Map, Filter and Reduce using Pandas, Numpy, and standard library

- Map, Filter and Reduce are design patterns for stream processing of data.
- Pandas has powerful constructs. Numpy and standard library also can be used.
- Lambda functions further simplify and clarify the code.

This notebook gives examples of Map, Filter and Reduce.

This notebook is available at https://github.com/cwinsor/pandas_gold.git

In [38]:

```
import pandas as pd
import numpy as np
```

In [39]:

```
#  Here we give examples of Map, Filter, Reduce.
```

---------- Map -------------

Map is an "elementwise" operation. Each element is mapped to a new value. This can be seen as creating a new column.

- https://www.geeksforgeeks.org/python-map-function/

Procedure:

1. create a function that will map a single item
2. create a list, tuple, np.array
3. map(function, iterable)

This returns an iterable, so you may want to list(map())

In [9]:

```
def addOne(n):
    return n + 1
```

In [10]:

```
# the first argument to "map" is a function
# the second argument is an iterable

# for List
numbers = [1, 2, 3, 4]
y = map(addOne,numbers)
print(y)
print(list(y))
# note it returns a map object so list() is used to view it
```

```
<map object at 0x000002413B8BA5F8>
[2, 3, 4, 5]
```

In [11]:

```
# for np.array
numbers = np.array([1,2,3,4])
list(map(addOne,numbers))
```

Out[11]:

```
[2, 3, 4, 5]
```

In [12]:

```
numbers = np.array([[1,2,3,4],[5,6,7,8]])
print(numbers)
print()
list(map(addOne,numbers))
```

```
[[1 2 3 4]
 [5 6 7 8]]
```

Out[12]:

```
[array([2, 3, 4, 5]), array([6, 7, 8, 9])]
```

in Pandas it is "apply" for DataFrame, "map" for Series

- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.apply.html
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.map.html

In [13]:

```
import pandas as pd
df = pd.DataFrame({'A': [1,2,3,4],
                   'B': [10,20,30,40],
                   'C': [20,40,60,80]
                  },
                  index=['Row 1', 'Row 2', 'Row 3', 'Row 4'])
df
```

Out[13]:

	A	B	C
Row 1	1	10	20
Row 2	2	20	40
Row 3	3	30	60
Row 4	4	40	80

In [14]:

```
# for DataFrame use "apply"
df[['E', 'F']] = df[['B','C']].apply(addOne)
df
```

Out[14]:

	A	B	C	E	F
Row 1	1	10	20	11	21
Row 2	2	20	40	21	41
Row 3	3	30	60	31	61
Row 4	4	40	80	41	81

In [15]:

```
# for Series use "map"
```

```
series = pd.Series([10,20,30,40],
                      index=['Item 1', 'Item 2', 'Item 3', 'Item 4'])
series
```

Out[15]:

```
Item 1    10
Item 2    20
Item 3    30
Item 4    40
dtype: int64
```

In [16]:

```
series.map(addOne)
```

Out[16]:

```
Item 1    11
Item 2    21
Item 3    31
Item 4    41
dtype: int64
```

------------------ Filter --------------------

Filter returns a subset of your data based on some criterion.

- With python standard library and numpy:
  - create a filtering function
  - use filter(function, sequence)
  - https://www.geeksforgeeks.org/filter-in-python/
- With Pandas
  - create a criterion and apply that to the DataFrame
  - https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/03_subset_data.html

In [17]:

```
# the filtering function takes in a single item
def isVowel(arg):
    vowels = ['a', 'e', 'i', 'o', 'u']
    return arg in vowels
```

In [18]:

```
# the first argument to "filter" is a function
# the second argument is an iterable

sequence = ['g', 'e', 'e', 'j', 'a', 's', 'p', 'r']
filtered = filter(isVowel, sequence)

# it returns an iterator...
print('filtered is: {}'.format(filtered))

# here we use the iterator...
print('The filtered letters are:')
for s in filtered:
    print(s)

# or transform the iterator to a list...
```

```
filtered = list(filter(isVowel, sequence))
print('filtered is: {}'.format(filtered))
```

```
filtered is: <filter object at 0x000002413B936208>
The filtered letters are:
e
e
a
filtered is: ['e', 'e', 'a']
```

Pandas - filter is done using a binary selector...

This does not create a new table, rather is a 'view' of the original table

In [19]:

```
import pandas as pd
df = pd.DataFrame({'A': [1,2,3,4],
                   'B': [10,20,30,40],
                   'C': [20,40,60,80]
                  },
                  index=['Row 1', 'Row 2', 'Row 3', 'Row 4'])
df
```

Out[19]:

	A	B	C
Row 1	1	10	20
Row 2	2	20	40
Row 3	3	30	60
Row 4	4	40	80

In [20]:

```
criterion1 = df['B'] > 10
criterion2 = df['C'] < 80
df[criterion1 & criterion2]
```

Out[20]:

	A	B	C
Row 2	2	20	40
Row 3	3	30	60

In [21]:

```
print(type(criterion1))
```

```
<class 'pandas.core.series.Series'>
```

In [22]:

```
criterion1
```

Out[22]:

```
Row 1    False
Row 2     True
Row 3     True
Row 4     True
Name: B, dtype: bool
```

In [23]:

```
# note only one copy of data in Pandas
```

-------- Reduce ------------------

Reduce applies a function along an axis returning a single value for the column. This is different from "map" which is an element-wise operation.

- Reduce can also be applied to rows
- Reduce can be used with Pandas groupBy to summarize subsets of data

Python Standard Library..

https://www.geeksforgeeks.org/reduce-in-python/

- Many standard reduce functions are available in std lib, numpy, scipy. Try to use those.
- If no standard reduction is available, try to break problem be broken into (map + reduce)
- Be careful about functools.reduce - see example!

functools.reduce

https://www.geeksforgeeks.org/reduce-in-python/ functools.reduce works as follows:

- At first step, first two elements of sequence are picked and the result is obtained.
- Next step is to apply the same function to the previously attained result and the number just succeeding the second element and the result is again stored.
- This process continues till no more elements are left in the container.
- The final returned result is returned and printed on console.

Be careful - it can be tricky as the example shows.

In [24]:

```
# an implementation of a function to  sum( square(a) + square(b) )
import functools
def sumOfSquares(arg1, arg2):
    return arg1 + (arg2*arg2)
```

In [25]:

```
arr = np.array([[1,5],[2,5],[3,5]])
print(arr)
print()
print(functools.reduce(sumOfSquares, arr))
```

```
[[1 5]
 [2 5]
 [3 5]]

[14 55]
```

try to debug *THAT*

Pandas

first map (square), then reduce (sum)

In [26]:

```
# using function
arr = np.array([[1,5],[2,5],[3,5]])
df = pd.DataFrame(arr, columns=["col1", "col2"])

def square(arg):
    return arg[0]**2

df['c1_sq'] = df[['col1']].apply(square, axis=1)
df['c2_sq'] = df[['col2']].apply(square, axis=1)

print(df)

# reduce using sum()
df.sum()
```

```
   col1  col2  c1_sq  c2_sq
0     1     5      1     25
1     2     5      4     25
2     3     5      9     25
```

Out[26]:

```
col1      6
col2     15
c1_sq    14
c2_sq    75
dtype: int64
```

lambda

Lambda is an in-line function. It replaces the "def" above.

https://towardsdatascience.com/lambda-functions-with-practical-examples-in-python-45934f3653a8

lambda with scalar

In [27]:

```
(lambda x: x*2)(12)
```

Out[27]:

```
24
```

lambda as the function for "map"

remember... map(function, iterable)

In [28]:

```
# map(function, iterable, ...)

# lambda plays the role of function
# list_1 is the iterable

# "add 100 to each item in the list"
```

```
list_1 = [1,2,3,4,5,6,7,8,9]
list(map(lambda x: x+100, list_1))
```

Out[28]:

```
[101, 102, 103, 104, 105, 106, 107, 108, 109]
```

lambda as the function for "filter"

remember... filter(function, iterable)

In [29]:

```
# filter(function, iterable)

# lambda plays the role of function
# list_1 is the iterable

# "find the even numbers in the list"
list_1 = [1,2,3,4,5,6,7,8,9]
list(filter(lambda x: x%2==0, list_1))
```

Out[29]:

```
[2, 4, 6, 8]
```

using lambda with DataFrame

- https://stackoverflow.com/questions/13331698/how-to-apply-a-function-to-two-columns-of-pandas-dataframe

See "so wonderfully readable":

1. pass a list to the lambda call
2. Selecting and ordering the list occurs outside the lambda
3. Within the lambda use [0],[1] to access as args

- This is "safe" (args are specified/chosen by caller, not inside lambda)
- This is "clear" ('col1', 'col2' are clearly called out as first and second args)

In [30]:

```
arr = np.array([[1,5],[2,5],[3,5]])
df = pd.DataFrame(arr, columns=["col1", "col2"])

df['squared'] = df[['col1','col2']].apply(lambda x: x[0]**2 + x[1]**2, axis=1)
print(df)

# reduce using sum()
df.sum()
```

```
   col1  col2  squared
0     1     5       26
1     2     5       29
2     3     5       34
```

Out[30]:

```
col1        6
col2       15
squared    89
dtype: int64
```

Example using above

In [37]:

```python
# We have farm animals that periodically get weighed
# find the animals that have weighed less than 50 -OR- have weighed more than 600

df = pd.DataFrame({'animal_id': [123, 123, 123, 123, 123, 456, 456, 255, 256, 257],
                   'weight':    [550, 575, 615, 620, 625, 460, 470,   5,   8,   4],
                  })
print('Original')
print(df)

# to do this...
# first make a categorical attribute indicating weight matches our requirements [true,
# can use 'apply' with lambda
#   df['weight_match'] = df[['weight']].apply(lambda args: (args[0]>600)|(args[0]<50),
# or
df['weight_match'] =  (df['weight']<50) | (df['weight']>600)
print('\ncategorical match')
print(df)

# to make a succinct list
# make a criterion that satisfies our requirement
criterion_1 = df['weight_match'] == True
print('\nsuccinct list')
print(df[criterion_1])

# we then return a series of just the animal_ids
series = df[criterion_1]['animal_id']
print('\njust the IDs')
print(series)

# we then find the unique animal_ids in the series
print('\nunique IDs')
print(series.unique())
```

```
Original
   animal_id  weight
0        123     550
1        123     575
2        123     615
3        123     620
4        123     625
5        456     460
6        456     470
7        255       5
8        256       8
9        257       4

categorical match
   animal_id  weight  weight_match
0        123     550         False
1        123     575         False
2        123     615          True
3        123     620          True
4        123     625          True
5        456     460         False
6        456     470         False
7        255       5          True
8        256       8          True
9        257       4          True
```

```
succinct list
   animal_id  weight  weight_match
2        123     615          True
3        123     620          True
4        123     625          True
7        255       5          True
8        256       8          True
9        257       4          True

just the IDs
2    123
3    123
4    123
7    255
8    256
9    257
Name: animal_id, dtype: int64

unique IDs
[123 255 256 257]
```

That is all for today.

Thank you !