Numpy Lecture - 2 [Fitbit Analysis]

In [1]:
```
import numpy as np
```

In [2]:
```
arr = np.array([1, 2, 3, 4])
arr.dtype
```

Out[2]:
```
dtype('int64')
```

In [3]:
```
arr = np.array([1, 2, 3, 4], dtype='float')
arr.dtype
```

Out[3]:
```
dtype('float64')
```

In [4]:
```
arr
```

Out[4]:
```
array([1., 2., 3., 4.])
```

In [6]:
```
arr = np.array([1, 2, 3, 4])
```

In [8]:
```
arr1 = arr.astype('float')
arr1.dtype
```

Out[8]:
```
dtype('float64')
```

Masking [Fancy Indexing]

In [9]:
```
arr = np.arange(1, 11)
arr
```

Out[9]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [10]:
```
arr[1]
```

Out[10]:
```
2
```

In [11]:
```
arr[[1, 0, 1, 2, 3, 1]]
```

Out[11]:
```
array([2, 1, 2, 3, 4, 2])
```

In [12]:
```
arr < 6
```

Out[12]:
```
array([ True,  True,  True,  True,  True, False, False, False, False,
       False])
```

In [13]:
```
arr[arr<6]
```

Out[13]:
```
array([1, 2, 3, 4, 5])
```

In [14]:
```
#Filter even values present in a given array
```

In [15]:
```
arr %2 == 0
```

Out[15]:
```
array([False,  True, False,  True, False,  True, False,  True, False,
        True])
```

In [16]:
```
arr[arr%2==0]
```

Out[16]:
```
array([ 2,  4,  6,  8, 10])
```

In [17]:
```
#multiples of 2 or multiples of 5
```

In [18]:
```
arr[(arr %2 == 0) | (arr%5 == 0)]
```

Out[18]:
```
array([ 2,  4,  5,  6,  8, 10])
```

In [19]:
```
arr[(arr %2 == 0) & (arr%5 == 0)]
```

Out[19]:
```
array([10])
```

where()

In [20]:
```
m1 = np.arange(1, 11)
np.where(m1<6)
```

Out[20]:
```
(array([0, 1, 2, 3, 4]),)
```

In [21]:
```
a = np.array([[1, 2, 3], [4, 5, 6]])
np.where(a<6, )
```

Out[21]:
```
(array([0, 0, 0, 1, 1]), array([0, 1, 2, 0, 1]))
```

In [22]:
```
help(np.where)
```

```
Help on function where in module numpy:

where(...)
    where(condition, [x, y], /)

    Return elements chosen from `x` or `y` depending on `condition`.

    .. note::
        When only `condition` is provided, this function is a shorthand for
        ``np.asarray(condition).nonzero()``. Using `nonzero` directly should be
        preferred, as it behaves correctly for subclasses. The rest of this
        documentation covers only the case where all three arguments are
        provided.

    Parameters
    ----------
    condition : array_like, bool
        Where True, yield `x`, otherwise yield `y`.
    x, y : array_like
        Values from which to choose. `x`, `y` and `condition` need to be
        broadcastable to some shape.

    Returns
    -------
    out : ndarray
        An array with elements from `x` where `condition` is True, and elements
        from `y` elsewhere.

    See Also
    --------
    choose
    nonzero : The function that is called when x and y are omitted

    Notes
    -----
    If all the arrays are 1-D, `where` is equivalent to::

        [xv if c else yv
         for c, xv, yv in zip(condition, x, y)]

    Examples
    --------
    >>> a = np.arange(10)
    >>> a
    array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    >>> np.where(a < 5, a, 10*a)
    array([ 0,  1,  2,  3,  4, 50, 60, 70, 80, 90])

    This can be used on multidimensional arrays too:

    >>> np.where([[True, False], [True, True]],
    ...          [[1, 2], [3, 4]],
    ...          [[9, 8], [7, 6]])
    array([[1, 8],
           [3, 4]])

    The shapes of x, y, and the condition are broadcast together:

    >>> x, y = np.ogrid[:3, :4]
    >>> np.where(x < y, x, 10 + y)  # both x and 10+y are broadcast
    array([[10,  0,  0,  0],
           [10, 11,  1,  1],
           [10, 11, 12,  2]])

    >>> a = np.array([[0, 1, 2],
    ...               [0, 2, 4],
    ...               [0, 3, 6]])
    >>> np.where(a < 4, a, -1)  # -1 is broadcast
    array([[ 0,  1,  2],
           [ 0,  2, -1],
           [ 0,  3, -1]])
```

In [23]:
```
np.where(arr<6, arr, arr*10)
```

Out[23]:
```
array([  1,   2,   3,   4,   5,  60,  70,  80,  90, 100])
```

In [24]:
```
a = np.array([-1, 2, -3, 4, 5])
```

```
np.where(a<0, a*(-10), a)
```

Out[24]:
```
array([10,  2, 30,  4,  5])
```

NPS Solution

In [25]:
```
!gdown 1c0ClC8SrPwJq5rrkyMKyPn80nyHcFikK
```

```
Downloading...
From: https://drive.google.com/uc?id=1c0ClC8SrPwJq5rrkyMKyPn80nyHcFikK
To: /Users/satish/Desktop/scaler/Dec Tue Batch - DAV-1/survey.txt
100%|████████████████████████████████████████| 2.55k/2.55k [00:00<00:00, 3.20MB/s]
```

In [26]:
```
score = np.loadtxt('survey.txt', dtype='int')
```

In [27]:
```
score
```

Out[27]:
```
array([ 7, 10,  5, ...,  5,  9, 10])
```

In [28]:
```
score[:5]
```

Out[28]:
```
array([ 7, 10,  5,  9,  9])
```

In [29]:
```
score.ndim
```

Out[29]:
```
1
```

In [30]:
```
score.shape
```

Out[30]:
```
(1167,)
```

In [31]:
```
score.min()
```

Out[31]:
```
1
```

In [32]:
```
score.max()
```

Out[32]:
```
10
```

In [33]:
```
#NPS = %PROMOTERS - %DETRACTORS
```

In [44]:
```
promoters = score[score>=9].shape[0]
```

In [38]:
```
detractors = score[score<=6].shape[0]
```

In [39]:
```
total = score.shape[0]
```

In [42]:
```
percent_promoters = promoters / total * 100
percent_detractors = detractors / total * 100
nps = percent_promoters - percent_detractors
nps
```

Out[42]:
```
23.73607540702657
```

In [45]:
```
score[score <=6] = 'Detractors'
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [45], in <cell line: 1>()
----> 1 score[score <=6] = 'Detractors'

ValueError: invalid literal for int() with base 10: 'Detractors'
```

In [46]:
```
help(np.empty)
```

```
Help on built-in function empty in module numpy:

empty(...)
    empty(shape, dtype=float, order='C', *, like=None)

    Return a new array of given shape and type, without initializing entries.

    Parameters
    ----------
    shape : int or tuple of int
        Shape of the empty array, e.g., ``(2, 3)`` or ``2``.
    dtype : data-type, optional
        Desired output data-type for the array, e.g, `numpy.int8`. Default is
        `numpy.float64`.
    order : {'C', 'F'}, optional, default: 'C'
        Whether to store multi-dimensional data in row-major
        (C-style) or column-major (Fortran-style) order in
        memory.
    like : array_like
        Reference object to allow the creation of arrays which are not
        NumPy arrays. If an array-like passed in as ``like`` supports
        the ``__array_function__`` protocol, the result will be defined
        by it. In this case, it ensures the creation of an array object
        compatible with that passed in via this argument.

        .. versionadded:: 1.20.0

    Returns
    -------
    out : ndarray
        Array of uninitialized (arbitrary) data of the given shape, dtype, and
        order.  Object arrays will be initialized to None.

    See Also
    --------
    empty_like : Return an empty array with shape and type of input.
    ones : Return a new array setting values to one.
    zeros : Return a new array setting values to zero.
    full : Return a new array of given shape filled with value.


    Notes
    -----
    `empty`, unlike `zeros`, does not set the array values to zero,
    and may therefore be marginally faster.  On the other hand, it requires
    the user to manually set all the values in the array, and should be
    used with caution.

    Examples
    --------
    >>> np.empty([2, 2])
    array([[ -9.74499359e+001,   6.69583040e-309],
           [  2.13182611e-314,   3.06959433e-309]])         #uninitialized

    >>> np.empty([2, 2], dtype=int)
    array([[-1073741821, -1067949133],
           [  496041986,    19249760]])                     #uninitialized
```

In [47]:
```
arr = np.empty(shape=score.shape, dtype='str')
arr
```

Out[47]:
```
array(['', '', '', ..., '', '', ''], dtype='<U1')
```

In [48]:
```
arr[0] = 'Hello'
arr
```

Out[48]:
```
array(['H', '', '', ..., '', '', ''], dtype='<U1')
```

In [49]:
```
arr = np.empty(shape=score.shape, dtype='U10')
arr
```

Out[49]:
```
array(['', '', '', ..., '', '', ''], dtype='<U10')
```

In [50]:
```
arr[score <=6] = 'Detractors'
```

```
arr[score >=9] = 'Promoters'
arr[(score>=7) & (score <= 8)] = 'Passive'
arr
```

Out[50]:
```
array(['Passive', 'Promoters', 'Detractors', ..., 'Detractors',
       'Promoters', 'Promoters'], dtype='<U10')
```

In [51]:
```
arr.shape
```

Out[51]:
```
(1167,)
```

In [52]:
```
detractor_count = arr[arr == 'Detractors'].shape[0]
promoter_count = arr[arr=='Promoters'].shape[0]
```

In [53]:
```
np.unique(arr)
```

Out[53]:
```
array(['Detractors', 'Passive', 'Promoters'], dtype='<U10')
```

In [54]:
```
help(np.unique)
```

```
Help on function unique in module numpy:

unique(ar, return_index=False, return_inverse=False, return_counts=False, axis=None)
    Find the unique elements of an array.

    Returns the sorted unique elements of an array. There are three optional
    outputs in addition to the unique elements:

    * the indices of the input array that give the unique values
    * the indices of the unique array that reconstruct the input array
    * the number of times each unique value comes up in the input array

    Parameters
    ----------
    ar : array_like
        Input array. Unless `axis` is specified, this will be flattened if it
        is not already 1-D.
    return_index : bool, optional
        If True, also return the indices of `ar` (along the specified axis,
        if provided, or in the flattened array) that result in the unique array.
    return_inverse : bool, optional
        If True, also return the indices of the unique array (for the specified
        axis, if provided) that can be used to reconstruct `ar`.
    return_counts : bool, optional
        If True, also return the number of times each unique item appears
        in `ar`.

        .. versionadded:: 1.9.0

    axis : int or None, optional
        The axis to operate on. If None, `ar` will be flattened. If an integer,
        the subarrays indexed by the given axis will be flattened and treated
        as the elements of a 1-D array with the dimension of the given axis,
        see the notes for more details.  Object arrays or structured arrays
        that contain objects are not supported if the `axis` kwarg is used. The
        default is None.

        .. versionadded:: 1.13.0

    Returns
    -------
    unique : ndarray
        The sorted unique values.
    unique_indices : ndarray, optional
        The indices of the first occurrences of the unique values in the
        original array. Only provided if `return_index` is True.
    unique_inverse : ndarray, optional
        The indices to reconstruct the original array from the
        unique array. Only provided if `return_inverse` is True.
    unique_counts : ndarray, optional
        The number of times each of the unique values comes up in the
        original array. Only provided if `return_counts` is True.

        .. versionadded:: 1.9.0

    See Also
```

```
--------
numpy.lib.arraysetops : Module with a number of other functions for
                        performing set operations on arrays.
repeat : Repeat elements of an array.

Notes
-----
When an axis is specified the subarrays indexed by the axis are sorted.
This is done by making the specified axis the first dimension of the array
(move the axis to the first dimension to keep the order of the other axes)
and then flattening the subarrays in C order. The flattened subarrays are
then viewed as a structured type with each element given a label, with the
effect that we end up with a 1-D array of structured types that can be
treated in the same way as any other 1-D array. The result is that the
flattened subarrays are sorted in lexicographic order starting with the
first element.

.. versionchanged: NumPy 1.21
    If nan values are in the input array, a single nan is put
    to the end of the sorted unique values.

    Also for complex arrays all NaN values are considered equivalent
    (no matter whether the NaN is in the real or imaginary part).
    As the representant for the returned array the smallest one in the
    lexicographical order is chosen - see np.sort for how the lexicographical
    order is defined for complex arrays.

Examples
--------
>>> np.unique([1, 1, 2, 2, 3, 3])
array([1, 2, 3])
>>> a = np.array([[1, 1], [2, 3]])
>>> np.unique(a)
array([1, 2, 3])

Return the unique rows of a 2D array

>>> a = np.array([[1, 0, 0], [1, 0, 0], [2, 3, 4]])
>>> np.unique(a, axis=0)
array([[1, 0, 0], [2, 3, 4]])

Return the indices of the original array that give the unique values:

>>> a = np.array(['a', 'b', 'b', 'c', 'a'])
>>> u, indices = np.unique(a, return_index=True)
>>> u
array(['a', 'b', 'c'], dtype='<U1')
>>> indices
array([0, 1, 3])
>>> a[indices]
array(['a', 'b', 'c'], dtype='<U1')

Reconstruct the input array from the unique values and inverse:

>>> a = np.array([1, 2, 6, 4, 2, 3, 2])
>>> u, indices = np.unique(a, return_inverse=True)
>>> u
array([1, 2, 3, 4, 6])
>>> indices
array([0, 1, 4, 3, 1, 2, 1])
>>> u[indices]
array([1, 2, 6, 4, 2, 3, 2])

Reconstruct the input values from the unique values and counts:

>>> a = np.array([1, 2, 6, 4, 2, 3, 2])
>>> values, counts = np.unique(a, return_counts=True)
>>> values
array([1, 2, 3, 4, 6])
>>> counts
array([1, 3, 1, 1, 1])
>>> np.repeat(values, counts)
array([1, 2, 2, 2, 3, 4, 6])    # original order not preserved
```

In [56]:
```
unique, counts = np.unique(arr, return_counts=True)
```

In [57]:
```
counts
```

Out[57]:
```
array([332, 226, 609])
```

In [58]:
```
percent_detractors = counts[0]/counts.sum()*100
percent_promoters = counts[2] /counts.sum()*100
nps = percent_promoters - percent_detractors
nps
```

Out[58]:
```
23.73607540702657
```

Logical Functions

In [59]:
```
a = np.array([1, 2, 3, 4])
b = np.array([4, 3, 2, 1])
np.any(a < b)
```

Out[59]:
```
True
```

In [60]:
```
np.all(a < b)
```

Out[60]:
```
False
```

In [61]:
```
a = np.array([[1, 2, 3], [4, 5, 6]])
a.ndim
```

Out[61]:
```
2
```

In [62]:
```
a.shape
```

Out[62]:
```
(2, 3)
```

In [63]:
```
a.size
```

Out[63]:
```
6
```

In [64]:
```
arr = np.arange(1, 13)
```

In [65]:
```
arr.reshape(4, 4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [65], in <cell line: 1>()
----> 1 arr.reshape(4, 4)

ValueError: cannot reshape array of size 12 into shape (4,4)
```

In [66]:
```
## What are the ways we can reshape the given array
```

In [67]:
```
## 4*3, 3*4, 2*6, 6*2, 1*12, 12*1
```

In [69]:
```
a = arr.reshape(3, 4)
```

In [70]:
```
a.shape
```

Out[70]:
```
(3, 4)
```

In [71]:
```
a
```

Out[71]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [72]:
```
a.T
```

Out[72]:
```
array([[ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11],
       [ 4,  8, 12]])
```

In [73]:
```
a = np.array([1, 2, 3])
a.T
```

Out[73]:
```
array([1, 2, 3])
```

In [74]:
```
a = np.array([[1, 2, 3]])
a.shape
```

Out[74]:
```
(1, 3)
```

In [75]:
```
a.ndim
```

Out[75]:
```
2
```

In [76]:
```
a.T
```

Out[76]:
```
array([[1],
       [2],
       [3]])
```

In [ ]:
```

```