Pandas

Data Manipulation in Python

Pandas

- Built on NumPy
- Adds

```
import pandas as pd
```

Pandas Fundamentals

Three fundamental Pandas data structures:

- `Series` - a one-dimensional array of values indexed by a `pd.Index`
- `Index` - an array-like object used to access elements of a `Series` or `DataFrame`
- `DataFrame` - a two-dimensional array with flexible row indices and column names

Series from List

```
In [4]: data = pd.Series([’a’,’b’,’c’,’d’])

In [5]: data
Out[5]:
0    a
1    b
2    c
3    d
dtype: object
```

The 0..3 in the left column are the `pd.Index` for `data`:

```
In [7]: data.index
Out[7]: RangeIndex(start=0, stop=4, step=1)
```

The elements from the Python list we passed to the `pd.Series` constructor make up the values:

```
In [8]: data.values
Out[8]: array([’a’, ’b’, ’c’, ’d’], dtype=object)
```

Notice that the values are stored in a Numpy array.

Series from Sequence

You can construct a list from any definite sequence:

```
In [24]: pd.Series(np.loadtxt(’exam1grades.txt’))
Out[24]:
0       72.0
1       72.0
2       50.0
...
134     87.0
dtype: float64
```

or

```
In [25]: pd.Series(open(’exam1grades.txt’).readlines())
Out[25]:
0       72\n
1       72\n
2       50\n
...
134     87\n
dtype: object
```

. . . but not an indefinite sequence:

```
In [26]: pd.Series(open(’exam1grades.txt’))
...
TypeError: object of type ’_io.TextIOWrapper’ has no len()
```

Series from Dictionary

```
salary = {"Data Scientist": 110000,
          "DevOps Engineer": 110000,
          "Data Engineer": 106000,
          "Analytics Manager": 112000,
          "Database Administrator": 93000,
          "Software Architect": 125000,
          "Software Engineer": 101000,
          "Supply Chain Manager": 100000}
```

Create a `pd.Series` from a `dict`: [1]

```
In [14]: salary_data = pd.Series(salary)

In [15]: salary_data
Out[15]:
Analytics Manager         112000
Data Engineer             106000
Data Scientist            110000
Database Administrator     93000
DevOps Engineer           110000
Software Architect        125000
Software Engineer         101000
Supply Chain Manager      100000
dtype: int64
```

The index is a sorted sequence of the keys of the dictionary passed to `pd.Series`

[1] https://www.glassdoor.com/List/Best-Jobs-in-America-LST_KQ0,20.htm

Series with Custom Index

General form of Series constructor is `pd.Series(data, index=index)`

- Default is integer sequence for sequence data and sorted keys of dictionaries
- Can provide a custom index:

```
In [29]: pd.Series([1,2,3], index=[’a’, ’b’, ’c’])
Out[29]:
a    1
b    2
c    3
dtype: int64
```

The index object itself is an immutable array with set operations.

```
In [30]: i1 = pd.Index([1,2,3,4])

In [31]: i2 = pd.Index([3,4,5,6])

In [32]: i1[1:3]
Out[32]: Int64Index([2, 3], dtype=’int64’)

In [33]: i1 & i2 # intersection
Out[33]: Int64Index([3, 4], dtype=’int64’)

In [34]: i1 | i2 # union
Out[34]: Int64Index([1, 2, 3, 4, 5, 6], dtype=’int64’)

In [35]: i1 ^ i2 # symmetric difference
Out[35]: Int64Index([1, 2, 5, 6], dtype=’int64’)
```

Series Indexing and Slicing

Indexing feels like dictionary access due to flexible index objects:

```
In [37]: data = pd.Series([’a’, ’b’, ’c’, ’d’])

In [38]: data[0]
Out[38]: ’a’

In [39]: salary_data[’Software Engineer’]
Out[39]: 101000
```

But you can also slice using these flexible indices:

```
In [40]: salary_data[’Data Scientist’:’Software Engineer’]
Out[40]:
Data Scientist          110000
Database Administrator   93000
DevOps Engineer         110000
Software Architect      125000
Software Engineer       101000
dtype: int64
```

DataFrame

```
In [42]: jobs = pd.DataFrame({’salary’: salary_data, ’openings’: openings})

In [43]: jobs
Out[43]:
                      openings  salary
Analytics Manager         1958  112000
Data Engineer             2599  106000
Data Scientist            4184  110000
Database Administrator    2877   93000
DevOps Engineer           2725  110000
Software Architect        2232  125000
Software Engineer        17085  101000
Supply Chain Manager      1270  100000
UX Designer               1691   92500
```

```
In [46]: jobs.index
Out[46]:
Index([’Analytics Manager’, ’Data Engineer’, ’Data Scientist’,
       ’Database Administrator’, ’DevOps Engineer’, ’Software Architect’,
       ’Software Engineer’, ’Supply Chain Manager’, ’UX Designer’],
      dtype=’object’)

In [47]: jobs.columns
Out[47]: Index([’openings’, ’salary’], dtype=’object’)
```

Simple DataFrame Indexing

In a NumPy array you index a 2-d array by row. In a Panda DataFrame you index by column.

```
In [48]: jobs['salary']
Out[48]:
Analytics Manager         112000
Data Engineer             106000
Data Scientist            110000
Database Administrator     93000
DevOps Engineer           110000
Software Architect        125000
Software Engineer         101000
Supply Chain Manager      100000
UX Designer                92500
Name: salary, dtype: int64
```

Each colum is a Series:

```
In [49]: type(jobs['salary'])
Out[49]: pandas.core.series.Series
```

Messy CSV Files

Remember the Tides Exercise? Pandas's `read_csv` can handle most of the data pre-processing:

```
pd.read_csv(’wpb-tides-2017.txt’, sep=’\t’, skiprows=14, header=None,
           usecols=[0,1,2,3,5,7],
           names=[’Date’, ’Day’, ’Time’, ’Pred(ft)’, ’Pred(cm)’, ’High/Low’],
           parse_dates=[’Date’,’Time’])
```