 1
  - df['new_column'] = value_list

- Create/Replace column with apply
  - df['x2'] = df['x'].apply(lambda x : x**2)

```
df['source'] = 'A'
df
```

	province	year	population	source
0	Chiang Mai	2016	1630428	A
1	Chiang Mai	2017	1664012	A
2	Chiang Mai	2018	1687971	A
3	Phrae	2016	398936	A
4	Phrae	2017	410382	A
5	Phrae	2018	421653	A

```
df['source'] = ['A','B','A','C','A','B']
df
```

	province	year	population	source
0	Chiang Mai	2016	1630428	A
1	Chiang Mai	2017	1664012	B
2	Chiang Mai	2018	1687971	A
3	Phrae	2016	398936	C
4	Phrae	2017	410382	A
5	Phrae	2018	421653	B

Viewing and Inspecting Data - DataFrame Information

- **df.shape** would give you the number of rows and columns.
- **df.info()** would give you the index, datatype and memory information.
- **df.dtypes** would give datatypes of each columns

```
df.shape
```

```
(6, 5)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   province        6 non-null      object
 1   year            6 non-null      int64
 2   population      6 non-null      int64
 3   source          6 non-null      object
 4   population (K)  6 non-null      float64
dtypes: float64(1), int64(2), object(2)
memory usage: 368.0+ bytes
```

```
df.dtypes
```

```
province           object
year                int64
population          int64
source             object
population (K)    float64
dtype: object
```

Filtering and Sorting

- Filter
  - df[condition]

```
condition_list = [True,False,True,False,True,False]
df[condition_list]
```

	province	year	population	source	population (K)
0	Chiang Mai	2016	1630428	A	1630.428
2	Chiang Mai	2018	1687971	A	1687.971
4	Phrae	2017	410382	A	410.382

- Sort Ascending
  - df.sort_values(by=col1)
- Sort Descending
  - df.sort_values(by=col1, ascending=False)
- Sort Multiple columns
  - df.sort_values(by=[col1,col2], ascending=[True,False])

```
df.sort_values(by=['year','population'], ascending=[True,False])
```

	province	year	population	source	population (K)
0	Chiang Mai	2016	1630428	A	1630.428
3	Phrae	2016	398936	C	398.936
1	Chiang Mai	2017	1664012	B	1664.012
4	Phrae	2017	410382	A	410.382
2	Chiang Mai	2018	1687971	A	1687.971
5	Phrae	2018	421653	B	421.653

Basic Statistic - Numeric

- 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.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
  - **df.corr() --** Returns the correlation between columns in a data frame

```
# Find Mean in all numeric columns
df.mean()
```

```
year              2.017000e+03
population        1.035564e+06
population (K)    1.035564e+03
dtype: float64
```

```
# Find Mean in some column
df['population'].mean()
```

```
1035563.6666666666
```

+ Basic Statistic - Category

- Distinct and Count Distinct
  - df[column_name].unique()
  - df[column_name].nunique()
  - df[column_name].values_count()

```
df['province'].unique()
```

```
array(['Chiang Mai', 'Phrae'], dtype=object)
```

```
df['province'].nunique()
```

```
2
```

```
df['province'].value_counts()
```

```
Chiang Mai    3
Phrae         2
Name: province, dtype: int64
```

Basic Statistic - Describe

- Describe
  - Numeric
    - df[column_list].describe()
  - Category
    - df[column_list].describe(include=[np.object])
  - All
    - df[column_list].describe(include='all')

```
#Numeric
df[columns_list].describe()
```

	year	population
count	6.000000	6.000000e+00
mean	2017.000000	1.035564e+06
std	0.894427	6.851977e+05
min	2016.000000	3.989360e+05
25%	2016.250000	4.131998e+05
50%	2017.000000	1.026040e+06
75%	2017.750000	1.655616e+06
max	2018.000000	1.687971e+06

```
#Category
df.describe(include=np.object)
```

	province	source
count	6	6
unique	2	3
top	Phrae	A
freq	3	3

+ Grouping

- Count
- Sum
- Mean
- Median
- Std
- Min, Max
- First, Last

	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

```
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

Grouping

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

Grouping with Window

- df.groupby(col).rolling(window, min_periods=window)

Example : df.groupby('Stock')['Price'].rolling(window=3, min_periods=3)

Stock	Date	Price
APPLE	1-1-2020	100
APPLE	2-1-2020	120
APPLE	3-1-2020	125
APPLE	4-1-2020	130
APPLE	5-1-2020	150

Stock	Date	AVG_Price(3)
APPLE	1-1-2020	NaN
APPLE	2-1-2020	NaN
APPLE	3-1-2020	(100+120+125)/3=115
APPLE	4-1-2020	(120+125+130)/3=125
APPLE	5-1-2020	(120+125+150)/3=135

Pivoting and Melting - Pivot

	province	year	population	source	population (K)	population (M)
0	Chiang Mai	2016	1630428	A	1630.428	1.630428
1	Chiang Mai	2017	1664012	B	1664.012	1.664012
2	Chiang Mai	2018	1687971	A	1687.971	1.687971
3	Phrae	2016	398936	C	398.936	0.398936
4	Phrae	2017	410382	A	410.382	0.410382
5	Phrae	2018	421653	B	421.653	0.421653

```
pivot = df.pivot_table(index=['province'], columns=['year'], values=['population'])
pivot
```

	population		
year	2016	2017	2018
province			
Chiang Mai	1630428	1664012	1687971
Phrae	398936	410382	421653

Pivoting and Melting - Melt

	province	2016	2017	2018
0	Chiang Mai	1630428	1664012	1687971
1	Phrae	398936	410382	421653

```
melt = pd.melt(pivot, id_vars=['province'], value_vars=['2016','2017','2018'])
melt
```

	province	variable	value
0	Chiang Mai	2016	1630428
1	Phrae	2016	398936
2	Chiang Mai	2017	1664012
3	Phrae	2017	410382
4	Chiang Mai	2018	1687971
5	Phrae	2018	421653

Data Cleansing

- Check Null
  - df.isnull()
  - df.isnull().sum()
- Drop missing row
  - df.dropna()
- Impute Missing
  - df.fillna()
  - df.fillna(mapper)

```
df
```

	province	year	population
0	Chiang Mai	2016	1630428.0
1	Chiang Mai	2017	1664012.0
2	Chiang Mai	2018	1687971.0
3	Phrae	2016	NaN
4	Phrae	2017	410382.0
5	None	2018	421653.0

```
df.fillna('Missing')
```

	province	year	population
0	Chiang Mai	2016	1.63043e+06
1	Chiang Mai	2017	1.66401e+06
2	Chiang Mai	2018	1.68797e+06
3	Phrae	2016	Missing
4	Phrae	2017	410382
5	Missing	2018	421653

```
mapper = {
    'province' : 'Unknown',
    'population' : df['population'].mean()
}
print(mapper)
display(df.fillna(mapper))
```

```
{'province': 'Unknown', 'population': 1162889.2}
```

	province	year	population
0	Chiang Mai	2016	1630428.0
1	Chiang Mai	2017	1664012.0
2	Chiang Mai	2018	1687971.0
3	Phrae	2016	1162889.2
4	Phrae	2017	410382.0
5	Unknown	2018	421653.0

Appending and Merging

- Append (Row)
  - df1.append(df2)
  - pd.concat(df_list)
- Append (Column)
  - pd.concat(df_list, axis=1)
- Join
  - df1.join(df2, on=col, how='inner')
  - df1.merge(df2, on=col, how='inner')
    - how{'left', 'right', 'outer', 'inner'}, default 'inner'
    - The **merge** method is more versatile **and** allows us to specify columns besides the index to **join** on for both dataframes.

```
df = df1.append(df2)
df
```

	province	year	population
0	Chiang Mai	2016	1630428
1	Chiang Mai	2017	1664012
2	Chiang Mai	2018	1687971
0	Phrae	2016	398936
1	Phrae	2017	410382
2	Phrae	2018	421653

```
df = pd.concat([df1, df2])
df
```

	province	year	population
0	Chiang Mai	2016	1630428
1	Chiang Mai	2017	1664012
2	Chiang Mai	2018	1687971
0	Phrae	2016	398936
1	Phrae	2017	410382
2	Phrae	2018	421653

Reference : https://towardsdatascience.com/pandas-join-vs-merge-c365fd4fbf49

Appending and Merging

- 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

+ Outlines

- **Pandas : Python Data Analysis Library**
- **Pandas Structure**
  - Series
  - DataFrame
- **Pandas Functions**
  - Creating Pandas
  - Viewing and Inspecting Data
  - Filtering and Sorting
  - Basic Statistic
  - Grouping
  - Pivoting and Melting
  - Data Cleansing
  - Appending and Merging
- **SQL Query in Pandas**
- **Pandas Summary**

Reference:
(1)http://pandas.pydata.org,
(2) https://medium.com/@adi.bronshtein/a-quick-introduction-to-the-pandas-python-library-f1b678f34673
(3) Wes McKinney Lecture, pandas: Powerful data analysis tools for Python

SQL Query in Pandas

- Use Pandasql package

DataFrame variable

```
import pandasql
```

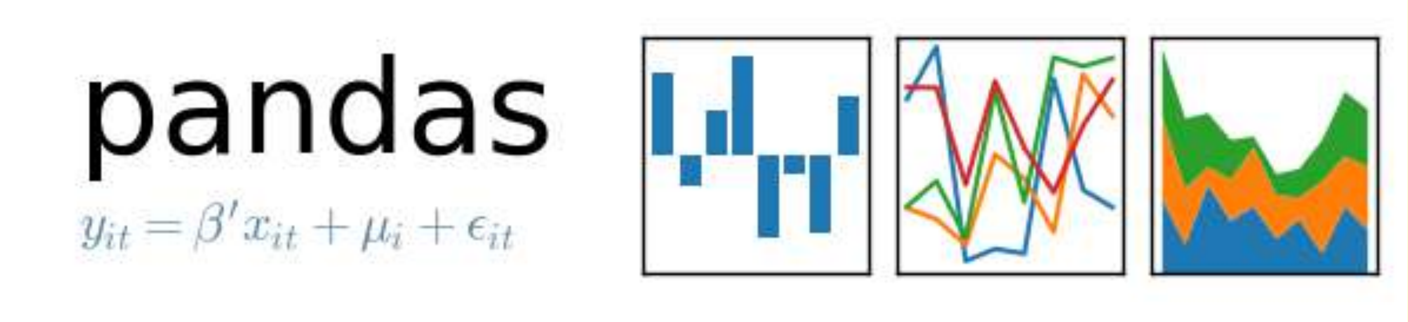
```
sql_df = pandasql.sqldf("SELECT * FROM df ", globals())
sql_df
```

	province	year	population
0	Chiang Mai	2016	1630428
1	Chiang Mai	2017	1664012
2	Chiang Mai	2018	1687971
3	Phrae	2016	398936
4	Phrae	2017	410382
5	Phrae	2018	421653

```
sql_df = pandasql.sqldf("select province, avg(population) from df group by province;", globals())
sql_df
```

	province	avg(population)
0	Chiang Mai	1.660804e+06
1	Phrae	4.103237e+05

+ Outlines



- **Pandas : Python Data Analysis Library**
- **Pandas Structure**
  - Series
  - DataFrame
- **Pandas Functions**
  - Creating Pandas
  - Viewing and Inspecting Data
  - Filtering and Sorting
  - Basic Statistic
  - Grouping
  - Pivoting and Melting
  - Data Cleansing
  - Appending and Merging
- **SQL Query in Pandas**
- **Pandas Summary**

Reference:
(1)http://pandas.pydata.org,
(2) https://medium.com/@adi.bronshtein/a-quick-introduction-to-the-pandas-python-library-f1b678f34673
(3) Wes McKinney Lecture, pandas: Powerful data analysis tools for Python

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.

+

Any Questions?