

INTRO TO PANDAS & NUMPY

INTRO TO PANDAS & NUMPY



In this section we'll introduce **Pandas** & **NumPy**, two critical Python libraries that help structure data in arrays & DataFrames and contain built-in functions for data analysis

TOPICS WE'LL COVER:

Intro to Pandas & NumPy

NumPy Array Basics

Array Creation

Array Indexing & Slicing

Array Operations

Vectorization & Broadcasting

GOALS FOR THIS SECTION:

- Convert Python lists to NumPy arrays, and create new arrays from scratch using functions
- Apply array indexing, slicing, methods, and functions to perform operations on NumPy arrays
- Understand the concepts of vectorization and broadcasting, which are critical in making NumPy and Pandas more efficient than base Python



MEET PANDAS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting



Pandas is Python's most widely used library for data analysis, and contains functions for accessing, aggregating, joining, and analyzing data

Its data structure, the DataFrame, is analogous to SQL tables or Excel worksheets

Custom indices & column titles make working with datasets more intuitive!



	id	date	store_nbr	family	sales	onpromotion
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0
1	1	2013-01-01	1	BABY CARE	0.0	0
2	2	2013-01-01	1	BEAUTY	0.0	0
3	3	2013-01-01	1	BEVERAGES	0.0	0
4	4	2013-01-01	1	BOOKS	0.0	0



MEET NUMPY

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting



NumPy is an open-source library that is the universal standard for working with numerical data in Python, and forms the foundation of other libraries like Pandas

Pandas DataFrames are built on NumPy arrays and can leverage NumPy functions

*The indices, column names,
and data columns are all
stored as NumPy arrays*

*Pandas just adds convenient
wrappers and functions!*

	id	date	store_nbr	family	sales	onpromotion
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0
1	1	2013-01-01	1	BABY CARE	0.0	0
2	2	2013-01-01	1	BEAUTY	0.0	0
3	3	2013-01-01	1	BEVERAGES	0.0	0
4	4	2013-01-01	1	BOOKS	0.0	0



NUMPY ARRAYS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

NumPy arrays are fixed-size containers of items that are more efficient than Python lists or tuples for data processing

- They only store a single data type (*mixed data types are stored as a string*)
- They can be one dimensional or multi-dimensional
- Array elements can be modified, but the array size cannot change

```
import numpy as np
```

'np' is the standard alias for the NumPy library

```
sales = [0, 5, 155, 0, 518, 0, 1827, 616, 317, 325]
```

```
sales_array = np.array(sales)  
sales_array
```

```
array([ 0, 5, 155, 0, 518, 0, 1827, 616, 317, 325])
```

NumPy's `array` function
converts Python lists and
tuples into NumPy arrays



ARRAY PROPERTIES

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

NumPy arrays have these key properties:

- **ndim** – the number of dimensions (axes) in the array
- **shape** – the size of the array for each dimension
- **size** – the total number of elements in the array
- **dtype** – the data type of the elements in the array

```
sales = [0, 5, 155, 0, 518, 0, 1827, 616, 317, 325]
```

```
sales_array = np.array(sales)
```

```
type(sales_array)
```

```
numpy.ndarray
```

NumPy arrays are a **ndarray** Python data type,
which stands for n-dimensional array

```
print(f"ndim: {sales_array.ndim}")  
print(f"shape: {sales_array.shape}")  
print(f"size: {sales_array.size}")  
print(f"dtype: {sales_array.dtype}")
```

ndim: 1 → The sales_array has 1 dimension
shape: (10,) → The dimension has a size of 10
size: 10 → The array has 10 elements total
dtype: int64 → The elements are stored as 64-bit integers



ARRAY PROPERTIES

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

NumPy arrays have these key properties:

- **ndim** – the number of dimensions (axes) in the array
- **shape** – the size of the array for each dimension
- **size** – the total number of elements in the array
- **dtype** – the data type of the elements in the array

```
sales = [[0, 5, 155, 0, 518], [0, 1827, 616, 317, 325]]
```

```
sales_array = np.array(sales)
```

```
array([[ 0,    5,  155,    0,  518],  
       [ 0, 1827,  616,  317,  325]])
```

Converting a nested list creates a multi-dimensional array, where each nested list is a dimension

NOTE: The nested lists must be of equal length

```
print(f"ndim: {sales_array.ndim}")  
print(f"shape: {sales_array.shape}")  
print(f"size: {sales_array.size}")  
print(f"dtype: {sales_array.dtype}")
```

ndim: 2 → The sales_array has 2 dimensions

shape: (2, 5) → The first dimension has a size of 2 (rows) and the second a size of 5 (columns)

size: 10 → It has 10 elements total

dtype: int64 → The elements are stored as 64-bit integers

ASSIGNMENT: ARRAY BASICS



NEW MESSAGE

June 16, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **NumPy?**

Hi there, welcome to the Maven MegaMart!

Your resume mentions you have basic Python experience. Our finance team has been asking us to cut software costs – can you help us dig into Python as an analysis tool?

I know NumPy is foundational, but not much beyond that.

Can you convert a Python list into a NumPy array and help me get familiar with their properties?

Thanks!

 `numpy_assignments.ipynb`

 Reply

 Forward

Results Preview

```
my_list = [x * 10 for x in range(1, 11)]
```

```
array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100])
```

```
ndim: 1  
shape: (10,)  
size: 10  
dtype: int64
```


SOLUTION: ARRAY BASICS



NEW MESSAGE

June 16, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **NumPy?**

Hi there, welcome to the Maven MegaMart!

Your resume mentions you have basic Python experience. Our finance team has been asking us to cut software costs – can you help us dig into Python as an analysis tool?

I know NumPy is foundational, but not much beyond that.

Can you convert a Python list into a NumPy array and help me get familiar with their properties?

Thanks!

 `numpy_assignments.ipynb`

 Reply

 Forward

Solution Code

```
my_list = [x * 10 for x in range(1, 11)]  
  
my_array = np.array(my_list)  
  
my_array  
  
array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100])
```

```
print(f"ndim: {my_array.ndim}")  
print(f"shape: {my_array.shape}")  
print(f"size: {my_array.size}")  
print(f"dtype: {my_array.dtype}")
```

```
ndim: 1  
shape: (10,)  
size: 10  
dtype: int64
```



ARRAY CREATION

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

As an alternative to converting lists, you can **create arrays** using functions

ones

Creates an array of ones of a given size, as float by default

`np.ones((rows, cols), dtype)`

zeros

Creates an array of zeros of a given size, as float by default

`np.zeros((rows, cols), dtype)`

arange

Creates an array of integers with given start & stop values, and a step size (only stop is required, and is not inclusive)

`np.arange(start, stop, step)`

linspace

Creates an array of floats with given start & stop values with n elements, separated by a consistent step size (stop is inclusive)

`np.linspace(start, stop, n)`

reshape

Changes an array into the specified dimensions, if compatible

`np.array.reshape(rows, cols)`



ARRAY CREATION

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

As an alternative to converting lists, you can **create arrays** using functions

np.ones((*rows*, *cols*), *dtype*)

```
np.ones(4,)
```

```
array([1., 1., 1., 1.])
```

np.zeros((*rows*, *cols*), *dtype*)

```
np.zeros((2, 5), dtype=int)
```

```
array([[0, 0, 0, 0, 0],  
       [0, 0, 0, 0, 0]])
```

np.arange(*start*, *stop*, *step*) *start is 0 and step is 1 by default*

```
np.arange(10)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

stop is not inclusive

np.linspace(*start*, *stop*, *n*)

```
np.linspace(0, 100, 5)
```

```
array([ 0., 25., 50., 75., 100.])
```

stop is inclusive

np.array.reshape(*rows*, *cols*)

```
np.arange(1, 9, 2).reshape(2, 2)
```

```
array([[1, 3],  
       [5, 7]])
```



RANDOM NUMBER ARRAYS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

You can create **random number arrays** from a variety of distributions using NumPy functions and methods (*great for sampling and simulation!*)

default_rng

Creates a random number generator (the seed is for reproducibility)

np.default_rng(seed)

random

Returns n random numbers from a uniform distribution between 0 and 1

mg.random(n)

normal

Returns n random numbers from a normal distribution with a given mean and standard deviation

mg.normal(mean, stdev, n)

'mg' is the standard variable name for
the default_mg number generator



RANDOM NUMBER ARRAYS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

You can create **random number arrays** from a variety of distributions using NumPy functions and methods (*great for sampling and simulation!*)

```
from numpy.random import default_rng

rng = default_rng(12345)

random_array = rng.random(10)
random_array
```

```
array([0.22733602, 0.31675834, 0.79736546, 0.67625467, 0.39110955,
       0.33281393, 0.59830875, 0.18673419, 0.67275604, 0.94180287])
```

First, we're creating a random number generator with a seed of 12345 and assigning it to 'rng' using **default_rng**

Then we're using the **random** method on 'rng' to return an array with 10 random numbers

```
rng = default_rng(12345)
mean, stddev = 5, 1
random_normal = rng.normal(mean, stddev, size=10)
random_normal
```

```
array([3.57617496, 6.26372846, 4.12933826, 4.74082677, 4.92465669,
       4.25911535, 3.6322073 , 5.6488928 , 5.36105811, 3.04713694])
```

Here we're using the **normal** method on 'rng' to return an array with 10 random numbers from a normal distribution with a mean of 5 and a st. deviation of 1



PRO TIP: Even though it's optional, make sure to **set a seed** when generating random numbers to ensure you and others can recreate the work you've done (the value for the seed is less important)

ASSIGNMENT: ARRAY CREATION



NEW MESSAGE

June 17, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **Array Creation**

Thanks for your help last time – I’m starting to understand NumPy Arrays!

Are there any NumPy functions that can create arrays so we don’t have to convert from a Python list? Recreate the array from the first assignment but make it 5 rows by 2 columns.

Once you’ve done that, create an array of random numbers between 0 and 1 in a 3x3 shape. One of our data scientists has been asking about this so I want to make sure it’s possible.

Thanks!

 numpy_assignments.ipynb

 Reply

 Forward

Results Preview

```
array([[ 10.,  20.],
       [ 30.,  40.],
       [ 50.,  60.],
       [ 70.,  80.],
       [ 90., 100.]])
```

```
random_array
array([[0.24742606, 0.09299006, 0.61176337],
       [0.06066207, 0.66103343, 0.75515778],
       [0.1108689 , 0.04305584, 0.41441747]])
```

SOLUTION: ARRAY CREATION



NEW MESSAGE

June 17, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **Array Creation**

Thanks for your help last time – I’m starting to understand NumPy Arrays!

Are there any NumPy functions that can create arrays so we don’t have to convert from a Python list? Recreate the array from the first assignment but make it 5 rows by 2 columns.

Once you’ve done that, create an array of random numbers between 0 and 1 in a 3x3 shape. One of our data scientists has been asking about this so I want to make sure it’s possible.

Thanks!

 `numpy_assignments.ipynb`

 Reply

 Forward

Solution Code

```
my_array = np.linspace(10, 100, 10).reshape(5, 2)
```

```
my_array
```

```
array([[ 10.,  20.],
       [ 30.,  40.],
       [ 50.,  60.],
       [ 70.,  80.],
       [ 90., 100.]])
```

```
from numpy.random import default_rng
```

```
rng = default_rng(2022)
```

```
random_array = rng.random(9).reshape(3, 3)
```

```
random_array
```

```
array([[0.24742606, 0.09299006, 0.61176337],
       [0.06066207, 0.66103343, 0.75515778],
       [0.1108689 , 0.04305584, 0.41441747]])
```



INDEXING & SLICING ARRAYS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Indexing & slicing one-dimensional arrays is the same as base Python

- **array[index]** – indexing to access a single element (*0-indexed*)
- **array[start:stop:step size]** – slicing to access a series of elements (*stop is not inclusive*)

```
product_array
```

```
array(['fruits', 'vegetables', 'cereal', 'dairy', 'eggs', 'snacks',  
      'beverages', 'coffee', 'tea', 'spices'], dtype='<U10')
```

```
print(product_array[1])  
print(product_array[-1])
```

```
vegetables  
spices
```

This grabs the **second** and **last** elements of `product_array`

```
product_array[:5]
```

```
array(['fruits', 'vegetables', 'cereal', 'dairy', 'eggs'], dtype='<U10')
```

This grabs the **first five** elements of `product_array`

```
product_array[5::2]
```

```
array(['snacks', 'coffee', 'spices'], dtype='<U10')
```

This starts at the **sixth** element and grabs **every other** element until the end of `product_array`



INDEXING & SLICING ARRAYS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Indexing & slicing two-dimensional arrays requires an extra index or slice

- **array[*row index*, *column index*]** – indexing to access a single element (*0-indexed*)
- **array[*start:stop:step size*, *start:stop:step size*]** – slicing to access a series of elements

```
product_array2D = product_array.reshape(2, 5)
product_array2D
```

```
array([[ 'fruits', 'vegetables', 'cereal', 'dairy', 'eggs'],
       [ 'snacks', 'beverages', 'coffee', 'tea', 'spices']], dtype='<U10')
```

```
product_array2D[1, 2]

'coffee'
```

This goes to the *second* row
and grabs the *third* element

```
product_array2D[:, 2:]

array([[ 'cereal', 'dairy', 'eggs'],
       [ 'coffee', 'tea', 'spices']], dtype='<U10')
```

This goes to *all* rows and grabs
all the elements starting from
the *third* in each row

```
product_array2D[1:, :]

array([[ 'snacks', 'beverages', 'coffee', 'tea', 'spices']], dtype='<U10')
```

This goes to the *second* row
and grabs *all* its elements

ASSIGNMENT: ARRAY ACCESS



NEW MESSAGE

May 17, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **Indexing & Slicing Arrays**

Ok, last 'theoretical' exercise before we start working with real data.

I am familiar with indexing and slicing in base Python but have no idea how it works in multiple dimensions.

I've provided a few different 'cuts' of the data in the notebook – can you slice and dice the random array we created in the last exercise?

Thanks!



numpy_assignments.ipynb

← Reply

→ Forward

Results Preview

```
array([[0.24742606, 0.09299006, 0.61176337],
       [0.06066207, 0.66103343, 0.75515778],
       [0.1108689 , 0.04305584, 0.41441747]])
```

First two 'rows'

```
array([[0.24742606, 0.09299006, 0.61176337],
       [0.06066207, 0.66103343, 0.75515778]])
```

First 'column'

```
array([0.09299006, 0.66103343, 0.04305584])
```

Second number in third 'row':

```
0.04305584439252108
```

SOLUTION: ARRAY ACCESS



NEW MESSAGE

May 17, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **Indexing & Slicing Arrays**

Ok, last 'theoretical' exercise before we start working with real data.

I am familiar with indexing and slicing in base Python but have no idea how it works in multiple dimensions.

I've provided a few different 'cuts' of the data in the notebook – can you slice and dice the random array we created in the last exercise?

Thanks!

 `numpy_assignments.ipynb`

 Reply

 Forward

Solution Code

```
array([[0.24742606, 0.09299006, 0.61176337],
       [0.06066207, 0.66103343, 0.75515778],
       [0.1108689 , 0.04305584, 0.41441747]])
```

First two 'rows'

```
random_array[:2]
```

```
array([[0.24742606, 0.09299006, 0.61176337],
       [0.06066207, 0.66103343, 0.75515778]])
```

First 'column'

```
random_array[:, 1]
```

```
array([0.09299006, 0.66103343, 0.04305584])
```

Second number in third 'row':

```
random_array[2, 1]
```

```
0.04305584439252108
```



ARRAY OPERATIONS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Arithmetic operators can be used to perform **array operations**

```
sales = [[0, 5, 155, 0, 518], [0, 1827, 616, 317, 325]]  
sales_array = np.array(sales)  
sales_array
```

```
array([[ 0,    5, 155,    0, 518],  
       [ 0, 1827, 616, 317, 325]])
```

```
sales_array + 2
```

```
array([[ 2,    7, 157,    2, 520],  
       [ 2, 1829, 618, 319, 327]])
```

```
quantity = sales_array[0, :]  
price = sales_array[1, :]
```

```
quantity * price
```

```
array([    0,   9135,  95480,    0, 168350])
```



Array operations are applied via **vectorization** and **broadcasting**, which eliminates the need to loop through the array's elements

This adds 2 to every element in the array

*This assigns all the elements in the first row to 'quantity'
Then assigns all the elements in the second row to 'price'
Finally, it multiplies the corresponding elements in each array:
(0*0, 5*1827, 155*616, 0*317, and 518*325)*

ASSIGNMENT: ARRAY OPERATIONS



NEW MESSAGE

May 1, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **Random Discounting Arrays**

Ok, so now that we've gotten the basics down, we can start using NumPy for our first tasks. As part of a promotion, we want to apply a random discount to surprise our customers and generate social media buzz.

First, add a flat shipping cost of 5 to our prices to get the 'total' amount owed.

The numbers in the random array represent 'discount percent'. To get the 'percent owed', subtract the first 6 numbers in the random array from 1, then multiply 'percent owed' by 'total' to get the final amount owed.

 numpy_assignments.ipynb

 Reply

 Forward

Results Preview

```
prices = np.array([5.99, 6.99, 22.49, 99.99, 4.99, 49.99])
```

```
total
```

```
array([ 10.99,  11.99,  27.49, 104.99,   9.99,  54.99])
```

```
print(discount_pct)
print(pct_owed)
print(final_owed.round(2))
```

```
[0.24742606 0.09299006 0.61176337 0.06066207 0.66103343 0.75515778]
[0.75257394 0.90700994 0.38823663 0.93933793 0.33896657 0.24484222]
[ 8.27 10.88 10.67 98.62  3.39 13.46]
```

The `.round()` array method rounds the elements of the array to the specified decimals

SOLUTION: ARRAY OPERATIONS



NEW MESSAGE

May 1, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **Random Discounting Arrays**

Ok, so now that we've gotten the basics down, we can start using NumPy for our first tasks. As part of a promotion, we want to apply a random discount to surprise our customers and generate social media buzz.

First, add a flat shipping cost of 5 to our prices to get the 'total' amount owed.

The numbers in the random array represent 'discount percent'. To get the 'percent owed', subtract the first 6 numbers in the random array from 1, then multiply 'percent owed' by 'total' to get the final amount owed.

 `numpy_assignments.ipynb`

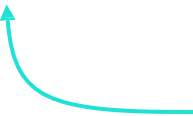
 Reply

 Forward

Solution Code

```
prices = np.array([5.99, 6.99, 22.49, 99.99, 4.99, 49.99])
total = prices + 5
total
array([ 10.99,  11.99,  27.49, 104.99,   9.99,  54.99])
```

```
discount_pct = random_array[:2].reshape(6)
pct_owed = 1 - discount_pct
final_owed = total * pct_owed
final_owed.round(2)
array([ 8.27, 10.88, 10.67, 98.62,  3.39, 13.46])
```

 The `.round()` array method rounds the elements of the array to the specified decimals



FILTERING ARRAYS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

You can **filter arrays** by indexing them with a logical test

- Only the array elements in positions where the logical test returns True are returned

```
sales_array
```

```
array([[ 0,   5, 155,   0, 518],  
       [ 0, 1827, 616, 317, 325]])
```

```
sales_array != 0
```

```
array([[False,  True,  True, False,  True],  
       [False,  True,  True,  True,  True]])
```

Performing a logical test on a NumPy array returns a **Boolean array** with the results of the logical test on each array element

```
sales_array[sales_array != 0]
```

```
array([  5, 155, 518, 1827, 616, 317, 325])
```

Indexing an array with a Boolean array returns an array with the elements where the Boolean value is **True**



FILTERING ARRAYS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

You can filter arrays with **multiple logical tests**

- Use `|` for **or** conditions and `&` for **and** conditions

```
sales_array
```

```
array([[ 0,   5, 155,   0, 518],  
       [ 0, 1827, 616, 317, 325]])
```

```
sales_array[(sales_array == 616) | (sales_array < 100)]
```

```
array([ 0,   5,   0,   0, 616])
```

This returns an array with elements equal to 616 or less than 100

```
sales_array[(sales_array > 100) & (sales_array < 500)]
```

```
array([155, 317, 325])
```

This returns an array with elements greater than 100 and less than 500

```
mask = (sales_array > 100) & (sales_array < 500)
```

```
sales_array[mask]
```



PRO TIP: Store complex filtering criteria in a variable (known as a Boolean mask)



FILTERING ARRAYS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

You can filter arrays based on **values in other arrays**

- Use the Boolean array returned from the other array to index the array you want to filter

```
sales_array
```

```
array([ 0,  5, 155,  0, 518])
```

```
product_array
```

```
array(['fruits', 'vegetables', 'cereal', 'dairy', 'eggs'], dtype='<U10')
```

```
product_array[sales_array > 0]
```

```
array(['vegetables', 'cereal', 'eggs'], dtype='<U10')
```

*This returns the elements from product_array
where values in sales_array are greater than 0*



MODIFYING ARRAY VALUES

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

You can **modify array values** by assigning new ones

```
sales_array  
array([ 0,  5, 155,  0, 518])
```

```
sales_array[1] = 25
```

```
sales_array  
array([ 0, 25, 155,  0, 518])
```

*This assigns a single value via **indexing***

```
sales_array[sales_array == 0] = 5
```

```
sales_array  
array([ 5, 25, 155,  5, 518])
```

*This **filters** the zero values in sales_array and assigns them a new value of 5*



THE WHERE FUNCTION

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

The **where()** NumPy function performs a logical test and returns a given value if the test is True, or another if the test is False

```
np.where(logical test,  
         value if True,  
         value if False)
```

*A logical expression that
evaluates to True or False*

*Value to return when
the expression is True*

*Calls the NumPy
function library*

*Value to return when
the expression is False*



THE WHERE FUNCTION

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

The **where()** NumPy function performs a logical test and returns a given value if the test is True, or another if the test is False

```
inventory_array
```

```
array([ 12, 102,  18,   0,   0])
```

```
product_array
```

```
array(['fruits', 'vegetables', 'cereal', 'dairy', 'eggs'], dtype='<U10')
```

```
np.where(inventory_array <= 0, "Out of Stock", "In Stock")
```

```
array(['In Stock', 'In Stock', 'In Stock', 'Out of Stock', 'Out of Stock'],  
      dtype='<U12')
```

If inventory is zero or negative,
assign 'Out of Stock', otherwise
assign 'In Stock'

```
np.where(inventory_array <= 0, "Out of Stock", product_array)
```

```
array(['fruits', 'vegetables', 'cereal', 'Out of Stock', 'Out of Stock'],  
      dtype='<U12')
```

If inventory is zero or negative,
assign 'Out of Stock', otherwise
assign the product_array value

ASSIGNMENT: FILTERING ARRAYS



NEW MESSAGE

May 1, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **Subsetting and Filtering Arrays**

Hey there,

We're working on some more promotions. Can you filter our product list to only include prices greater than 25?

Once you've done that, modify your logic to force cola into the list. Call this array 'fancy_feast_special'.

Finally, we need to modify our shipping logic. Create a new shipping cost array, but this time if price is greater than 20, shipping cost is 0, otherwise shipping cost is 5.

Thanks!

 numpy_assignments.ipynb

 Reply

 Forward

Results Preview

```
products[ ]
```

```
array(['rare tomato', 'gourmet ice cream'], dtype='<U17')
```

```
fancy_feast_special
```

```
array(['rare tomato', 'cola', 'gourmet ice cream'], dtype='<U17')
```

```
shipping_cost
```

```
array([5, 5, 0, 0, 5, 0])
```

SOLUTION: FILTERING ARRAYS



NEW MESSAGE

May 1, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **Subsetting and Filtering Arrays**

Hey there,

We're working on some more promotions. Can you filter our product list to only include prices greater than 25?

Once you've done that, modify your logic to force cola into the list. Call this array 'fancy_feast_special'.

Finally, we need to modify our shipping logic. Create a new shipping cost array, but this time if price is greater than 20, shipping cost is 0, otherwise shipping cost is 5.

Thanks!



numpy_assignments.ipynb

← Reply

→ Forward

Solution Code

```
products[prices > 25]
array(['rare tomato', 'gourmet ice cream'], dtype='<U17')
```

```
mask = (prices > 25) | (products == "cola")
fancy_feast_special = products[mask]
fancy_feast_special
array(['rare tomato', 'cola', 'gourmet ice cream'], dtype='<U17')
```

```
shipping_cost = np.where(prices > 20, 0, 5)
shipping_cost
array([5, 5, 0, 0, 5, 0])
```



ARRAY AGGREGATION METHODS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Array aggregation methods let you calculate metrics like sum, mean, and max

```
sales_array
```

```
array([[ 0,    5, 155,    0, 518],  
       [ 0, 1827, 616, 317, 325]])
```

array.sum() *Returns the sum of all values in an array*

```
sales_array.sum()
```

3763

array.mean() *Returns the average of the values in an array*

```
sales_array.mean()
```

376.3

array.max() *Returns the largest value in an array*

```
sales_array.max()
```

1827

array.min() *Returns the smallest value in an array*

```
sales_array.min()
```

0



ARRAY AGGREGATION METHODS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

You can also aggregate across **rows** or **columns**

```
sales_array
```

```
array([[ 0,    5, 155,    0, 518],  
       [ 0, 1827, 616,  317, 325]])
```

array.sum() *Returns the sum of all values in an array*

```
sales_array.sum()
```

```
3763
```

array.sum(axis=0) *Aggregates across rows*

```
sales_array.sum(axis=0)
```

```
array([ 0, 1832, 771,  317, 84
```

array.sum(axis=1) *Aggregates across columns*

```
sales_array.sum(axis=1)
```

```
array([ 678, 3085])
```




ARRAY FUNCTIONS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Array functions let you perform other aggregations like median and percentiles

```
sales_array
```

```
array([[ 0,    5,  155,    0,  518],  
       [ 0, 1827,  616,  317,  325]])
```

np.median(array) *Returns the median value in an array*

```
np.median(sales_array)
```

```
236.0
```

np.percentile(array, n) *Returns a value in the n^{th} percentile in an array*

```
np.percentile(sales_array, 90)
```

```
737.0999999999996
```



This uses linear interpolation by default



ARRAY FUNCTIONS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

You can also return a **unique** list of values or the **square root** for each number

```
sales_array
```

```
array([[ 0,    5, 155,    0, 518],  
       [ 0, 1827, 616, 317, 325]])
```

np.unique(array) *Returns the unique values in an array*

```
np.unique(sales_array)
```

```
array([ 0,    5, 155, 317, 325, 518, 616, 1827])
```

np.sqrt(array) *Returns the square root of each value in an array*

```
np.sqrt(sales_array)
```

```
array([[ 0.          ,  2.23606798, 12.4498996 ,  0.          , 22.75961335],  
       [ 0.          , 42.74342055, 24.81934729, 17.80449381, 18.02775638]])
```



SORTING ARRAYS

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

The `sort()` method will **sort arrays** in place

- Use the `axis` argument to specify the dimension to sort by

```
sales_array
```

```
array([[ 0,    5, 155,    0, 518],  
       [ 0, 1827, 616, 317, 325]])
```

```
sales_array.sort()
```

```
sales_array
```

```
array([[ 0,    0,    5, 155, 518],  
       [ 0, 317, 325, 616, 1827]])
```

*axis=1 by default, which sorts a
two-dimensional array row by row*

```
sales_array.sort(axis=0)
```

```
sales_array
```

```
array([[ 0,    5, 155,    0, 325],  
       [ 0, 1827, 616, 317, 518]])
```

axis=0 will sort by columns

ASSIGNMENT: SORTING AND AGGREGATING



NEW MESSAGE

May 1, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **Top 3 Price Stats**

Hey there,

Thanks for all your hard work. I know we're working with small sample sizes, but we're proving that analysis can be done in Python!

Can you calculate the mean, min, max, and median of our 3 most expensive product prices? Sorting the array first should help!

Then, calculate the number of unique price tiers we have.

Thanks!

 `numpy_assignments.ipynb`

 Reply

 Forward

Results Preview

Top 3 Price Stats

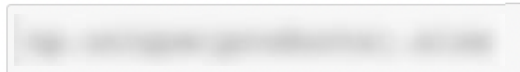
Mean: 40.98667062008267

Min: 10.875049160510919

Max: 98.62108889965567

Median: 13.46387380008141

Unique Price Tiers



3

SOLUTION: SORTING AND AGGREGATING



NEW MESSAGE

May 1, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **Top 3 Expensive Products**

Hey there,

Thanks for all your hard work. I know we're working with small sample sizes, but we're proving that analysis can be done in Python!

Can you calculate the mean, min, max, and median of our 3 most expensive product prices? Sorting the array first should help!

Then, calculate the number of unique price tiers we have.

Thanks!

 `numpy_assignments.ipynb`

 Reply

 Forward

Solution Code

Top 3 Price Stats

Mean: 40.98667062008267
Min: 10.875049160510919
Max: 98.62108889965567
Median: 13.46387380008141

Unique Price Tiers

```
np.unique(products).size
```

3



VECTORIZATION

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Vectorization is the process of pushing array operations into optimized C code, which is easier and more efficient than writing for loops

```
def for_loop_multiply_lists(list1, list2):  
    product_list = []  
    for element1, element2 in zip(tuple1, tuple2):  
        product_list.append(element1 * element2)  
    return product_list
```

Function that multiplies two Python lists

```
def multiply_arrays(array1, array2):  
    return array1 * array2
```

Function that multiplies two NumPy arrays

```
list1 = list(range(1000))  
list2 = list(range(1000))
```

```
%%timeit -r 5 -n 10000  
for_loop_multiply_lists(list1, list2)
```

Generating and multiplying two lists

75.8 μ s \pm 2 μ s per loop (mean \pm std. dev. of 5 runs, 10000 loops each)

```
array1 = np.array(list1)  
array2 = np.array(list2)
```

```
%%timeit -r 5 -n 10000  
multiply_arrays(array1, array2)
```

Converting and multiplying two arrays

~86 times faster!

876 ns \pm 44.7 ns per loop (mean \pm std. dev. of 5 runs, 10000 loops each)



VECTORIZATION

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Vectorization is the process of pushing array operations into optimized C code, which is easier and more efficient than writing for loops

```
def for_loop_multiply_lists(list1, list2):  
    product_list = []  
    for element1, element2 in zip(tuple1, tuple2):  
        product_list.append(element1 * element2)  
    return product_list
```

```
def multiply_arrays(array1, array2):  
    return array1 * array2
```

```
list1 = list(range(1000))  
list2 = list(range(1000))
```

```
%%timeit -r 5 -n 10000  
for_loop_multiply_lists(list1, list2)
```

75.8 μ s \pm 2 μ s per loop (mean \pm std. dev. of 5 runs, 10000 loops each)

```
array1 = np.array(list1)  
array2 = np.array(list2)
```

```
%%timeit -r 5 -n 10000  
multiply_arrays(array1, array2)
```

876 ns \pm 44.7 ns per loop (mean \pm std. dev. of 5 runs, 10000 loops each)



PRO TIP: Use vectorized operations whenever possible when manipulating data, and avoid writing loops

Converting and multiplying two arrays
~86 times faster!



BROADCASTING

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Broadcasting lets you perform vectorized operations with arrays of different sizes, where NumPy will expand the smaller array to 'fit' the larger one

- Single values (scalars) can be broadcast into arrays of any dimension

```
test_array = np.array([[1, 2, 3], [1, 2, 3], [1, 2, 3]])
```

```
test_array
```

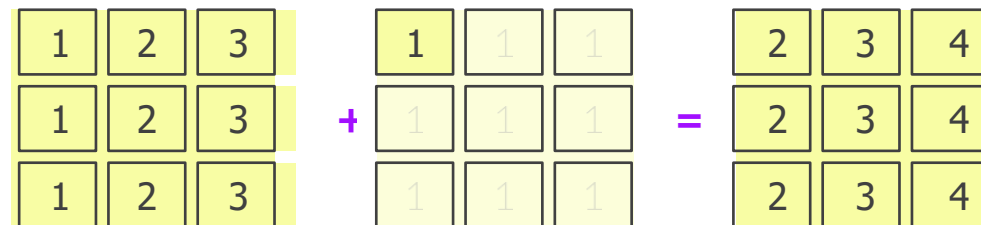
```
array([[1, 2, 3],  
       [1, 2, 3],  
       [1, 2, 3]])
```

```
test_array + 1
```

```
array([[2, 3, 4],  
       [2, 3, 4],  
       [2, 3, 4]])
```



How does this code work?





BROADCASTING

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Broadcasting lets you perform vectorized operations with arrays of different sizes, where NumPy will expand the smaller array to 'fit' the larger one

- Single values (scalars) can be broadcast into arrays of any dimension
- Dimensions with a length greater than one must be the same size

```
test_array = np.array([[1, 2, 3], [1, 2, 3], [1, 2, 3]])
```

```
test_array
```

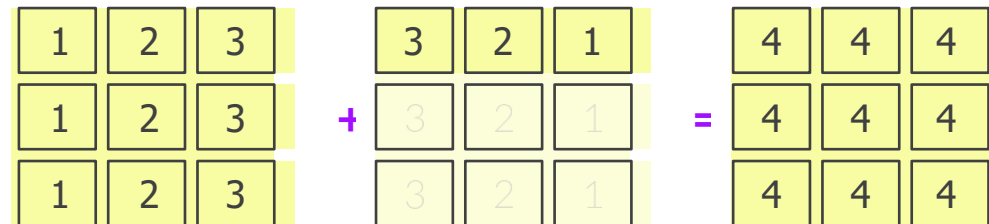
```
array([[1, 2, 3],  
       [1, 2, 3],  
       [1, 2, 3]])
```

```
test_array + np.array([3, 2, 1])
```

```
array([[4, 4, 4],  
       [4, 4, 4],  
       [4, 4, 4]])
```



How does this code work?





BROADCASTING

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Broadcasting lets you perform vectorized operations with arrays of different sizes, where NumPy will expand the smaller array to 'fit' the larger one

- Single values (scalars) can be broadcast into arrays of any dimension
- Dimensions with a length greater than one must be the same size

```
test_array = np.array([[1, 2, 3], [1, 2, 3], [1, 2, 3]])
```

```
test_array
```

```
array([[1, 2, 3],  
       [1, 2, 3],  
       [1, 2, 3]])
```

```
test_array + np.array([3, 2, 1]).reshape(3, 1)
```

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



How does this code work?





BROADCASTING

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Broadcasting lets you perform vectorized operations with arrays of different sizes, where NumPy will expand the smaller array to 'fit' the larger one

- Single values (scalars) can be broadcast into arrays of any dimension
- Dimensions with a length greater than one must be the same size

```
test_array = np.array([[1, 2, 3], [1, 2, 3], [1, 2, 3]])
```

```
test_array
```

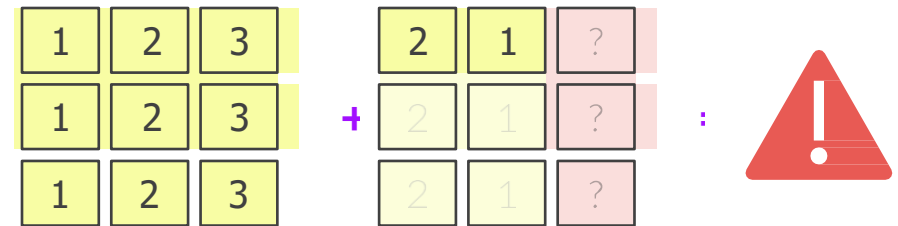
```
array([[1, 2, 3],  
       [1, 2, 3],  
       [1, 2, 3]])
```

```
test_array + np.array([2, 1])
```

ValueError: operands could not be broadcast together with shapes (3,3) (2,)



How does this code work?





BROADCASTING

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Broadcasting lets you perform vectorized operations with arrays of different sizes, where NumPy will expand the smaller array to 'fit' the larger one

- Single values (scalars) can be broadcast into arrays of any dimension
- Dimensions with a length greater than one must be the same size

```
test_array = np.array([[1, 2, 3], [1, 2, 3], [1, 2, 3]])
```

```
test_array
```

```
array([[1, 2, 3],  
       [1, 2, 3],  
       [1, 2, 3]])
```

```
test_array[0, :] + test_array[:, 1].reshape(3, 1)
```

```
array([[3, 4, 5],  
       [3, 4, 5],  
       [3, 4, 5]])
```



How does this code work?





BROADCASTING

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Broadcasting lets you perform vectorized operations with arrays of different sizes, where NumPy will expand the smaller array to 'fit' the larger one

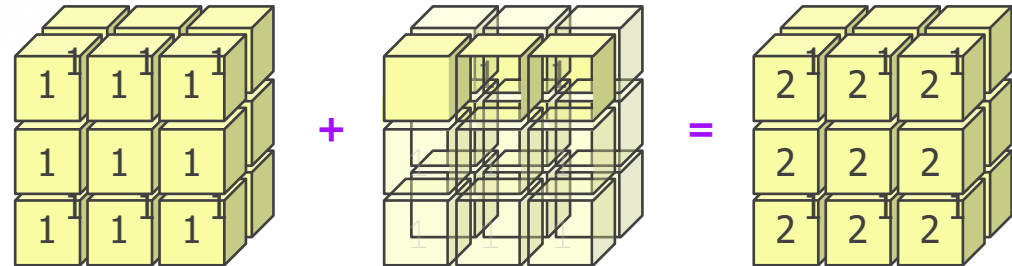
- Single values (scalars) can be broadcast into arrays of any dimension
- Dimensions with a length greater than one must be the same size

```
np.ones((2, 3, 3), dtype=int) + np.ones(3, dtype=int)
```

```
array([[[2, 2, 2],  
        [2, 2, 2],  
        [2, 2, 2]],  
       [[2, 2, 2],  
        [2, 2, 2],  
        [2, 2, 2]]])
```



How does this code work?





BROADCASTING

Intro to Pandas
& NumPy

NumPy Array
Basics

Array Creation

Array Indexing
& Slicing

Array Operations

Vectorization &
Broadcasting

Broadcasting lets you perform vectorized operations with arrays of different sizes, where NumPy will expand the smaller array to 'fit' the larger one

- Single values (scalars) can be broadcast into arrays of any dimension
- Dimensions with a length greater than one must be the same size



Compatible shapes:

Array 1	Array 2	Output
(3, 3)	(100, 3, 3)	(100, 3, 3)
(3, 1, 5, 1)	(4, 1, 6)	(3, 4, 5, 6)



Incompatible shapes:

Array 1	Array 2
(4 , 3)	(100, 3 , 3)
(3, 1, 5 , 1)	(4, 2 , 6)

ASSIGNMENT: BRINGING IT ALL TOGETHER



NEW MESSAGE

May 12, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **Bringing Pd Chaining Arrays**

Alright, our new data scientist set up a little test case for us. She provided code to read in data from a csv and convert two columns to arrays.

Filter 'sales_array' to only include sales that had the product family 'PRODUCE' in the 'family_array'. Call this produce_sales.

Then randomly sample half of the remaining sales and calculate the mean of those sales.

Thanks!

 numpy_assignments.ipynb

 Reply

 Forward

Results Preview

```
import pandas as pd

retail_df = pd.read_csv("../retail/retail.csv").sample(1000, random_state=100)

family_array = np.array(retail_df["family"])
sales_array = np.array(retail_df["sales"])
```

1556.4381428571428

SOLUTION: BRINGING IT ALL TOGETHER



NEW MESSAGE

May 12, 2022

From: **Ross Retail** (Head of Analytics)

Subject: **Bringing Pd Chaining Arrays**

Alright, our new data scientist set up a little test case for us. She provided code to read in data from a csv and convert two columns to arrays.

Filter 'sales_array' to only include sales that had the product family 'PRODUCE' in the 'family_array'. Call this produce_sales.

Then randomly sample half of the remaining sales and calculate the mean of those sales.

Thanks!

 numpy_assignments.ipynb

 Reply

 Forward

Solution Code

```
import pandas as pd

retail_df = pd.read_csv("../retail/retail.csv").sample(1000, random_state=100)

family_array = np.array(retail_df["family"])
sales_array = np.array(retail_df["sales"])

produce_sales = sales_array[family_array == "PRODUCE"]

produce_sales[rng.random(30) < 0.5].mean()

1556.4381428571428
```


KEY TAKEAWAYS



NumPy forms the **foundation for Pandas**

- *As an analyst, it's important to be comfortable working with NumPy arrays & functions in order to use Pandas data structures & functions properly*



NumPy arrays are **more efficient** than base Python lists and tuples

- *They are semi-mutable data structures capable of storing many data types*
- *Their values can be modified, but their size cannot*



Array operations let you **aggregate, filter, and sort** data

- *Broadcasting and vectorization make these operations convenient and efficient without the use of loops*
- *The syntax for NumPy array operations is very similar to Pandas*