Pandas

Pandas

- ✓ Pandas is an open-source, Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- ✓ It is built on top of NumPy, SciPy, to some extent matplotlib.
- ✓ Pandas is well suited for many different kinds of data:
  - ➢ Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
  - ➢ Ordered and unordered time series data.
  - ➢ Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
  - ➢ Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

Primary Data structures of pandas

The two primary data structures of pandas are :

- Series (1-dimensional) and
- DataFrame (2-dimensional)

✓ They are used to handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering.

pandas - Data Structure Types

1. 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 as index.

- The basic method to create a Series is to call:

```
>>> import pandas as pd
>>> s = pd.Series(data = d, index=index)
```

- Here, data can be many different things:
  - a Python dictinary
  - an ndarray
  - a scalar value (like 5)

pandas - Data Structure Types

2. DataFrame :

It 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.

```
>>> s = pd.DataFrame(data = d , index=index)
```

- DataFrame accepts many different kinds of input:

  - Dictionary of 1D ndarrays, lists, dicts, or Series
  - 2-D numpy.ndarray
  - Structured or record ndarray
  - A Series
  - Another DataFrame

pandas - Series

✓ Series

- One-dimensional array like object containing data and labels (or index)

```
import pandas as pd
series = pd.Series(list('98765'))
print( series)
#output
0   9
1   8
2   7
3   6
4   5
dtype: object
```

```
series = pd.Series(tuple('abcdef'))
print(series)
#output
0   a
1   b
2   c
3   d
4   e
5   f
dtype: object
```

pandas - Traversing Series

Working with Index :

- Index in a series can be specified and using the index we can fetch its corresponding value.

```
series = pd.Series([9,8,7,6],index = ['*','**','***','****'])
print(series)
#output
*      9
**     8
***    7
****   6
dtype: int64
```

- Multiple values can be fetched with multiple indexes

```
print (series[ ['*','***'] ])
#output
*     9
***   7
dtype: int64
```

pandas-Traversing Series (Contd…)

- Series is like a fixed-length, order dictionary.

```
series = pd.Series(range(5),index =list('xyzxy'))
print( series)
#output
x   0
y   1
z   2
x   3
y   4
dtype: int64
```

- Unlike dictionary, index items don't have to be unique

```
# Fetching the values with index x
print( series['x'])
#output
x   0
x   3
dtype: int64
```

pandas - Incomplete Data in Series

- ✓ pandas can accommodate incomplete data. Incomplete data is replaced with "NaN" and "NaN" value is not an issue in arithmetic operations. Unlike in Numpy ndarray, data is automatically aligned

```
series = pd.Series({1:10,2:20,3:30},index = [1,2,3,4])
print( series)
#output
1    10.0
2    20.0
3    30.0
4     NaN
dtype: float64
```

- ✓ We can use numpy-operations on data for filtering

```
print( series * 4)
#output
1     40.0
2     80.0
3    120.0
4      NaN
dtype: float64
```

pandas - DataFrame

✓ DataFrame Creation

➢ Creation with dictionary of equal-length list

```
data={'country':['INDIA','USA','INDIA','USA'],
            'Year':[2010,2011,2012,2013],
            'Population':[20,28,32,38]}
dataFrame=pd.DataFrame(data)
print(dataFrame)
#output
  Population        Year       country
0          20        2010       INDIA
1          28        2011       USA
2          32        2012       INDIA
3          38        2013       USA
```

pandas - DataFrame

- ✓ New Columns can be added by computation or direction assignment

```
# Adding new column
dataFrame[ 'GDP' ]  =  { 6.3, 5.7, 7.8, 5.5 }
print(dataFrame)
#output
   country  Year  Population   GDP
0   INDIA   2010        20      6.3
1     USA   2011        28      5.5
2   INDIA   2012        32      7.8
3     USA   2013        38      5.7
```

pandas - DataFrame

- ✓ Function on DataFrame
  - ➢ We can apply functions like sum(), mean(), max(), head(), count(), tail() etc on dataFrame

```
print("\nMin GDP:",dataFrame['GDP'].min())
print("\nMax GDP:",dataFrame['GDP'].max())
print("\nSum of GDP:",dataFrame['GDP'].sum())
print("\nMean of GDP:",dataFrame['GDP'].mean())
print("\nMedian of GDP:",dataFrame['GDP'].median())
print("\nCount of GDP:",dataFrame['GDP'].count())
print("\nHead 2 of GDP:\n",dataFrame['GDP'].head(2))
print("\nTail 2 of GDP:\n",dataFrame['GDP'].tail(2))
```

pandas - DataFrame

- describe() functions will give the summary of DataFrame

```
>>> dataFrame.describe()
```

	Population	Year	GDP
count	4.000000	4.000000	4.000000
mean	29.500000	2011.500000	6.325000
std	7.549834	1.290994	1.040433
min	20.000000	2010.000000	5.500000
25%	26.000000	2010.750000	5.650000
50%	30.000000	2011.500000	6.000000
75%	33.500000	2012.250000	6.675000
max	38.000000	2013.000000	7.800000

pandas - Data Loading

- ✓ pandas support several ways to handle data loading
- ✓ Text file data
  - ➢ read_csv
  - ➢ read_table
- ✓ Structured data (JSON, XML & HTML)
  - ➢ It works fine with existing libraries
- ✓ Excel (depends upon xlrd and openpyxl packages)
- ✓ Database
  - ➢ pandas.io.sql module (read_frame)

pandas – Loading CSV data

- ✓ pandas.read_csv will load the csv data

```
>>>data = pd.read_csv('employee.csv')
>>> data
```

	Emp_ID	EMP_Name	Gender	Age	Join_year
0	5001	Purnima Pandey	Female	56	1971
1	5002	Chandramohan Doss S	Male	45	1972
2	5003	Zaheer Begum M	Male	36	1979
3	5004	Sreenath A N	Male	38	1987
4	5005	Sivaramakrishnan R	Male	57	1994
5	5006	Gita N Murthy	Female	48	1988

pandas – Loading CSV data

- ✓ Using len on DataFrame will give you the number of rows

```
>>> len(data)
19
```

- ✓ We can get the column name using columns property

```
>>> data.columns
Index(['Emp_ID', 'EMP_Name', 'Gender', 'Age', 'Join_year'], dtype='object')
```

pandas – Loading CSV data

- ✓ Columns can be accessed in two ways
  - ➢ The first is using the DataFrame like a dictionary with string keys. Multiple columns can be accessed by passing multiple column names

    >>> data[‘Open’]
  - ➢ The second way to access columns is using the dot syntax. This only works if your column name could also be a Python variable name (i.e., no spaces), and if it doesn't collide with another DataFrame property or function name (e.g., count, sum)

    >>> data.Open

```
>>> data['Emp_ID']
0    5001
1    5002
2    5003
3    5004
4    5005
5    5006
Name: Emp_ID, dtype: int64
```

pandas – Loading CSV data

- ✓ head() function lists the first 5 rows as default. If we want to display first n rows. Use head(n)

```
>>> data.head(2)
   Emp_ID   EMP_Name            Gender  Age  Join_year
0  5001     Purnima Pandey      Female  56   1971
1  5002     Chandramohan Doss S Male    45   1972
```

- ✓ head() function can be applied on columns also

```
>>> data.Emp_ID.head(2)
0  5001
1  5002
```

- ✓ We can use other functions like tail(), max(), min(), std(), mean() etc

```
>>> data.Emp_ID.max()
5006
```

THANK YOU!!