12-Numpy和Pandas的介绍和使用 🐿️

Numpy

In [1]:
```
# 安装numpy模块
%pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\administrator\miniconda3\envs\sam\lib\site-pac
kages (1.24.3)
Note: you may need to restart the kernel to use updated packages.
```

为什么需要Numpy❓

- 运行速度
- 方便使用

In [9]:
```
# 导入numpy
import numpy as np
np.random.seed(12345)
```

In [10]:
```
# 长度100万的numpy格式的数组
my_arr = np.arange(1_000_000)

# 长度100万的list
my_list = list(range(1_000_000))
```

计算数组中一百万个数的平方，使用 timeit 测试两种方法的运行时间

In [11]:
```
print('numpy running time:')
%timeit my_arr2 = my_arr ** 2

print('\npython list running time:')
%timeit my_list2 = [x ** 2 for x in my_list]
```

```
numpy running time:
895 µs ± 32.1 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)

python list running time:
272 ms ± 13.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

上面的运算都是使用CPU进行计算，调用pytorch，使用gpu进行同样的计算

In [12]:
```
# assuming there is a CUDA-compatible GPU available
import torch
my_arr = torch.arange(1_000_000).cuda()
```

In [13]:
```
%timeit my_arr2 = my_arr ** 2
```

62.8 µs ± 402 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)

Numpy ndarray**数据结构**

numpy只有一种数据结构：ndarray，它是一个多维数组，每个元素都是相同类型的。

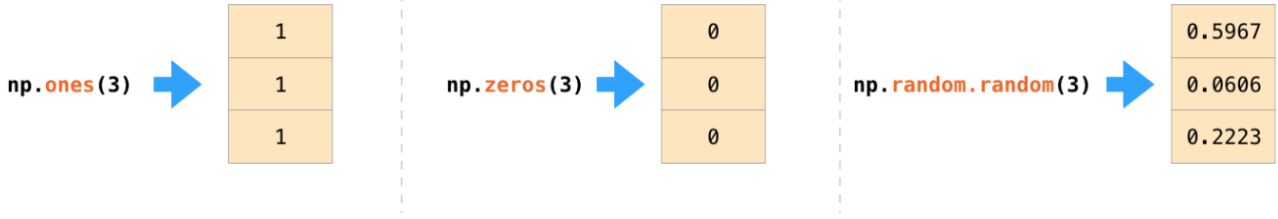
In [7]:
```
import numpy as np

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

Out[7]:
```
array([1, 2, 3])
```

In [8]:
```
# ndarray的数据类型,数组只能包含相同类型的数据
data.dtype
```

Out[8]:
```
dtype('int32')
```

创建ndarray



In [9]:
```
np.ones(3)
```

Out[9]:
```
array([1., 1., 1.])
```

In [10]:
```
np.zeros(3)
```

Out[10]:
```
array([0., 0., 0.])
```

In [11]:
```
np.random.random(3)
```

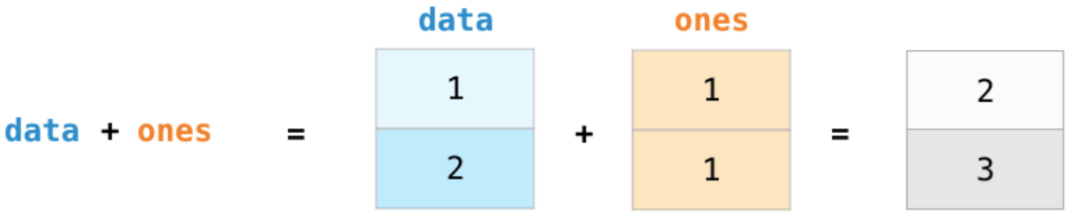
Out[11]:
```
array([0.92961609, 0.31637555, 0.18391881])
```

In [12]:
```
# 类似Python中的range函数
np.arange(3)
```

Out[12]:
```
array([0, 1, 2])
```

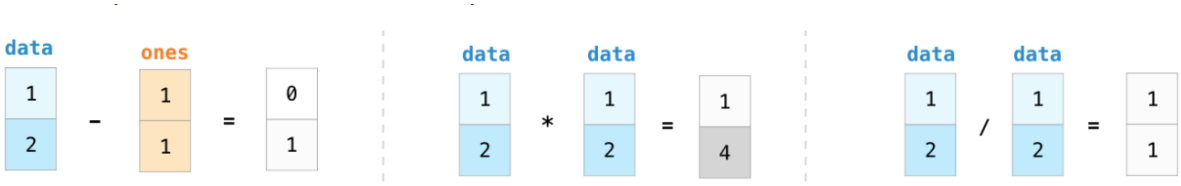
算术运算

Adding them up position-wise (i.e. adding the values of each row) is as simple as typing `data + ones`:



In [13]:
```
data = np.array([1, 2])
ones = np.array([1, 1])
data + ones
```

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



In [14]:
```
data - ones
```

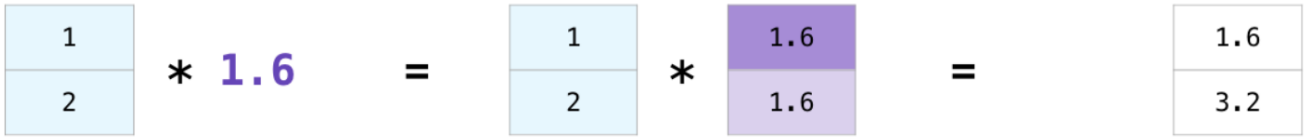
Out[14]:
```
array([0, 1])
```

In [15]:
```
data * data
```

Out[15]:
```
array([1, 4])
```

In [16]:
```
data / data
```

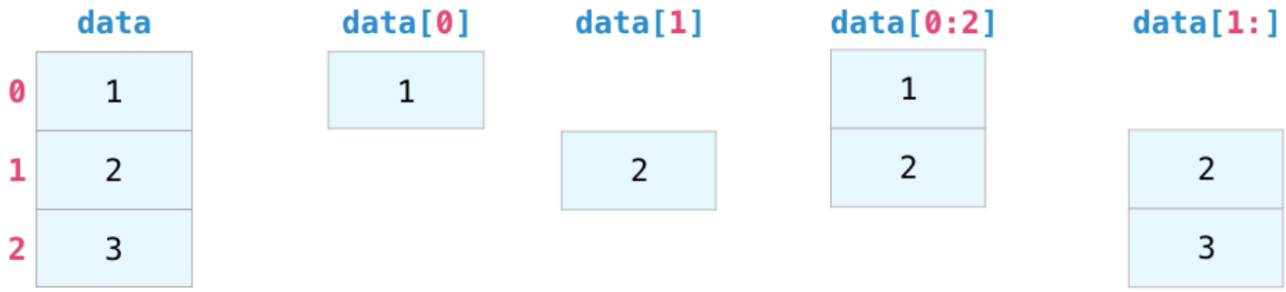
Out[16]:
```
array([1., 1.])
```



In [17]:
```
data * 1.6
```

Out[17]:
```
array([1.6, 3.2])
```

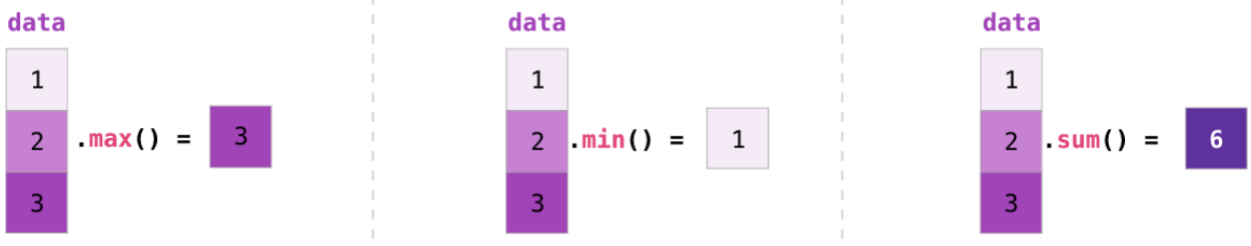
索引和切片



In [18]:
```
data = np.array([1, 2, 3])
print(data[0])
print(data[-1])
print(data[:2])
print(data[1:])
```

```
1
3
[1 2]
[2 3]
```

一维数据的聚合



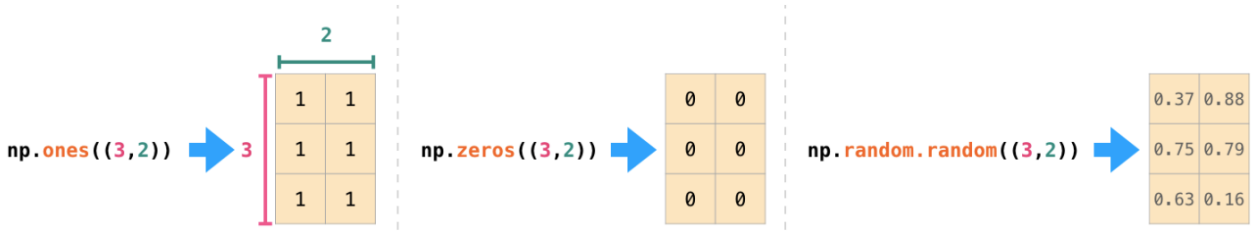
In [19]:
```
# 求最大值、最小值、求和
print(data.max())
print(data.min())
print(data.sum())

# 求均值、标准差
print(data.mean())
print(data.std())

# 求最大值、最小值的索引
print(data.argmax())
print(data.argmin())
```

```
3
1
6
2.0
0.816496580927726
2
0
```

二维矩阵

In [20]:
```
data = np.array([[1, 2], [3, 4]])
data
```

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



In [21]:
```
print(np.ones((3,2)))
```

```
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```

In [22]:
```
print(np.zeros((3,2)))
```

```
[[0. 0.]
 [0. 0.]
 [0. 0.]]
```

In [23]:
```
print(np.random.random((3,2)))
```

```
[[0.20456028 0.56772503]
 [0.5955447  0.96451452]
 [0.6531771  0.74890664]]
```

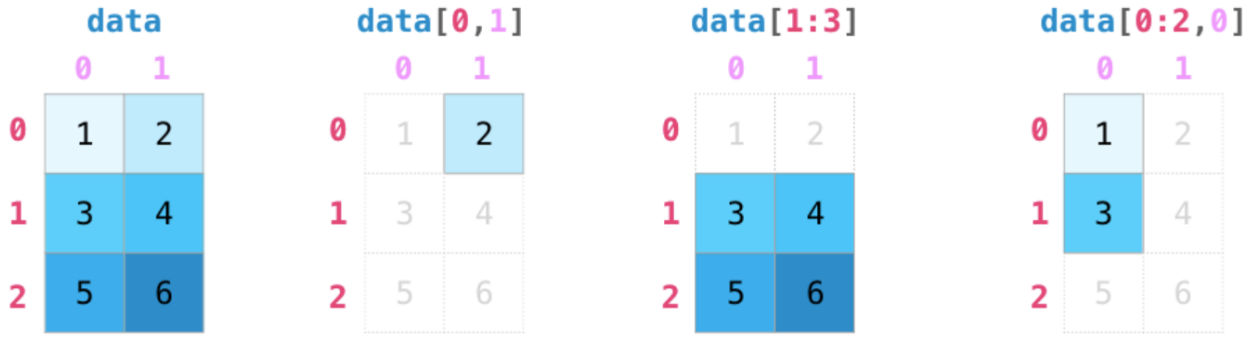
In [24]:
```
np.diag([1, 2, 3])
```

Out[24]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

In [25]:
```
np.eye(3)
```

Out[25]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

二维矩阵的索引和切片



In [26]:
```
data = np.array([[1, 2], [3, 4], [5, 6]])
# 使用行索引和列索引选择到一个元素
print(data[0, 1])

# 使用行索引的切片
print(data[1:3])

# 行索引和列锁引同时使用切片
print(data[0:2, 0])
```

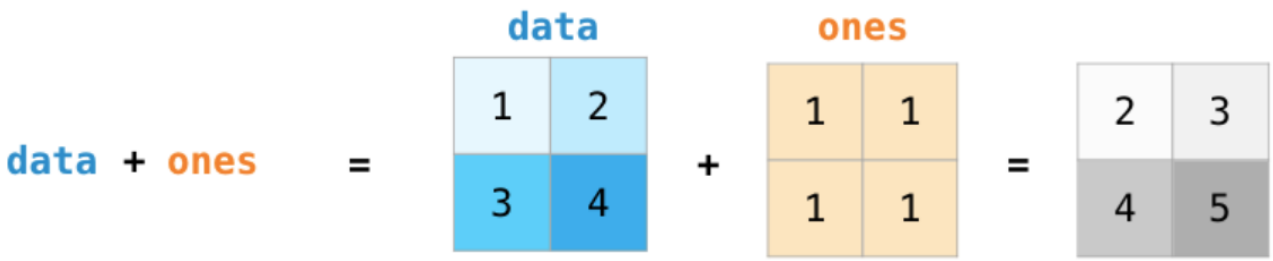
```
2
[[3 4]
 [5 6]]
[1 3]
```

如何选择到所有的奇数行？奇数列？🧐

In [27]:
```
data = np.arange(1,26).reshape(5, 5)
print(data)
```

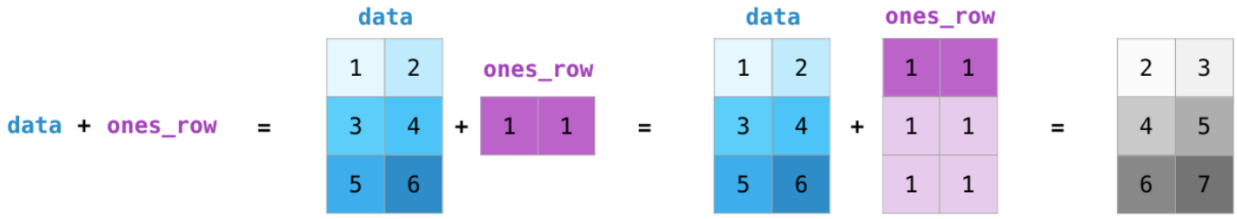
```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]]
```

矩阵运算🧮



In [28]:
```
data = np.array([[1, 2], [3, 4]])
ones = np.ones([2, 2])
data + ones
```

Out[28]:
```
array([[2., 3.],
       [4., 5.]])
```



In [29]:
```
# 按行相加
ones_row = np.ones([1, 1])
data + ones_row
```

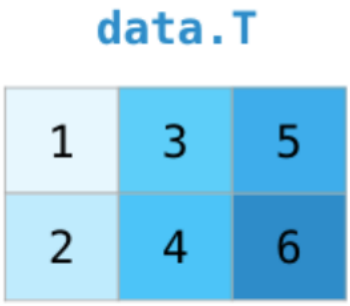
Out[29]:
```
array([[2., 3.],
       [4., 5.]])
```

In [30]:
```
# 按列相加
column_data = np.array([10, 20])

# 转换为列向量
print(column_data[:, np.newaxis])

data + column_data[:, np.newaxis]
```

```
[[10]
 [20]]
```

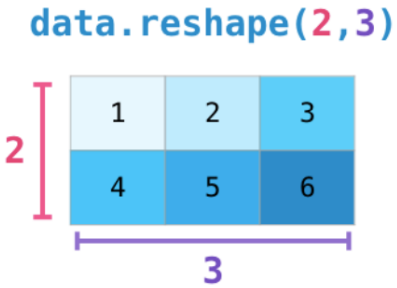
Out[30]:
```
array([[11, 12],
       [23, 24]])
```

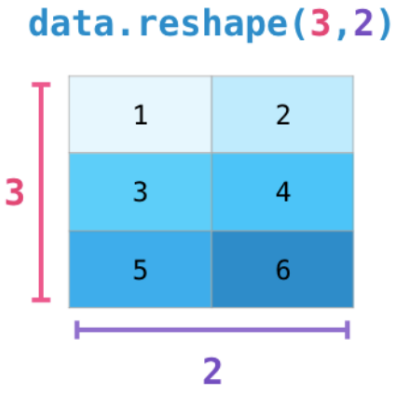
矩阵的转置和变形



In [31]:
```
# 矩阵转置
data = np.array([[1, 2], [3, 4], [5, 6]])
data.T
```

Out[31]:
```
array([[1, 3, 5],
       [2, 4, 6]])
```

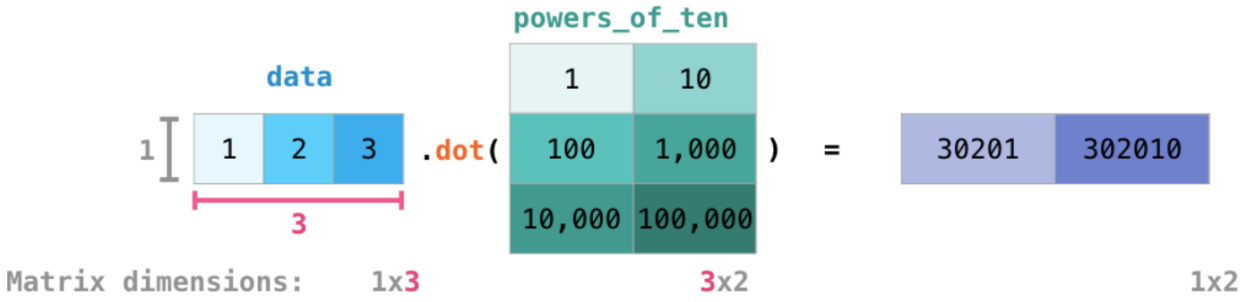




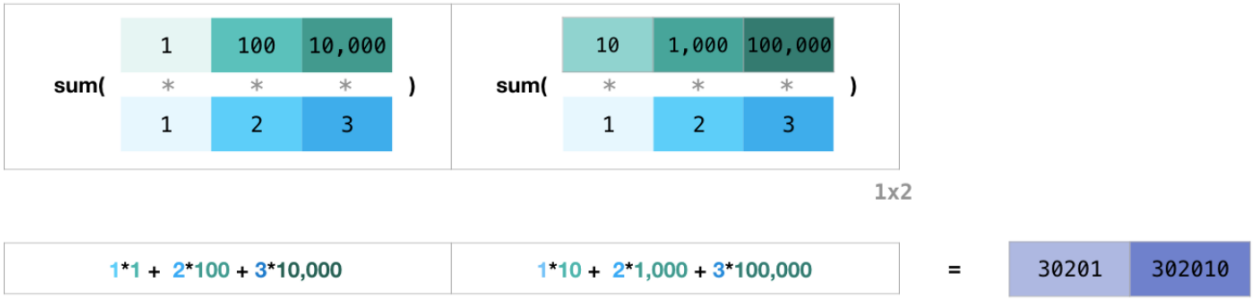
In [32]:
```
data = np.arange(1, 7)
print(data)
print(data.reshape(2, 3))
print(data.reshape(3, 2))
```

```
[1 2 3 4 5 6]
[[1 2 3]
 [4 5 6]]
[[1 2]
 [3 4]
 [5 6]]
```

线性代数



I've added matrix dimensions at the bottom of this figure to stress that the two matrices have to have the same dimension on the side they face each other with. You can visualize this operation as looking like this:



In [33]:
```
# 点乘
data = np.array([1, 2, 3])
power_of_tens = np.array([[1, 10],
                          [100, 1000],
                          [10000, 100000]])
data.dot(power_of_tens)
```

Out[33]:
```
array([ 30201, 302010])
```

In [34]:
```
# @符号表示点乘，也就是矩阵乘法
data @ power_of_tens
```

Out[34]:
```
array([ 30201, 302010])
```

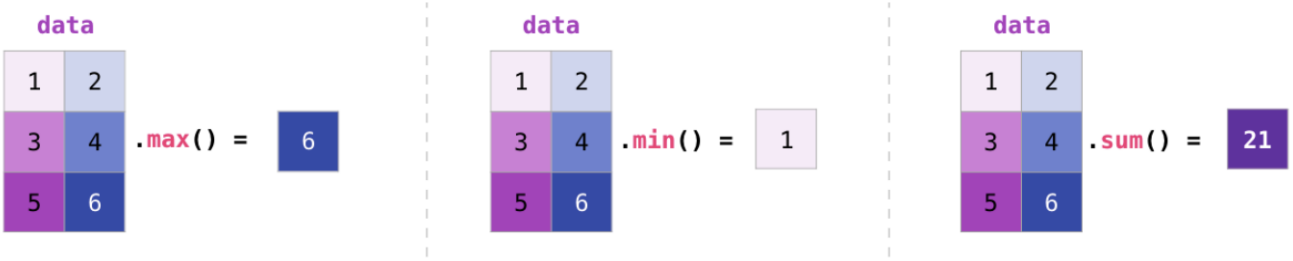
In [35]:
```
# 矩阵求逆
data = np.array([[1, 2], [3, 4]])
np.linalg.inv(data)
```

Out[35]:
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

In [36]:
```
# 矩阵行列式
np.linalg.det(data)
```

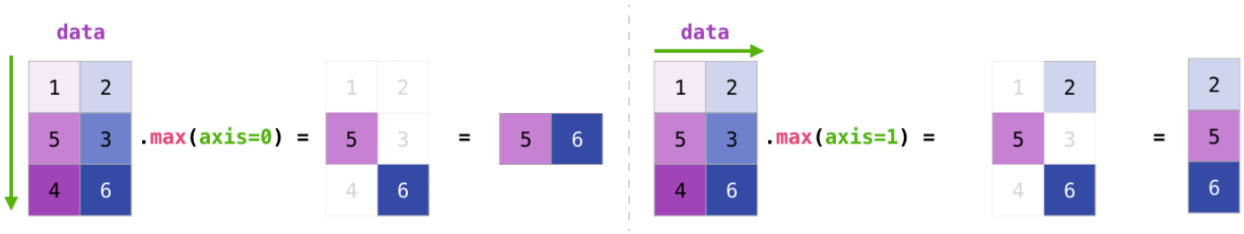
Out[36]:
```
-2.0000000000000004
```

矩阵的聚合



In [37]:
```
data = np.array([[1, 2],[3, 4], [5, 6]])
print(data.max())
print(data.min())
print(data.sum())
```

```
6
1
21
```

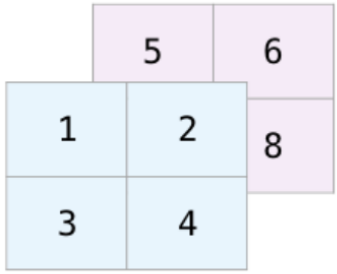


In [38]:
```
print('求每列最大值:',data.max(axis=0))

print('求每行最大值:',data.max(axis=1))

print('计算每列的和:',data.sum(axis=0))

print('计算每行的和:',data.sum(axis=1))
```

```
求每列最大值: [5 6]
求每行最大值: [2 4 6]
计算每列的和: [ 9 12]
计算每行的和: [ 3  7 11]
```

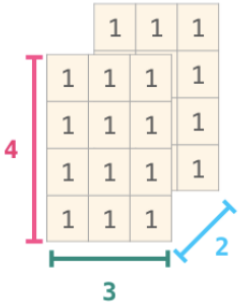
更高维的数组（tensor--张量）

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

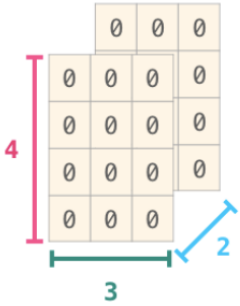


a lot of ways, dealing with a new dimension is just adding a comma to the parameters of a NumPy function:

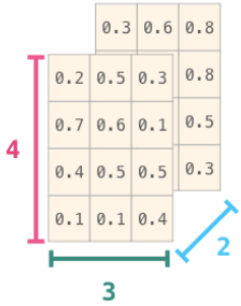
`np.ones((4,3,2))`



`np.zeros((4,3,2))`



`np.random.random((4,3,2))`



In [39]:
```
np.arange(1,9).reshape(2, 2, 2)
```

Out[39]:
```
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
```

In [41]:
```
print(np.ones((4, 3, 2)))
print(np.zeros((4, 3, 2)))
print(np.random.random((4, 3, 2)))
```

```
[[[1. 1.]
  [1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]
  [1. 1.]]]
[[[0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]]]
[[[0.65356987 0.74771481]
  [0.96130674 0.0083883 ]
  [0.10644438 0.29870371]]

 [[0.65641118 0.80981255]
  [0.87217591 0.9646476 ]
  [0.72368535 0.64247533]]

 [[0.71745362 0.46759901]
  [0.32558468 0.43964461]
  [0.72968908 0.99401459]]

 [[0.67687371 0.79082252]
  [0.17091426 0.02684928]
  [0.80037024 0.90372254]]]
```

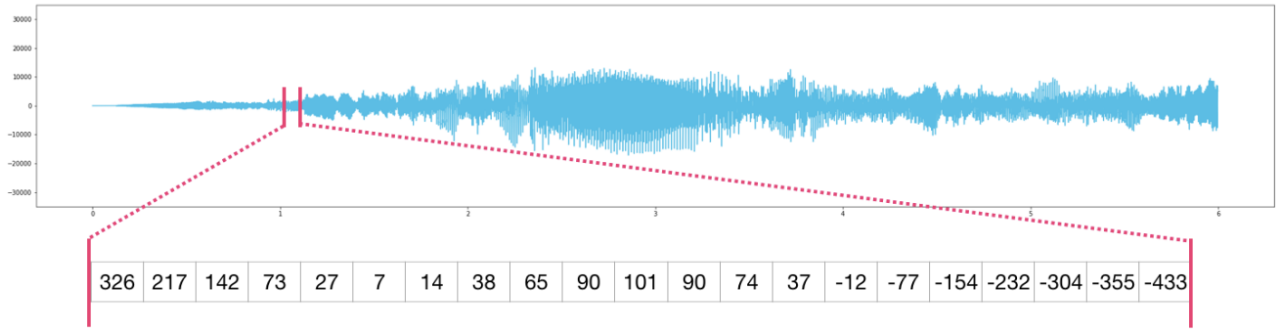
面向数列的编程（Array-Oriented Programming）

Array-Oriented Programming是一种编程范式，旨在利用NumPy库提供的强大数组操作功能来高效地执行数值计算和数据处理任务。

- 对整个数组或数组的子集进行操作
- 而不是使用for循环逐个处理数组中的元素

Numpy的应用

- 线性代数，统计学方面的运算
- 表示各种类型的数据：图像，音频，文本
- 用于机器学习和深度学习的算法



In [ ]:
```
import IPython
IPython.display.Audio("audio.mp3")
```

Out[7]:

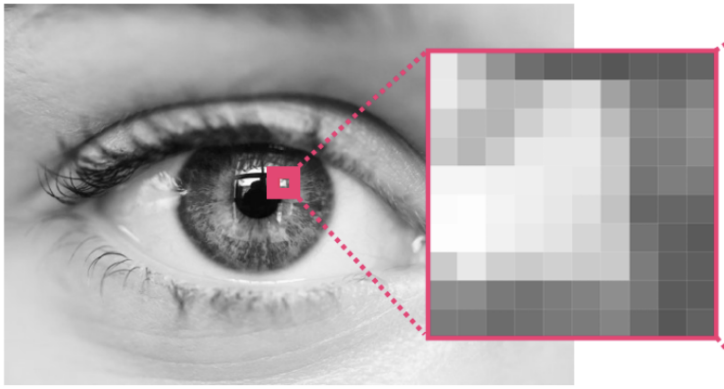
0:00 / 0:00

In [42]:
```
# 打印音频文件的码率
from pydub.utils import mediainfo

info = mediainfo("audio.mp3")
print("Bitrate:", info["bit_rate"], "b/s")
```

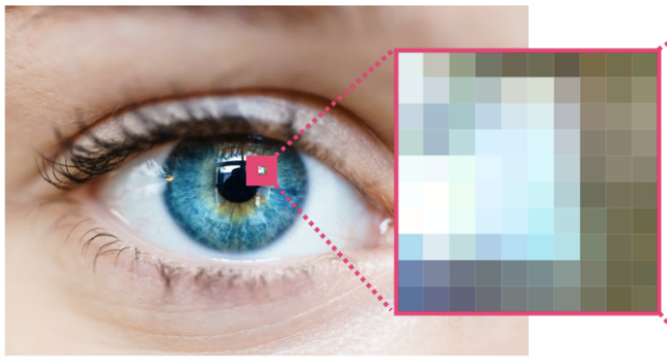
```
Bitrate: 64115 b/s
```

In [43]:
```
import numpy as np

# Read the MP3 file as binary data
with open('audio.mp3', 'rb') as file:
    mp3_data = file.read()

# Convert the binary data to a NumPy array
numpy_array = np.frombuffer(mp3_data, dtype=np.uint8)
print(len(numpy_array))

# Print the NumPy array
print(numpy_array[:20])
```

```
2265597
[ 73  68  51   3   0   0   0   0  31 118  84  89  69  82   0   0   0   1
   0   0]
```



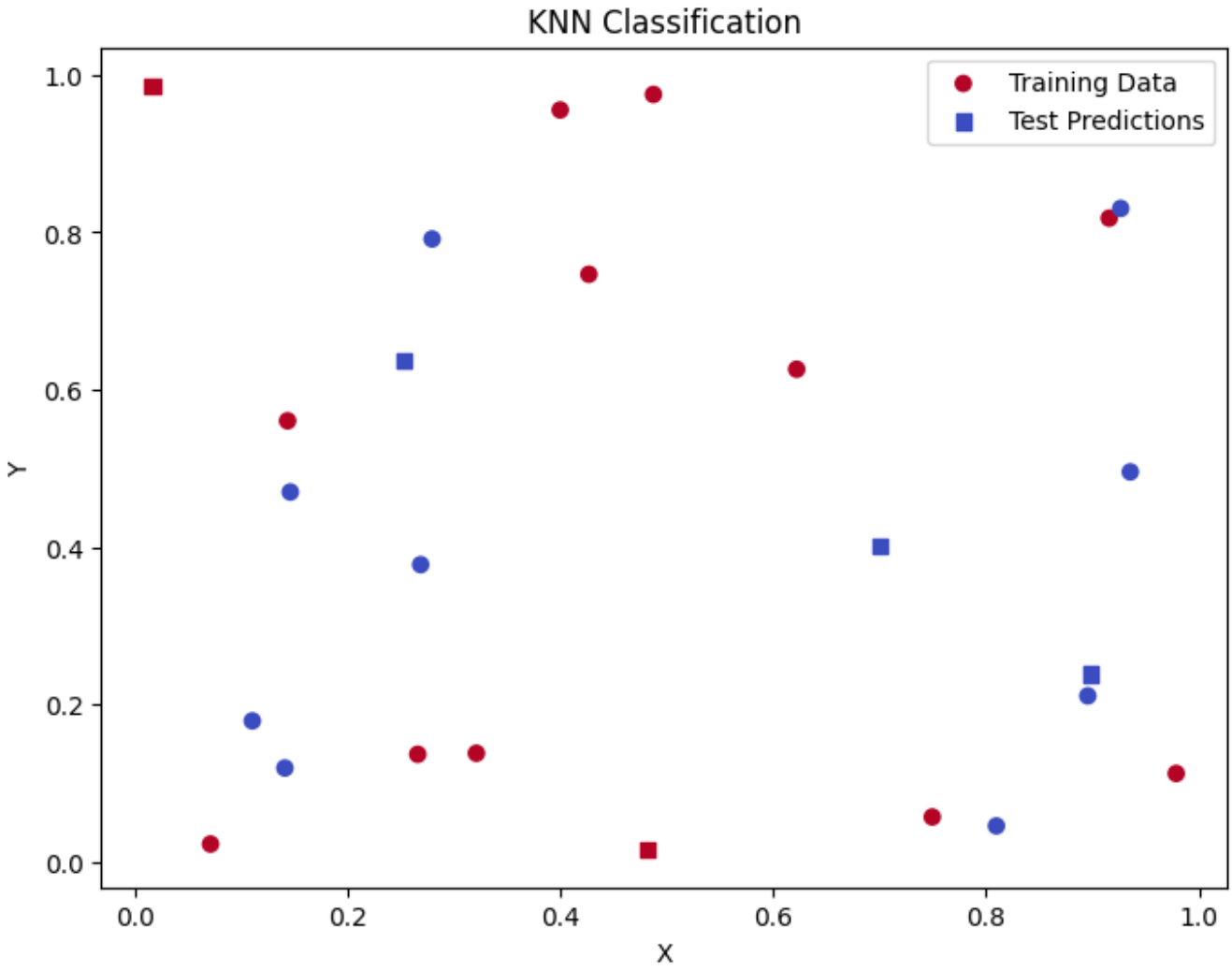
230	194	147	108	90	98	84	96	91	101
237	206	188	195	207	213	163	123	116	128
210	183	180	205	224	234	188	122	134	147
198	189	201	227	229	232	200	125	127	135
249	241	237	244	232	226	202	116	125	126
251	254	241	239	230	217	196	102	103	99
243	255	240	231	227	214	203	116	95	91
204	231	208	200	207	201	200	121	95	95
144	140	120	115	125	127	143	118	92	91
121	121	108	109	122	121	134	106	86	97



	233	188	137	96	90	95	63	73	73	82
237	202	159	120	105	110	88	107	112	121	109
226	191	147	110	101	112	98	123	110	119	142	131
221	191	176	182	203	214	169	144	133	145	155	122
185	160	161	184	205	223	186	137	147	161	140	115
181	174	189	207	206	215	194	136	142	151	133	87
246	237	237	231	208	206	192	122	143	144	111	74
254	254	241	224	199	192	181	99	122	117	107	74
239	248	232	207	187	182	184	110	114	110	113	74
193	215	193	167	158	164	181	114	112	111	105	82
113	119	110	111	113	123	135	120	108	106	113	
93	97	91	103	107	111	122	112	104	114		

In [ ]:

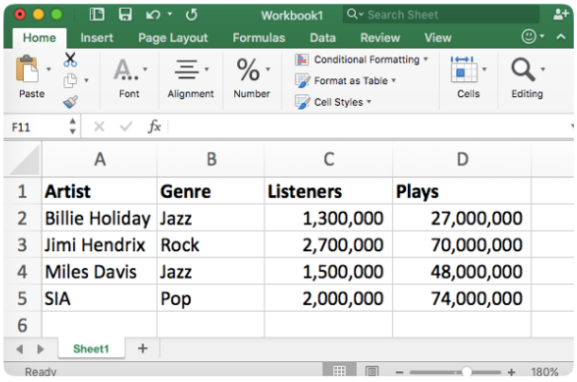
```
import numpy as np
import matplotlib.pyplot as plt

# knn算法
def knn(X_train, y_train, X_test, k):
    distances = np.sqrt(np.sum((X_train - X_test) ** 2, axis=1))
    nearest_indices = np.argsort(distances)[:k]
    nearest_labels = y_train[nearest_indices]
    unique_labels, counts = np.unique(nearest_labels, return_counts=True)
    return unique_labels[np.argmax(counts)]

# Generate random data
X_train = np.random.rand(20, 2)
y_train = np.random.choice([0, 1], size=20)
X_test = np.random.rand(5, 2)

# Classify test samples using KNN
k = 3
predictions = []
for sample in X_test:
    predicted_label = knn(X_train, y_train, sample, k)
    predictions.append(predicted_label)
predictions = np.array(predictions)

# Plot the results
plt.figure(figsize=(8, 6))
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='coolwarm', label='Training Data')
plt.scatter(X_test[:, 0], X_test[:, 1], c=predictions, cmap='coolwarm', marker='s', label='Test Pr
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.title('KNN Classification')
plt.show()
```



Pandas

加载数据

music.csv



pandas.read_csv('music.csv')

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	70,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

In [2]:
```
import pandas as pd

df = pd.read_csv('music.csv')
df
```

Out[2]:

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1300000	27000000
1	Jimi Hendrix	Rock	2700000	70000000
2	Miles Davis	Jazz	1500000	48000000
3	SIA	Pop	2000000	74000000

pandas可以从下面这些来源加载数据:

- CSV
- Excel
- HTML
- JSON
- SQL
- 等等其他

选择数据

In [ ]:
```
# 按照列名选择数据
df['Artist']
```

Out[23]:
```
0    Billie Holiday
1      Jimi Hendrix
2       Miles Davis
3               SIA
Name: Artist, dtype: object
```

In [ ]:
```
# 选择多列数据
df[['Artist', 'Plays']]
```

Out[24]:

	Artist	Plays
0	Billie Holiday	27000000
1	Jimi Hendrix	70000000
2	Miles Davis	48000000
3	SIA	74000000

In [ ]:
```
# 按照行索引选择数据
df[1:3]
```

Out[19]:

	Artist	Genre	Listeners	Plays
1	Jimi Hendrix	Rock	2700000	70000000
2	Miles Davis	Jazz	1500000	48000000

In [ ]:
```
# 按照行索引和列索引选择数据
df.loc[1:3, ['Artist', 'Plays']]
```

Out[26]:

	Artist	Plays
1	Jimi Hendrix	70000000
2	Miles Davis	48000000
3	SIA	74000000

过滤数据

In [ ]:
```
df[df['Genre']=='Jazz']
```

Out[27]:

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1300000	27000000
2	Miles Davis	Jazz	1500000	48000000

In [ ]:
```
df[df['Listeners'] > 1800000]
```

Out[29]:

	Artist	Genre	Listeners	Plays
1	Jimi Hendrix	Rock	2700000	70000000
3	SIA	Pop	2000000	74000000

处理数据缺失的情况

df

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
1	Jimi Hendrix	Rock	2,700,000	NaN
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

s to deal with this. The easiest is to just drop rows with missing values:

```
df.dropna()
```

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1,300,000	27,000,000
2	Miles Davis	Jazz	1,500,000	48,000,000
3	SIA	Pop	2,000,000	74,000,000

In [3]:
```
df2 = pd.read_csv('music copy.csv')
df2
```

Out[3]:

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1300000	27000000.0
1	Jimi Hendrix	Rock	2700000	NaN
2	Miles Davis	Jazz	1500000	48000000.0
3	SIA	Pop	2000000	74000000.0

In [4]:
```
df2.dropna()
```

Out[4]:

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1300000	27000000.0
2	Miles Davis	Jazz	1500000	48000000.0
3	SIA	Pop	2000000	74000000.0

In [5]:
```
df2.fillna(method='ffill')
```

Out[5]:

	Artist	Genre	Listeners	Plays
0	Billie Holiday	Jazz	1300000	27000000.0
1	Jimi Hendrix	Rock	2700000	27000000.0
2	Miles Davis	Jazz	1500000	48000000.0
3	SIA	Pop	2000000	74000000.0

Grouping

In [ ]:
```
df.groupby('Genre').sum()
```

Out[50]:

Genre	Listeners	Plays
Jazz	2800000	75000000
Pop	2000000	74000000
Rock	2700000	70000000

创建新的列

In [ ]:
```
df['Avg Plays'] = df['Plays'] / df['Listeners']
df
```

Out[51]:

	Artist	Genre	Listeners	Plays	Avg Plays
0	Billie Holiday	Jazz	1300000	27000000	20.769231
1	Jimi Hendrix	Rock	2700000	70000000	25.925926
2	Miles Davis	Jazz	1500000	48000000	32.000000
3	SIA	Pop	2000000	74000000	37.000000

扩展阅读 📖

- Numpy文章：A Visual Intro to NumPy and Data Representation (https://jalammar.github.io/visual-numpy/)
- Numpy文章的中文版：Numpy和数据展示的可视化介绍 (http://www.junphy.com/wordpress/index.php/2019/10/24/visual-numpy)
- Pandas文章:A Gentle Visual Intro to Data Analysis in Python Using Pandas (https://jalammar.github.io/gentle-visual-intro-to-data-analysis-python-pandas)

参考书：利用Python进行数据分析（原书第2版）(https://item.jd.com/12398725.html)

github地址：书籍源代码 (https://github.com/wesm/pydata-book)