CSC 223 - Advanced Scientific Programming

Basic Pandas Types

Pandas

- Pandas is a library built on Numpy that provides an implementation of a `DataFrame`
- A `DataFrame` is a multidimensional array with row and column labels and can contain heterogeneous types
- Pandas provides three main data types: `Series`, `DataFrame`, and `Index`
- Conventional way to import Pandas:

```
import pandas as pd
```

Pandas `Series`

- The `Series` type represents a one-dimensional array of indexed data
- Constructing `Series` objects
  - `pd.Series(data, index=index)`
  - `data` can be a list, numpy array, or dict
  - `index` is an array of index values
- Indexing `Series` Object
  - A `Series` is indexed by its index values
  - A `Series` can also be sliced like a Python list

Pandas Series Example

```
>>> data = pd.Series([0.25, 0.5, 0.75, 1.0])
>>> data
0   0.25
1   0.50
2   0.75
3   1.00
dtype: float64
>>> data[1]
0.5
>>> data[1:3]
1    0.50
2    0.75
dtype: float64
```

Pandas Series with Index Example

```
>>> data = pd.Series([0.25, 0.5, 0.75, 1.0],
              index = [’a’, ’b’, ’c’, ’d’])
>>> data
a   0.25
b   0.50
c   0.75
d   1.00
dtype: float64
>>> data[’b’]
0.5
>>> data[’b’:’d’]
b    0.50
c    0.75
d    1.00
dtype: float64
```

Pandas Series Attributes

- index: the index object

```
>>> s1 = pd.Series([2,4,6])
>>> s1.index
RangeIndex(start=0, stop=3, step=1)
>>> s2 = pd.Series({100: 1, 200: 3, 300: 5})
>>> s2.index
Int64Index([100, 200, 300], dtype='int64')
```

- values: the underlying NumPy array

```
>>> s1.values
array([2, 4, 6])
>>> s2.values
array([1, 3, 5])
```

Pandas `DataFrame` Object

- A `DataFrame` is two-dimensional array with flexible row and column names
- Each column in a `DataFrame` is a `Series`
- `DataFrame` objects can be constructed from:
  - a single `Series`
  - a list of dicts
  - a dict of `Series` objects
  - a two-dimensional Numpy array

Pandas DataFrame Construction Example

```
>>> df = pd.DataFrame([[2,4,6], [1,3,5]])
>>> df
   0  1  2
0  2  4  6
1  1  3  5
>>> df.index
RangeIndex(start=0, stop=2, step=1)
>>> df.columns
RangeIndex(start=0, stop=3, step=1)
```

Pandas DataFrame Construction Examples

```
>>> pd.DataFrame(np.ones((3,2)),
                 columns=[’one’, ’two’],
                 index=[’a’, ’b’, ’c’])
   one  two
a  1.0  1.0
b  1.0  1.0
c  1.0  1.0
>>> pd.DataFrame([{’a’: i, ’b’: 2 * i}
                  for i in range(3)])
   a  b
0  0  0
1  1  2
2  2  4
```

Adding/Removing Columns from a DataFrame

- Add a column (similar to adding an element to a `dict`):

```
>>> df
   one  two
0    1    3
1    2    4
>>> df['three'] = [5, 6]
>>> df
   one  two  three
0    1    3      5
1    2    4      6
```

- Remove a column:

```
>>> df.pop('three') # or del df['three']
>>> df
   one  two
0    1    3
1    2    4
```

Pandas Index Object

- An Index enables the reference and modification of elements in Series and Index objects
- An Index can be thought of as an immutable array or as an ordered set
- Example

```
>>> indA = pd.Index([1, 3, 5, 7, 9])
>>> indB = pd.Index([2, 3, 5, 7, 11])
>>> indA[::2]
Int64Index([1, 5, 9], dtype='int64')
>>> indA & indB # set intersection
Int64Index([3, 5, 7], dtype='int64')
>>> indA | indB # set union
Int64Index([1, 2, 3, 5, 7, 9, 11], dtype='int6
```

Pandas Indexers

- Indexer attributes expose slicing interfaces to the data in a `Series` object
  - `loc` allows indexing and slicing based on the explicit index
  - `iloc` allows indexing and slicing based on the implicit Python-style index
  - `ix` is a hybrid of the previous approaches
- Indexers can provide access to Numpy-style indexing such as masking and fancy indexing
- In Pandas, *indexing* refers to columns, *slicing* refers to rows

Pandas Indexer Examples

```
>>> data
        one       two
a  0.495141  0.965454
b  0.673145  0.246473
c  0.716398  0.730835

>>> data.loc[:’b’, :’one’]
        one
a  0.495141
b  0.673145

# equivalent to the above
>>> data.iloc[:2, :1]
>>> data.ix[:2, :’one’]
```

Summary of Selection on DataFrames

Operation	Syntax	Result Type
select a column	`df[col]`	`Series`
select row by label	`df.loc[label]`	`Series`
select row by integer location	`df.iloc[loc]`	`Series`
slice rows	`df[5:10]`	`DataFrame`
select rows by boolean vector	`df[bool_vec]`	`DataFrame`

Pandas and UFuncs

- Indices are preserved when using ufuncs
- Indices are aligned when performing binary ufuncs
- Index and column alignment is preserved when performing operations between `DataFrame` and `Series` objects

Pandas and UFuncs Examples

```
>>> A = pd.DataFrame(
          np.arange(4).reshape((2,2)),
          columns=[’one’, ’two’])
>>> B = pd.DataFrame(
          np.arange(3).reshape((3,3)),
          columns=[’three’, ’two’, ’one’])
>>> A + B
   one  three  two
0  2.0    NaN  2.0
1  7.0    NaN  7.0
2  NaN    NaN  NaN
```

Missing Data

- Pandas treats `None` and `NaN` as null (missing) values
- Functions related to missing values:
  - `isnull`: return boolean mask of null values
  - `notnull`: opposite of `isnull`
  - `dropna`: filter out missing values
  - `fillna`: replace missing values with a specified value