1 Using masks to filter data, and perform search and replace, in NumPy and Pandas

In both NumPy and Pandas we can create masks to filter data. Masks are 'Boolean' arrays - that is arrays of true and false values and provide a powerful and flexible method to selecting data.

1.1 NumPy

1.1.1 creating a mask

Let's begin by creating an array of 4 rows of 10 columns of uniform random number between 0 and 100.

```
import numpy as np

array1 = np.random.randint(0,100,size=(4,10))

print (array1)

OUT:

[[68 56 72 91 64 98  3 54 49 67]
 [ 1  6 54 65 24 97 68  9 28 47]
 [30 88 52 11 22 12 35 65 66  3]
 [13 83 81 32 87 74 79 34 26  1]]
```

Now we'll create a mask to show those numbers greater than 70.

```
mask = array1 > 70

print(mask)

OUT:

[[False False  True  True False  True False False False False]
 [False False False False False  True False False False False]
 [False  True False False False False False False False False]
 [False  True  True False  True  True  True False False False]]
```

We can use that mask to extract the numbers:

```
print (array1[mask])

OUT:
[72 91 98 97 88 83 81 87 74 79]
```

1.1.2 Using *any* and *all*

any and all allow us to check for all true or all false.

We can apply that to the whole array:

```
print (mask.any())
print (mask.all())

OUT:

True
False
```

Or we can apply it column-wise (by passing *axis=1*) or row-wise (by passing *axis=1*).

```
print ('All tests in a column are true:')
print (mask.all(axis=0))
print ('\nAny test in a row is true:')
print (mask.any(axis=1))

OUT:

All tests in a column are true:
[False False False False False False False False False False]

Any test in a row is true:
[False  True  True  True]
```

We can use != to invert a mask if needed (all trues become false, and all falses become true). This can be useful, but can also become a little confusing!

```
inverted_mask = mask!=True
print (inverted_mask)

OUT:

[[ True  True  True  True  True  True  True  True  True  True]
 [ True  True False False  True False  True False False False]
 [False False False  True False  True  True  True False  True]
 [ True False  True  True  True  True  True  True  True  True]]
```

1.1.3 Adding or averaging trues

Boolean values (True/False) in Python also take the values 1 and 0. This can be useful for counting trues/false, for example:

```
print ('Number of trues in array:')
print (mask.sum())

OUT:
Number of trues in array:
12

print('Number of trues in array by row:')
print (mask.sum(axis=1))

OUT:
Number of trues in array by row:
[0 6 5 1]

print('Average of trues in array by column:')
print (mask.mean(axis=0))

OUT:
Average of trues in array by column:
[0.25 0.5  0.5  0.25 0.25 0.25 0.   0.25 0.5  0.25]
```

1.1.4 Selecting rows or columns based on one value in that row or column

Let's select all columns where the value of the first element is equal to, or greater than 50:

```
mask = array1[0,:] >= 50 # colon indicates all columns, zero indicates row 0
```

```
print ('\nHere is the mask')
print (mask)
print ('\nAnd here is the mask applied to all columns')
print (array1[:,mask]) # colon represents all rows of chosen columns

OUT:

Here is the mask
[ True  True False False  True False False False False  True]

And here is the mask applied to all columns
[[60 68 57 57]
 [47 52 64 74]
 [77 78 79 21]
 [11 78  5 14]]
```

Similarly if we wanted to select all rows where the 2nd element was equal to, or greater, than 50

```
mask = array1[:,1] >= 50 # colon indicates all rows, 1 indicates row 1)
print ('\nHere is the mask')
print (mask)
print ('\nAnd here is the mask applied to all rows')
print (array1[mask,:]) # colon represents all rows of chosen columns

OUT:


Here is the mask
[ True  True  True  True]

And here is the mask applied to all rows
[[60 68 34 25 57 33 49  5 33 57]
 [47 52 78 89 64 75  8 98 93 74]
 [77 78 74 41 79 50 43 21 81 21]
 [11 78 39 18  5 67 69  1 50 14]]
```

1.1.5 Using *and* and *or*, and combining filters from two arrays

We may create and combine multiple masks. For example we may have two masks that look for values less than 20 or greater than 80, and then combine those masks with or which is represented by — (stick).

```
print ('Mask for values <20:')
mask1 = array1 < 20
print (mask1)

print ('\nMask for values >80:')
mask2 = array1 > 80
print (mask2)

print ('\nCombined mask:')
mask = mask1  | mask2 # | (stick) is used for 'or' with two boolean arrays
print (mask)

print ('\nSelected values using combined mask')
print (array1[mask])

OUT:

Mask for values <20:
```

```
[[False False False False False False False  True False False]
 [False False False False False False  True False False False]
 [False False False False False False False False False False]
 [ True False False  True  True False False  True False  True]]

Mask for values >80:
[[False False False False False False False False False False]
 [False False False  True False False False  True  True False]
 [False False False False False False False False  True False]
 [False False False False False False False False False False]]

Combined mask:
[[False False False False False False False  True False False]
 [False False False  True False False  True  True  True False]
 [False False False False False False False False  True False]
 [ True False False  True  True False False  True False  True]]

Selected values using combined mask
[ 5 89  8 98 93 81 11 18  5  1 14]
```

We can combine these masks in a single line:

```
mask = (array1 < 20) | (array1 > 80)
print (mask)

OUT:

mask = (array1 < 20) | (array1 > 80)

print (mask)

[[False False False False False False False  True False False]
 [False False False  True False False  True  True  True False]
 [False False False False False False False False  True False]
 [ True False False  True  True False False  True False  True]]
```

We can combine masks derived from different arrays, so long as they are the same shape. For example let's produce an another array of random numbers and check for those element positions where corresponding positions of both arrays have values of greater than 50. When comparing boolean arrays we represent 'and' with &.

```
array2 = np.random.randint(0,100,size=(4,10))

print ('Mask for values of array1 > 50:')
mask1 = array1 > 50
print (mask1)

print ('\nMask for values of array2 > 50:')
mask2 = array2 > 50
print (mask2)

print ('\nCombined mask:')
mask = mask1  & mask2
print (mask)

OUT:

Mask for values of array1 > 50:
[[ True  True False False  True False False False False  True]
 [False  True  True  True  True  True False  True  True  True]
 [ True  True  True False  True False False False  True False]
 [False  True False False False  True  True False False False]]

Mask for values of array2 > 50:
[[ True False  True False False  True  True False False  True]
 [ True False  True  True False  True  True  True False False]
 [False  True  True False  True False  True  True  True False]
 [ True False  True False False  True  True  True  True  True]]

Combined mask:
[[ True False False False False False False False False  True]
 [False False  True  True False  True False  True False False]
 [False  True  True False  True False False False  True False]
 [False False False False False  True  True False False False]]
```

We could shorten this to:

```
mask = (array1 > 50) & (array2 > 50)
print (mask)

OUT:

[[ True False False False False False False False False  True]
 [False False  True  True False  True False  True False False]
 [False  True  True False  True False False False  True False]
 [False False False False False  True  True False False False]]
```

1.1.6 Setting values based on mask

We can use masks to reassign values only for elements that meet the given criteria. For example we can set the values of all cells with a value less than 50 to zero, and set all other values to 1.

```
print ('Array at sttart:')
print (array1)
mask = array1 < 50
array1[mask] = 0
mask = mask != True # invert mask
array1[mask] = 1
print('\nNew array')
print (array1)

OUT:

Array at sttart:
[[60 68 34 25 57 33 49  5 33 57]
 [47 52 78 89 64 75  8 98 93 74]
 [77 78 74 41 79 50 43 21 81 21]
 [11 78 39 18  5 67 69  1 50 14]]

New array
[[1 1 0 0 1 0 0 0 0 1]
 [0 1 1 1 1 1 0 1 1 1]
 [1 1 1 0 1 1 0 0 1 0]
 [0 1 0 0 0 1 1 0 1 0]]
```

We can shorten this, by making the mask implicit in the assignment command.

```
array2[array2<50] = 0
array2[array2>=50] = 1
```

```
print('New array2:')
print(array2)

OUT:

New array2:
[[1 0 1 0 0 1 1 0 0 1]
 [1 0 1 1 0 1 1 1 0 0]
 [0 1 1 0 1 0 1 1 1 0]
 [1 0 1 0 0 1 1 1 1 1]]
```

1.1.7 Miscellaneous examples

Select columns where the average value across the column is greater than the average across the whole array, and return both the columns and the column number.

```
array = np.random.randint(0,100,size=(4,10))
number_of_columns = array.shape[1]
column_list = np.arange(0, number_of_columns) # create a list of column ids
array_average = array.mean()
column_average = array.mean(axis=0)
column_average_greater_than_array_average = column_average > array_average
selected_columns = column_list[column_average_greater_than_array_average]
selected_data = array[:,column_average_greater_than_array_average]

print ('Selected columns:')
print (selected_columns)
print ('\nSeelcted data:')
print (selected_data)

OUT:

Selected columns:
[1 2 4 9]

Seelcted data:
[[56 48 97 78]
 [26 87  6 45]
 [56 65 71 59]
 [41 34 98 70]]
```

1.2 Pandas

Filtering with masks in Pandas is very similar to numpy. It is perhaps more usual in Pandas to be creating masks testing specific columns, with resulting selection of rows. For example let's use a mask to select characters meeting conditions on magical power and aggression:

```
i
import pandas as pd

df = pd.DataFrame()

names = ['Gandolf','Gimli','Frodo','Legolas','Bilbo']
types = ['Wizard','Dwarf','Hobbit','Elf','Hobbit']
magic = [10, 1, 4, 6, 4]
aggression = [7, 10, 2, 5, 1]
stealth = [8, 2, 5, 10, 5]
```

```
df['names'] = names
df['type'] = types
df['magic_power'] = magic
df['aggression'] = aggression
df['stealth'] = stealth

mask = (df['magic_power'] > 3) & (df['aggression'] < 5)
print ('Mask:')
print (mask) # notice mask is a 'series'; a one dimensial DataFrame
# when passing a Boolean series to a dataframe we select the appropriate rows
filtered_data = df[mask]
print (filtered_data)

OUT:

Mask:
0    False
1    False
2     True
3    False
4     True
dtype: bool
   names    type  magic_power  aggression  stealth
2  Frodo  Hobbit            4           2        5
4  Bilbo  Hobbit            4           1        5
```

Though creating masks based on particular columns will be most common in Pandas. We can also filter on the entire dataframe. Look what happens when we filter on values >3:

```
mask = df > 3
print('Mask:')
print (mask)
print ('\nMasked data:')
df2 = df[mask]
print (df2)

OUT:

Mask:
   names  type  magic_power  aggression  stealth
0   True  True         True        True     True
1   True  True        False        True    False
2   True  True         True       False     True
3   True  True         True        True     True
4   True  True         True       False     True

Masked data:
     names    type  magic_power  aggression  stealth
0  Gandolf  Wizard         10.0         7.0      8.0
1    Gimli   Dwarf          NaN        10.0      NaN
2    Frodo  Hobbit          4.0         NaN      5.0
3  Legolas     Elf          6.0         5.0     10.0
4    Bilbo  Hobbit          4.0         NaN      5.0
```

The structure of the dataframe is maintained, and all text is maintained. Those values not >3 have been removed (NaN represents 'Not a Number').

1.2.1 Conditional replacing of values in Pandas

Replacing values in Pandas, based on the current value, is not as simple as in NumPy. For example, to replace all values in a given column, given a conditional test, we have to (1) take one column at a time, (2) extract the column values into an array, (3) make our replacement, and (4) replace the column values with our adjusted array.

For example to replace all values less than 4 with zero (in our numeric columns):

```
columns = ['magic_power','aggression','stealth']
# note: to get a list of all columns you can use list(df)

for column in columns: # loop through our column list
    values = df[column].values # extract the column values into an array
    mask = values < 4 # create Boolean mask
    values [mask] = 0 # apply Boolean mask
    df[column] = values # replace the dataframe column with the array

print (df)

OUT:

     names    type  magic_power  aggression  stealth
0  Gandolf  Wizard           10           7        8
1    Gimli   Dwarf            0          10        0
2    Frodo  Hobbit            4           0        5
3  Legolas     Elf            6           5       10
4    Bilbo  Hobbit            4           0        5
```