01z_AppendixPythonFuncDataProc

May 17, 2022

1 Python Functional Data Processing

In base python we often want to repeat a process for each entry in a collection…

[12]:
```
prices = [1, 4, 65, 13]
```

[13]:
```
for p in prices:
    print(p)
```

```
1
4
65
13
```

When processing data, more specifically, we want to create a new collection *derived* from this existing one…

[14]:
```
profits = []
profit_percent = 0.05

for p in prices:
    profits.append( p * profit_percent)
```

[15]:
```
profits
```

[15]:
```
[0.05, 0.2, 3.25, 0.65]
```

Above we understanding obtaining `profits` from `prices`, *algorithmically*: as a sequence of steps.

```
1. obtain first price
2. multiply it by a ratio
3. append it to a list
4. next price
5. ....
```

Thinking this way is useful for software engineering but not data analysis, which is more mathematical.

We would really like "multiply all prices by 0.05"… who cares how it's done.

[16]:
```python
[ p * 0.05   for p in prices ]
```

[16]: [0.05, 0.2, 3.25, 0.65]

This syntax is called a "comprehension" and is essentially a python syntax for the SELECT sql command.

```
NEW_LIST = [  CHANGE(ELEMENT)  for ELEMENT in OLD_LIST ]
```

1.1 How do I write a comprehension?

Comprehensions are often best written (and read) right-to-left,

- write in the original collection

```
[  ...  prices ]
```

- name each element

```
[  ... for p in  prices ]
```

- write the "transformation" (ie., the operation which computes the *new* element)

```
[  p * 0.05 for p in  prices ]
```

- NB. this does not change prices, just like SELECT, we get a new collection returned

Good idea to start with a comprehension which doesn't change anything (ie., SELECT *)

1.2 Why do I need to know about comprehensions?

If you are using a library, *you may not need to know*!

Eg., pandas will *automatically* do this for you…

[17]:
```python
import pandas as pd

df = pd.DataFrame({
    'Prices': prices
})

df
```

[17]:

	Prices
0	1
1	4
2	65
3	13

Pandas *automatically* "vectorizes" operations on columns,

[18]:
```python
df['Prices'] * 0.05
```

[18]:
```
0    0.05
1    0.20
2    3.25
3    0.65
Name: Prices, dtype: float64
```

NB. vectorizes = runs on every element

However python is a *software engineering language* (algorithms, not maths)… so not every library will do this for you, and python itself does not.

[21]:
```python
prices * 0.05 # ERROR: python doesnt know what this means!
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-21-40af694f45cb> in <module>
----> 1 prices * 0.05 # ERROR: python doesnt know what this means!

TypeError: can't multiply sequence by non-int of type 'float'
```

Where this happens, comprehensions are the easiest way to do the same thing.

[19]:
```python
[ p * 0.05 for p in prices ]
```

[19]:
```
[0.05, 0.2, 3.25, 0.65]
```

1.3 Simple Examples

[32]:
```python
names = ["Alice A", "Eve E", "Bob B"]
```

lowercase all names,

[40]:
```python
[ n.lower() for n in names ]
```

[40]:
```
['alice a', 'eve e', 'bob b']
```

last letter of each name,

[39]:
```python
[ n[-1] for n in names ]
```

[39]:
```
['A', 'E', 'B']
```

length of all names,

[38]:
```python
[ len(n) for n in names ]
```

[38]:
```
[7, 5, 5]
```

split each name into two (at the space) and return the first part,

[35]:
```python
[ n.split()[0]  for n in names ]
```

[35]:
```
['Alice', 'Eve', 'Bob']
```

…ie., all the first names.

1.4 Complex Example

A list of dictionaries is quite a common data structure in python,

[50]:
```python
events = [
    {'subject': 'Alice', 'verb': 'SEND', 'object': 'Eve', 'context': 18},
    {'subject': 'Eve', 'verb': 'SEND', 'object': 'Alice', 'context': 8},
    {'subject': 'Bob', 'verb': 'SEND', 'object': 'Eve', 'context': 12},
]
```

Each element is a dictionary, so we can eg., find the first entry's subject field via,

[51]:
```python
events[0]['subject']
```

[51]:
```
'Alice'
```

Suppose we want to know the mean `context` (ie., mean number of messages sent),

[60]:
```python
from statistics import mean # import the mean function
```

[62]:
```python
mean( # mean of...
    [ e['context']  for e in events ] # each entry's context
)
```

[62]:
```
12.666666666666666
```

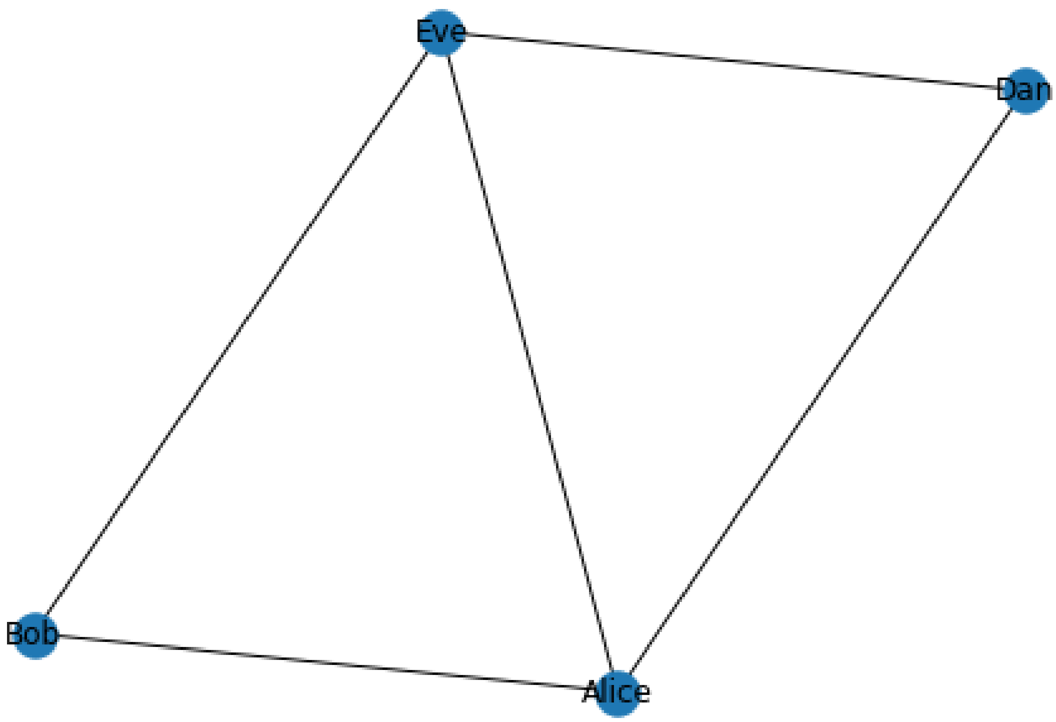
1.5 Example: Graph Analysis

ie., Social Network Analysis We can represent a social network as a sequence of connections (or "edges") between people,

[24]:
```python
S = [
    ('Alice', 'Eve', {'msg': 18})   , # alice messages eve 18 times / day
    ('Alice', 'Bob', {'msg': 6})    ,
    ('Alice', 'Dan', {'msg': 7})    ,
    ('Eve',   'Dan', {'msg': 2})    ,
    ('Bob',   'Eve', {'msg': 2})    ,
]
```

Aside: we can draw this using `networkx`

[28]:
```python
import networkx as nx
nx.draw(nx.from_edgelist(S), with_labels=True)
```



Let's compute the total number of messages sent,

[30]:
```
sum([ edge[-1]['msg'] for edge in S ])
```

[30]: 35

Or, more conveniently,

[31]:
```
sum([ weight['msg'] for frm, to, weight in S ])
```

[31]: 35

1.6 Exercise

1.6.1 Part 1: Modify Comprehensions

[54]:
```
sales = [1_000, 12_000, 3_5600] # per month
```

Modify the following to compute the discounted price (ie., after 0.9*) in* USD*. Ie., also multiply by a conversion ratio of 1.35.

[56]:
```
[ 0.9 * s for s in sales ]
```

[56]: [900.0, 10800.0, 32040.0]

[59]:
```python
items = ["iPad", "Mouse", "Mac Mini"]

[ i.lower() for i in items ]
```

[59]: ['ipad', 'mouse', 'mac mini']

Modify the above to uppercase the item name *and* return the first letter!

1.6.2 Part 2: Write Comprehensions

Choose your own problem domain (eg., health, retail, insurance, …).

Define two lists which represent categorical data (ie., text) and quantity data (ie., numbers).

Write a comprehension over both which summarises, or transforms these columns in a meaningful way.

Suggestions Eg., sleep: HIGH/LOW QUALITY; hr: 60, 60, …

- `len()` = 4 for H, 3 for L… mean of that…. is a kind of quality
- `sum()` on all hrs * 0.85… maybe .85 is the resting heart rate.

1.7 Appendix: Filters

We can also *filter* using comprehensions,

[63]:
```python
prices
```

[63]: [1, 4, 65, 13]

Keep only those above £5,

[65]:
```python
[ p for p in prices if p > 5]
```

[65]: [65, 13]

Discount them,

[67]:
```python
[ 0.9 * p for p in prices if p > 5]
```

[67]: [58.5, 11.700000000000001]

What's the mean price of discounted goods above £5 ?

[68]:
```python
mean([ 0.9 * p for p in prices if p > 5])
```

[68]: 35.1