Library - Pandas

1. Introduction

What is Pandas?

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.

Why use pandas?

- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- Relevant data is very important in data science.

Installation of Pandas

- Windows

```
pip install pandas
```

- Macbook

```
pip3 install pandas
```

Usage

- Like NumPy, we need to import pandas library

```
import pandas as pd
```

2. Basic data structures - Series

Series

- `Series` is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.).
- The axis labels are collectively referred to as the **index**. The basic method to create a `Series` is to call： `s = pd.Series(data, index=index)`,
  - where `data` can be many different things:
    - a Python dict
    - a NumPy ndarray
    - a scalar value.
  - The passed `index` is a list of axis labels.

From ndarray

- If `data` is an ndarray, **index** must be the same length as **data**. If no index is passed, one will be created having values `[0, ..., len(data) - 1]`.
- For example,

```
>>> import numpy as np
>>> import pandas as pd
>>> s = pd.Series(np.array([10,11,12]))
>>> s
0    10
1    11
2    12
dtype: int64
```

- In the above code, we didn't set index, thus the indices are default. Next, let us set index

```
>>> import numpy as np
>>> import pandas as pd
>>> s = pd.Series(np.array([10,11,12]), index = ['a', 'b', 'c'])
>>> s
a    10
b    11
c    12
dtype: int64
```

From dictionary

- Instantiate Series from a dictionary. The index would be the keys if no index is passed.

```
>>> d = {'Jack': 90, 'Ben': 88, 'Mary': 30 }
>>> pd.Series(d)
Jack    90
Ben     88
Mary    30
dtype: int64
```

- If an index is passed, the values in data corresponding to the labels in the index will be pulled out.

Series is ndarray-like

- `Series` acts very similarly to a `ndarray`. For example, it can support vectorized operations (element-wise operations).

```
>>> s1 = pd.Series(np.array([10,11,12]))
>>> s2 = pd.Series(np.array([2,3,4]))
>>> s1 + s2
0    12
1    14
2    16
dtype: int64
>>> s1 * s2
0    20
1    33
2    48
dtype: int64
```

- `Series` is a valid argument to most NumPy functions.

```
>>> s1 = pd.Series(np.array([10,11,12]))
>>> np.sum(s1)
33
>>> np.log(s1)
0    2.302585
1    2.397895
2    2.484907
dtype: float64
```

- Slicing in Series is also like NumPy

```
>>> s = pd.Series(np.array([9,10,11,12, 13]))
>>> s[:3]
0     9
1    10
```

```
2    11
dtype: int64
>>> s[-2:]
3    12
4    13
dtype: int64
```

- However, `Series` usually uses `.iloc[]` operations to get the value(s).

```
>>> s = pd.Series(np.array([9,10,11,12, 13]))
>>> s.iloc[0]
9
>>> s.iloc[:3]
0     9
1    10
2    11
dtype: int64
>>> s.iloc[-2:]
3    12
4    13
dtype: int64
```

Series is dict-like

- A `Series` is also like a fixed-size dict in that you can get and set values by index label:

```
>>> s = pd.Series(np.array([9,10,11,12]), index = ['a', 'b', 'c', 'd'])
>>> s['a']
9
```

3. Basic data structures - DataFrame

DataFrame

- `DataFrame` is a 2-dimensional labeled data structure with columns of potentially different types.
- You can think of it like a spreadsheet or SQL table, or a dict of Series objects.
- It is generally the most commonly used pandas object. Like Series, DataFrame accepts many different kinds of input:
  - Dict of 1D ndarrays, lists, dicts, or `Series`
  - 2-D numpy.ndarray
  - A `Series`
  - Another `DataFrame`

From series

- The resulting **index** will be the **union** of the indexes of the various Series.

```
import pandas as pd

d = {
    "one": pd.Series([1.0, 2.0, 3.0], index=["a", "b", "c"]),
    "two": pd.Series([21.0, 22.0, 23.0, 4.0], index=["b", "a", "c", "d"]),
}

df = pd.DataFrame(d)

print(df)
```

- The output is

```
   one   two
a  1.0  22.0
b  2.0  21.0
c  3.0  23.0
d  NaN   4.0
```

From dictionary

- We can easily get a pandas DataFrame from a dictionary.

```
import pandas as pd
Dict = {
        'name': ['Jack', 'Mary', 'Joan'],
        'grade': [90, 85, 77],
        'Section': ['Section 7', 'Section 15', 'Section 7']
}
df = pd.DataFrame(Dict)
print(df)
```

- Output

```
   name  grade     Section
0  Jack     90   Section 7
1  Mary     85  Section 15
2  Joan     77   Section 7
```

- Since no index was passes, the default indices are 0, 1, 2.
- We can pass index.

```
import pandas as pd
Dict = {
```

```
    'grade': [90, 85, 77],
    'Section': ['Section 7', 'Section 15', 'Section 7'],
    "HW": [91, 92, 93]
}
df = pd.DataFrame(Dict, index= ['Jack', 'Mary', 'Joan'])
print(df)
```

- The output would be

```
      grade     Section  HW
Jack     90   Section 7  91
Mary     85  Section 15  92
Joan     77   Section 7  93
```

4. Obtain characteristics of dataFrame

Based on the dataFrame `df` defined above, we can obtain the characteristics.

Shape

- `df.shape` return the shape of the dataFrame, i.e., the number of rows and the number of columns.

```
>>> df.shape
(3, 3)
```

Column name

- to obtain column names

```
>>> df.column
Index(['grade', 'Section', 'HW'], dtype='object')
```

index

- obtain index

```
>>> df.index
Index(['Jack', 'Mary', 'Joan'], dtype='object')
```

5. Column selection, addition, deletion

- Based on the dataFrame `df` defined above, let us select the column(s) we are interested.

Choose a single column

- Using *column name* to obtain a column.

```
>>> c = df['grade']
>>> c
Jack    90
Mary    85
Joan    77
Name: grade, dtype: int64
>>> type(c)
<class 'pandas.core.series.Series'>
```

- The type is `Series`, rather than dataFrame.
- We can also use `.loc[,]` method to obtain a single column. In the method, the number before comma indicates row number; the number after comma indicates column number. They all start with 0. `:` means all rows or columns.

```
>>> df.iloc[:,0]
Jack    90
Mary    85
Joan    77
Name: grade, dtype: int64
```

Choose multiple columns

- We can use a list of columns to obtain multiple columns.

```
>>> df1 = df[['grade', 'HW']]
>>> df1
      grade  HW
Jack     90  91
Mary     85  92
Joan     77  93
>>> type(df1)
<class 'pandas.core.frame.DataFrame'>
```

- Take care here. The type of `df1` is DataFrame, rather than `Series`.
- We can also use `.loc[,]` method to obtain the same result.

```
>>> df2 = df.iloc[:,[0,2]]
      grade  HW
Jack     90  91
Mary     85  92
Joan     77  93
<class 'pandas.core.frame.DataFrame'>
```

- In the above program, the value after comma is a list of numbers, which are the column indices.

```
>>> df2 = df.iloc[:,:2]
      grade     Section
Jack     90   Section 7
Mary     85  Section 15
Joan     77   Section 7
```

6. Write and Read a csv file

Write a csv file

- Sometimes, we need to save our dataFrame. It is good to save it as a csv file by `df.to_csv('filename')`. For example,

```
df.to_csv('grade.csv')
```

Read a csv file

- We can use `pd.read_csv('Filename')`.

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

- You may need to adjust your current directory or put your file path in the file name.

7. Some important methods

- `df.info()` - give us the overall information of the dataFrame.
- `df.describe()` - give us the summary for the numerical columns
- Transform column to NumPy array (using `fastfood_NJ.csv`).

```
rating = df['rating'].to_numpy()
```

References:

- https://pandas.pydata.org/docs/user_guide/10min.html
- https://www.w3schools.com/python/pandas/default.asp