Data Analytics Course - Lesson 01

Ths. Vu Duy Khuong

Agenda

- I. Pandas Library
- II. Import Dataset
- III. Export Dataset
- IV. Summary Dataset

I. Pandas Library

1. Pandas Libraries

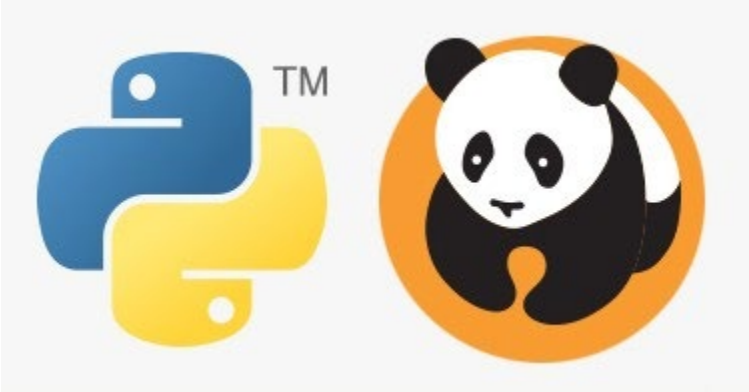
- The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.

- Install pandas by following command:

```
$ pip install pandas
```

- Use the following import convention:

```
>>> import pandas as pd
```



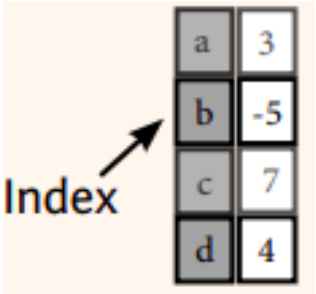
I. Pandas Library

2. Pandas Data Structures

- Series

A one-dimensional labeled array a capable of holding any data type.

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```



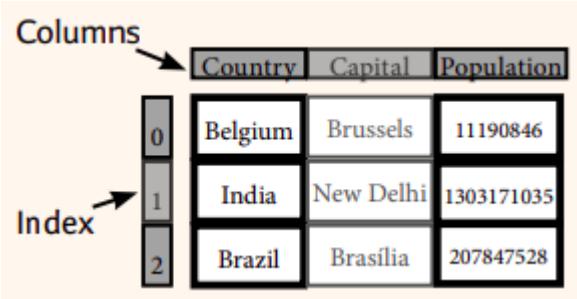
I. Pandas Library

2. Pandas Data Structures

- *DataFrame*

A two-dimensional labeled data structure with columns of potentially different types.

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}
>>> df = pd.DataFrame(data, columns=['Country', 'Capital', 'Population'])
```



I. Pandas Library

3. Selection

- Getting

```
>>> s['b'] # Get one element
   -5
>>> df[1:] # Get subset of a DataFrame
   Country   Capital   Population
 1 India     New Delhi  1303171035
 2 Brazil    Brasília   207847528
```

I. Pandas Library

3. Selection

- Selecting, Boolean Indexing & Setting

- By Position

```
>>> df.iloc([0],[0]) # Select single value by row & column
   'Belgium'
>>> df.iat([0],[0])  # # Select single value by row & column
   'Belgium'
```

- By Label

```
>>> df.loc([0], ['Country']) # Select single value by row & column labels
   'Belgium'
>>> df.at([0], ['Country'])
   'Belgium'
```

3. Selection

- Selecting, Boolean Indexing & Setting

- By Label/Position

```
>>> df.ix[2]  # Select single row of subset of rows
    Country Brazil
    Capital Brasília
    Population 207847528

>>> df.ix[:,'Capital'] # Select a single column of subset of columns
    0 Brussels
    1 New Delhi
    2 Brasília

>>> df.ix[1,'Capital'] # Select rows and columns
    'New Delhi'
```

I. Pandas Library

3. Selection

- Selecting, Boolean Indexing & Setting

- Boolean Indexing

```
>>> s[~(s > 1)] # Series s where value is not >1
>>> s[(s < -1) | (s > 2)] # Series s where value is <-1 or >2
>>> df[df['Population']>1200000000] #  Use filter to adjust DataFrame
```

- Setting

```
>>> s['a'] = 6 # Set index a of Series s to 6
```

I. Pandas Library

4. Dropping

```
>>> s.drop(['a', 'c']) # Drop values from rows (axis=0)
>>> df.drop('Country', axis=1) # Drop values from columns(axis=1)
```

5. Sort

```
>>> df.sort_index(ascending=True) # Sort by index values
>>> df.sort_values(by='Country', ascending=True) # Sort by the values of a column.
```

6. Data Alignment

- Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
   a 10.0
   b NaN
   c 5.0
   d 7.0
```

6. Data Alignment

- Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
   a 10.0
   b -5.0
   c 5.0
   d 7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

II. Import Dataset

1. Overview

- Data can be in any of the popular formats - CSV, TXT, XLS/XLSX (Excel), sas7bdat (SAS), Stata, Rdata (R) etc.
- Loading data in python environment is the most initial step of analyzing data.

II. Import Dataset

1. Overview

While importing external files, we need to check the following points -

1. Check whether header row exists or not
2. Treatment of special values as missing values
3. Consistent data type in a variable (column)
4. Date Type variable in consistent date format.
5. No truncation of rows while reading external data

II. Import Dataset

2. Import CSV files

- Code:

```
df = pd.read_csv(<path_to_csv_file>)
```

- If no header (title) in raw data file:

```
df = pd.read_csv(<path_to_csv_file>, header = None)
```

- Can include column names by using names= option:

```
df = pd.read_csv(<path_to_csv_file>, header = None, names = ['column 1', 'column 2, 'column 3'])
```

The variable names can also be added separately by using the following command.

```
df .columns = ['column 1', 'column 2, 'column 3']
```

II. Import Dataset

3. Import File from URL

- Don't need to perform additional steps to fetch data from URL.
- Simply put URL in read_csv() function (applicable only for CSV files stored in URL)

```
df = pd.read_csv(<URL_to_csv_file>)
```

For example:

```
df = pd.read_csv("http://winterolympicsmedals.com/medals.csv")
```

II. Import Dataset

4. Read sample of rows and columns

- By specifying *nrows=* and *usecols=*, you can fetch specified number of rows and columns.

```
df = pd.read_csv("http://winterolympicsmedals.com/medals.csv", nrows=5, usecols=(1,5,7))
```

-*nrows = 5* implies you want to import only first 5 rows and usecols= ref columns you want to import.

II. Import Dataset

5. Skip rows while importing

- Suppose you want to skip first 5 rows and wants to read data from 6th row (6th row would be a header row)

```
df = pd.read_csv("http://winterolympicsmedals.com/medals.csv", skiprows=5)
```

II. Import Dataset

6. Specify values as missing values

- By including na_values= option, you can specify values as missing values. In this case, we are telling python to consider dot *(.)* as missing cases.

```
df = pd.read_csv("workingfile.csv", na_values=['.'])
```

II. Import Dataset

7. Read Text File

- We can use `read_table()` function to pull data from text file.

- We can also use `read_csv()` with sep= `"\t"` to read data from tab-separated file.

```
df = pd.read_table(<path_to_txt_file>)
df = pd.read_csv(<path_to_txt_file>, sep ="\t")
```

II. Import Dataset

8. Read Excel File

- The read_excel() function can be used to import excel data into Python.

```
df = pd.read_excel(<path_to_excel_file>, sheetname=<sheet_name>, skiprows=2)
```

- If you do not specify name of sheet in `sheetname` option, it would take by default first sheet.

II. Import Dataset

9. Read delimited file

- Suppose you need to import a file that is separated with white spaces.

```
df = pd.read_table(<path_to_delimited_file>, sep="\s+", header = None)
```

For example:

```
df =
pd.read_table("http://www.ssc.wisc.edu/~bhansen/econometrics/invest.dat",
sep="\s+", header = None)
```

- To include variable names, use the names= option like below:

```
df =
pd.read_table("http://www.ssc.wisc.edu/~bhansen/econometri
sep="\s+", names=['a', 'b', 'c', 'd'])
```

II. Import Dataset

10. Read SAS File

- We can import SAS data file by using *read_sas()* function:

```
df = pd.read_sas(<path_to_sas_file>)
```

For example:

```
df = pd.read_sas('cars.sas7bdat')
```

- If you have a large SAS File, you can try package named pyreadstat which is faster than pandas.

II. Import Dataset

11. Read Stata File

- We can load Stata data file via read_stata() function.

```
df = pd.read_stata(<path_to_stata_file>)
```

For example:

```
df = pd.read_stata('cars.dta')
```

II. Import Dataset

12. Read sample of rows and columns

- We can extract table from SQL database:

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

III. Export Dataset

1. Export dataset to a file

- We can export dataset to csv, txt, excel, etc. file.

```
df.to_csv(<path_to_csv_file>)
df.to_excel(<path_to_excel_file>)
df.to_sql('myDf', engine)
```

IV. Summary Dataset

- Get number of rows and columns

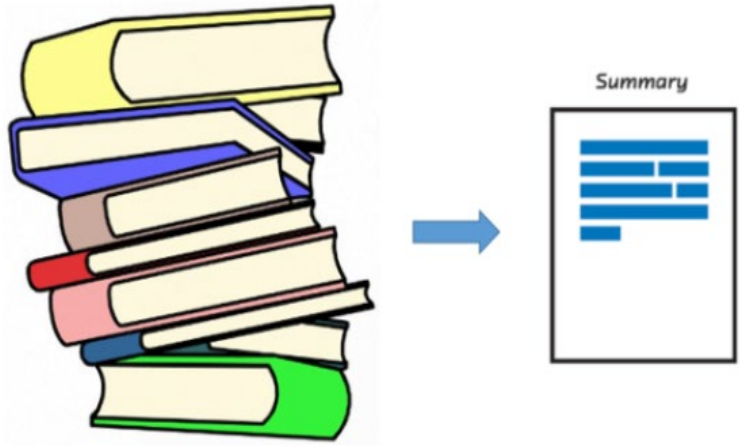
```
>>> df.shape # (rows,columns)
```

- Get index

```
>>> df.index # Describe index
```

- Get list of columns name

```
>>> df.columns # Describe DataFrame columns
```



IV. Summary Dataset

- Get number of row, data type of features, number of missing values, etc.

```
>>> df.info() # Info on DataFrame
```

- Get statistic values of dataset

```
>>> df.describe() # Describe DataFrame
```

- Get number of non-NA values

```
>>> df.count() # Number of non-NA values
```

IV. Summary Dataset

- Get sum of values

```
>>> df.sum() # Sum of values
```

- Get cummulative sum of values

```
>>> df.cumsum() # Cummulative sum of values
```

- Get minimum/maximum values

```
>>> df.min()/df.max() # Minimum/maximum values
```

IV. Summary Dataset

- Get minimum/maximum index value

```
>>> df.idxmin()/df.idxmax() # Minimum/Maximum index value
```

- Get mean of values

```
>>> df.mean() # Mean of values
```

- Get median of values

```
>>> df.median() # Median of values
```

V. Reference

Book:

Learning Pandas, chapter 2, 3, 4, 7

Q & A