Python Programming

Jian Zhang

https://jianzhang.tech/

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy stands for Numerical Python.

!pip install numpy

What is Vectorization?

```
# Converting iterative statements into a vector based operation is called vectorization.
# It is faster as modern CPUs are optimized for such operations.

%%time
a = [i for i in range(100000)]
b = [i for i in range(100000)]
%%time
c = []
for i in range(len(a)):
    c.append(a[i] + 2 * b[i])

%%time
a = np.arange(100000)
b = np.arange(100000)
%%time
c = a + 2 * b
```

NumPy as np

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))

print(np.__version__)
```

Dimensions in Arrays

```
import numpy as np

arr = np.array(42)
print(arr)
print(type(arr))

arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))

arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
print(type(arr))

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
```

Dimensions in Arrays

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Access Array Elements

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0])

print(arr[1])

print(arr[2] + arr[3])
```

Access Array Elements

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])

print('5th element on 2nd row: ', arr[1, 4])

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(arr[0, 1, 2])
```

Negative Indexing

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('Last element from 2nd dim: ', arr[1, -1])
```

Slicing Arrays

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])

print(arr[4:])

print(arr[:4])

print(arr[-3:-1])
```

STEP

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5:2])


print(arr[::2])
```

Slicing 2-D Arrays

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])

print(arr[0:2, 2])

print(arr[0:2, 1:4])
```

Checking Data Type of an Array

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr.dtype)

arr = np.array([1.2, 2, 3, 4])

print(arr.dtype)

arr = np.array(['apple', 'banana', 'cherry'])

print(arr.dtype)
```

Converting Data Type on Existing Arrays

```
import numpy as np

arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype('i')
print(newarr)
print(newarr.dtype)

newarr = arr.astype(int)
print(newarr)
print(newarr.dtype)

arr = np.array([1, 0, 3])
newarr = arr.astype(bool)
print(newarr)
print(newarr.dtype)
```

NumPy Array Copy vs View

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42
print(arr)
print(x)

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42
print(arr)

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
x[0] = 31
print(arr)
print(x)
```

Check if Array Owns it's Data

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

x = arr.copy()
y = arr.view()

print(x.base)
print(y.base)
```

Shape of an Array

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(arr.shape)

row, col = arr.shape
print(row)
print(col)
```

Reshaping arrays

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)

newarr = arr.reshape(2, 3, 2)
print(newarr)

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
newarr = arr.reshape(3, 3)
print(newarr)
```

Reshaping arrays

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
print(arr.reshape(2, 4).base)

# Unknown Dimension
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
newarr = arr.reshape(2, 2, -1)
print(newarr)

# Flattening the arrays
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
newarr = arr.reshape(-1)
print(newarr)
```

Iterating Arrays

```
import numpy as np

arr = np.array([1, 2, 3])
for x in arr:
    print(x)

arr = np.array([[1, 2, 3], [4, 5, 6]])
for x in arr:
    print(x)

arr = np.array([[1, 2, 3], [4, 5, 6]])
for x in arr:
    for y in x:
        print(y)
```

Joining NumPy Arrays

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2))
print(arr)

arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=1)
print(arr)
```

Searching Arrays

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 0)
print(x)

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 1)
print(x)
```

Filtering Arrays

```
import numpy as np

arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]

print(newarr)
```

Filtering Arrays

```
import numpy as np
arr = np.array([41, 42, 43, 44])

# Create an empty list
filter_arr = []

# go through each element in arr
for element in arr:
  # if the element is higher than 42, set the value to True, otherwise False:
    if element > 42:
        filter_arr.append(True)
    else:
        filter_arr.append(False)

newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
```

Generate Random Number

```
from numpy import random

x = random.randint(100)
print(x)

x = random.rand()
print(x)

x=random.randint(100, size=(5))
print(x)



x = random.randint(100, size=(3, 5))
print(x)
```

Generate Random Number From Array

```
from numpy import random

x = random.choice([3, 5, 7, 9])

print(x)


from numpy import random

x = random.choice([3, 5, 7, 9], size=(3, 5))

print(x)
```

Random Permutations

```
from numpy import random
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

random.shuffle(arr)
print(arr)



from numpy import random
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(random.permutation(arr))
```

Some Initializations

```
import numpy as np

a = np.zeros((2, 2))
print(a)

b = np.full((2, 2), 7)
print(b)

c = np.eye(3)
print(c)

d = np.random.random((2, 2))
print(d)

a = np.ones((2, 2))
print(a)
```

Some Initializations

```
nums = np.arange(8)
print(nums)
print(nums.min())
print(np.min(nums))

np.linspace(-1, 1, 5)

np.linspace(-1, 1, 10)
```

Array Operations/Math

```
x = np.array([[1, 2],
              [3, 4]], dtype='f')
y = np.array([[5, 6],
              [7, 8]], dtype='f')

print(x + y)
print(np.add(x, y))

print(x - y)
print(np.subtract(x, y))

print(x * y)
print(np.multiply(x, y))

print(x / y)
print(np.divide(x, y))
```

Array Operations/Math

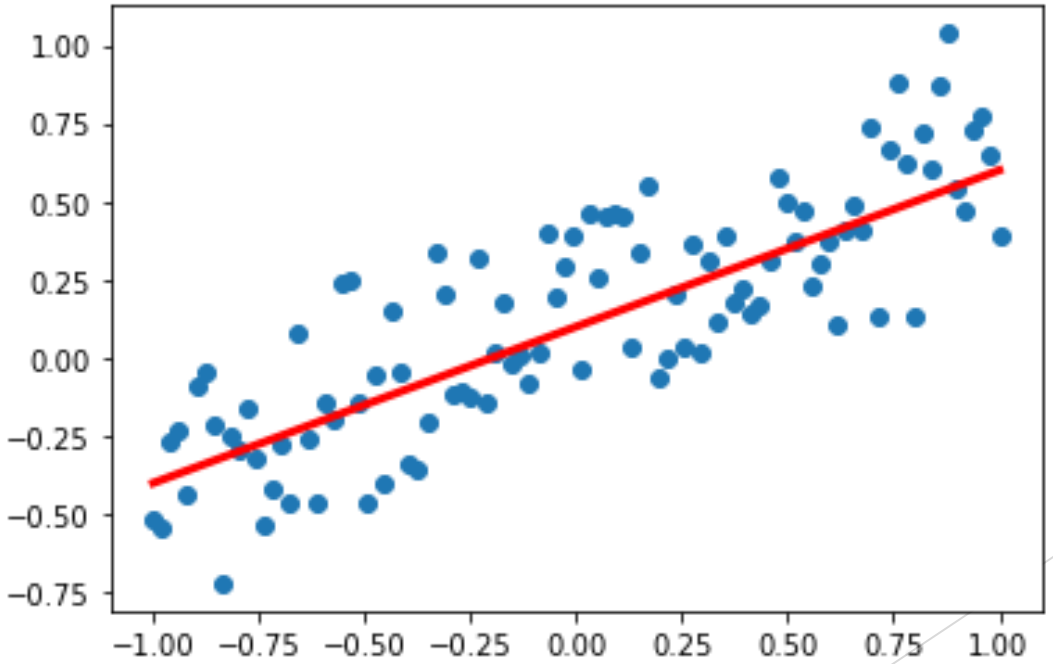
```
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])

v = np.array([9, 10])
w = np.array([11, 12])

print(v.dot(w))
print(np.dot(v, w))

print(x.dot(v))
print(np.dot(x, v))

print(x.dot(y))
print(np.dot(x, y))
```

Array Operations/Math

```
x = np.array([[1, 2, 3],
              [4, 5, 6]])

print(np.sum(x))

print(np.sum(x, axis=0))

print(np.sum(x, axis=1))

print(np.max(x, axis=1))

x = np.array([[1, 2, 3],
              [3, 5, 7]])
print(x * 2)
```

Regression Problem

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1, 1, 100)
y = 0.5*x + 0.1 + + 0.2*np.random.randn(100)
```



Regression Problem

```
learning_rate = 0.01
initial_b = 0
initial_w = 0
num_iterations = 101

b = initial_b
w = initial_w

for i in range(num_iterations):
    b, w = step_gradient(b, w, points, learning_rate)

print(b, w)
```

Regression Problem

```
def step_gradient(b_current, w_current, points, learningRate):
    b_gradient = 0
    w_gradient = 0
    N = float(len(points))
    for i in range(0, len(points)):
        x = points[i, 0]
        y = points[i, 1]
        b_gradient += -(2/N) * (y - ((w_current * x) + b_current))
        w_gradient += -(2/N) * x * (y - ((w_current * x) + b_current))
    new_b = b_current - (learningRate * b_gradient)
    new_w = w_current - (learningRate * w_gradient)
    return [new_b, new_w]
```

Regression Problem

```
def step_gradient_fast(b_current, w_current, points, learningRate):
    N = float(len(points))
    b_gradient = np.sum(-(2/N)*(points[:,1] - ((w_current * x) + b_current)))
    w_gradient = np.sum(-(2/N)*x * (y - ((w_current * x) + b_current)))

    new_b = b_current - (learningRate * b_gradient)
    new_w = w_current - (learningRate * w_gradient)
    return [new_b, new_w]
```

Regression Problem

```
learning_rate = 0.01
b = 0
w = 0
num_iterations = 1001
points = np.zeros((100, 2))
points[:,0] = x
points[:,1] = y

for i in range(num_iterations):
    if i % 100 == 0:
        print('iteration %d, error is %.4f' % (i, np.mean((y - (w * x + b)) ** 2)))
        plt.scatter(x, y)
        plt.plot(x, 0.5*x + 0.1 , 'r-', lw=2)
        plt.show()
    b, w = step_gradient_fast(b, w, points, learning_rate)

print("b = %.4f, w = %.4f" % (b, w))
```

Questions?