CSC 223 - Advanced Scientific Programming

Pandas Aggregation and Grouping

Aggregation

- Data analysis of large data typically requires some form of summarization.
- Aggregations (sum, mean, min, etc.) reduce the data to a single number and may provide insight into the nature of large datasets.
- Pandas includes several methods for computing aggregations

Simple Aggregations

- A Pandas `Series` supports aggregations similar to Numpy arrays:

```
>>> s = pd.Series(range(10))
>>> s.sum()
45
>>> s.mean()
4.5
```

- A `DataFrame` aggregates column-wise by default, but can also take an axis argument:

```
>>> df = pd.DataFrame({’A’: [1,2], ’B’: [3,4]})
>>> df.mean()
A    1.5
B    3.5
dtype: float64
>>> df.mean(axis=1)
0    2.0
1    3.0
dtype: float64
```

Pandas describe

- The describe method computes all the common aggregations of a Series or DataFrame:

```
>>> df = pd.DataFrame({’A’: [1,2], ’B’: [3,4]})
>>> df.describe()
              A         B
count  2.000000  2.000000
mean   1.500000  3.500000
std    0.707107  0.707107
min    1.000000  3.000000
25%    1.250000  3.250000
50%    1.500000  3.500000
75%    1.750000  3.750000
max    2.000000  4.000000
```

Pandas Built-in Aggregations

Aggregation	**Description**
`count`	total number of items
`first, last`	first and last item
`mean, median`	mean and median
`min, max`	minimum and maximum
`std, var`	standard deviation and variance
`mad`	mean absolute deviation
`prod`	product of all items
`sum`	sum of all items

Groupby

- The groupby operation enables conditional aggregations based on some label of index
- The name “group by” comes from SQL database language
- The groupby operation essentially does the following:
  1. split: break up and group the data based on the specified key
  2. apply: compute some function (aggregation, transformation, or filtering) within the individual groups
  3. combine: merge the results into an output array

Basic groupby Example

- Create a DataFrame:

```
>>> df = pd.DataFrame({’key’: list(’ABCABC’),
...    ’data1’: range(6), ’data2’: range(7,13)})
>>> df
  key  data1  data2
0   A      0      7
1   B      1      8
2   C      2      9
3   A      3     10
4   B      4     11
5   C      5     12
```

- Group by key (becomes the new index) and apply sum:

```
>>> df.groupby(’key’).sum()
     data1  data2
key
A        3     17
B        5     19
C        7     21
```

Performing Multiple Aggregations

- The aggregate method can apply multiple aggregations
- Example: a list of functions

```
>>> df.groupby(’key’).aggregate(
...        [’min’, np.median, max])
    data1            data2
      min median max   min median max
key
A        0    1.5   3     7    8.5  10
B        1    2.5   4     8    9.5  11
C        2    3.5   5     9   10.5  12
```

- Example: a dictionary mapping column names to functions

```
>>> df.groupby(’key’).aggregate({’data1’: min,
                                 ’data2’: max})
     data1  data2
key
A        0     10
B        1     11
C        2     12
```

Filtering

- A filtering operation drops data based on group properties
- The `filter` method takes a function that takes a `DataFrame` as a parameter and returns a Boolean

```
>>> df.groupby(’key’).filter(
...          lambda x: x[’data1’].sum() > 3)
  key  data1  data2
1   B      1      8
2   C      2      9
4   B      4     11
5   C      5     12
```

Transformation

- The `transform` method returns a transformed version of the data; the output is the same shape as the input.
- Common example: center data by subtracting the group-wise mean:

```
>>> df.groupby(’key’).transform(
...        lambda x: x - x.mean())
   data1  data2
0   -1.5   -1.5
1   -1.5   -1.5
2   -1.5   -1.5
3    1.5    1.5
4    1.5    1.5
5    1.5    1.5
```

Applying Arbitrary Functions

- The apply method can apply arbitrary functions that take a DataFrame as an argument and returns a DataFrame, Series, or scalar value; the combine operation will be tailored to the type of output returned.
- Example: normalize the first column by the sum of the second

```
>>> def f(x):
...     x[’data1’] /= x[’data2’].sum()
...     return x
...
>>> df.groupby(’key’).apply(f)
  key     data1  data2
0   A  0.000000      7
1   B  0.052632      8
2   C  0.095238      9
3   A  0.176471     10
4   B  0.210526     11
5   C  0.238095     12
```

Specifying the Split Key

- The key can be a sequence with a length matching that of the DataFrame

```
>>> L = [0, 1, 0, 1, 2, 0]
>>> df.groupby(L).sum()
   data1  data2
0      7     28
1      4     18
2      4     11
```

- The key can be a dictionary that maps index values to group keys

```
>>> df2 = df.set_index(’key’)
>>> mapping = {’A’: ’vowel’, ’B’: ’consonant’,
...            ’C’: ’consonant’}
>>> df2.groupby(mapping).sum()
           data1  data2
consonant     12     40
vowel          3     17
```

Specifying the Split Key (Continued)

- The key can be a Python function that takes an index value and returns a group

```
>>> df2.groupby(str.lower).mean()
   data1  data2
a    1.5    8.5
b    2.5    9.5
c    3.5   10.5
```

- The key can be a list of key choices combined to group on a multi-index

```
>>> df2.groupby([str.lower, mapping]).mean()
             data1  data2
a vowel        1.5    8.5
b consonant    2.5    9.5
c consonant    3.5   10.5
```