

本课件仅用于教学使用。未经许可，任何单位、组织和个人不得将课件用于该课程教学之外的用途(包括但不限于盈利等)，也不得上传至可公开访问的网络环境



# 数据科学导论

## Introduction to Data Science

### 专题一：Python编程基础

肖桐

黄振亚，陈恩红

Email: [huangzhy@ustc.edu.cn](mailto:huangzhy@ustc.edu.cn), [cheneh@ustc.edu.cn](mailto:cheneh@ustc.edu.cn)

课程主页：

<http://staff.ustc.edu.cn/~huangzhy/Course/DS202.html>



# 目录

- Python、Jupyter notebook 简介
- Python 基础
- 矩阵运算示例：numpy 库
- 数据分析与特征处理示例
  - pandas 库
  - sklearn 库
    - 特征工程
    - 模型训练
  - 扩展：文本数据的分析
  - 扩展：深度学习库



# Python历史

- Python的创始人是荷兰人Guido van Rossum。1989年，为了打发圣诞节假期，Guido开始写Python语言的编译/解释器。
- Python名字来自Guido所喜爱的电视剧Monty Python's Flying Circus。他希望这个新的叫做Python的语言，能实现他的理念(一种C和shell之间，功能全面，易学易用，可拓展的语言)。
- Python 2于2000年10月16日发布，稳定版本是Python 2.7。Python 3于2008年12月3日发布，不完全兼容Python 2。



# Why Python?

- 语法简单，易上手。
- 强大的第三方库，如 `numpy`、`pandas`、`sklearn`。  
非常适合数据分析、数据处理、机器学习等任务。
- 代码简洁。  
有一个段子：完成同一个任务，C语言要写1000行代码，Java只需要写100行，而Python可能只要20行
- 不同于C语言，Python 中的变量不需要声明。  
每个变量在使用前都必须赋值，变量赋值以后该变量才会被创建。



# Python安装

- 方法1：直接安装
  - 下载对应操作系统的python版本

<https://www.python.org/downloads/release/python-385/>

## Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
<a href="#">Gzipped source tarball</a>	Source release		e2f52bcf531c8cc94732c0b6ff933ff0	24149103	<a href="#">SIG</a>
<a href="#">XZ compressed source tarball</a>	Source release		35b5a3d0254c1c59be9736373d429db7	18019640	<a href="#">SIG</a>
<a href="#">macOS 64-bit installer</a>	Mac OS X	for OS X 10.9 and later	2f8a736eeb307a27f1998cfd07f22440	30238024	<a href="#">SIG</a>
<a href="#">Windows help file</a>	Windows		3079d9cf19ac09d7b3e5eb3fb05581c4	8528031	<a href="#">SIG</a>
<a href="#">Windows x86-64 embeddable zip file</a>	Windows	for AMD64/EM64T/x64	73bd7aab047b81f83e473efb5d5652a0	8168581	<a href="#">SIG</a>
<a href="#">Windows x86-64 executable installer</a>	Windows	for AMD64/EM64T/x64	0ba2e9ca29b719da6e0b81f7f33f08f6	27864320	<a href="#">SIG</a>
<a href="#">Windows x86-64 web-based installer</a>	Windows	for AMD64/EM64T/x64	eeab52a08398a009c90189248ff43dac	1364128	<a href="#">SIG</a>
<a href="#">Windows x86 embeddable zip file</a>	Windows		bc354669bfd81a4ca14f06817222e50	7305731	<a href="#">SIG</a>
<a href="#">Windows x86 executable installer</a>	Windows		959873b37b74c1508428596b7f9df151	26777232	<a href="#">SIG</a>
<a href="#">Windows x86 web-based installer</a>	Windows		c813e6671f334a269e669d913b1f9b0d	1328184	<a href="#">SIG</a>

官方宣布，2020年1月1日，停止对Python2的更新。  
本次课程，我们以Python3为例进行教学。

# Python安装

## □ 方法2：安装python环境管理工具（推荐）

□ 下载：<https://www.anaconda.com/products/individual>

□ 优点：环境管理、含jupyter notebook等

### Windows

Python 3.8

64-Bit Graphical Installer (477 MB)

32-Bit Graphical Installer (409 MB)

### MacOS

Python 3.8

64-Bit Graphical Installer (440 MB)

64-Bit Command Line Installer (433 MB)

### Linux

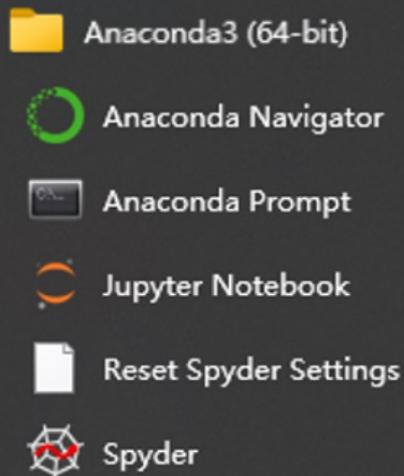
Python 3.8

64-Bit (x86) Installer (544 MB)

64-Bit (Power8 and Power9) Installer (285 MB)

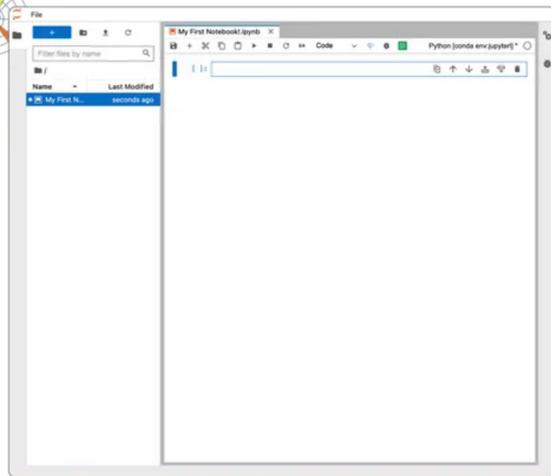
64-Bit (AWS Graviton2 / ARM64) Installer (413 M)

64-bit (Linux on IBM Z & LinuxONE) Installer (292 M)



# Jupyter notebook

- Jupyter notebook (<http://jupyter.org/>) 是一种 Web 应用，具备非常强的交互能力，能让用户将说明文本、代码和可视化内容全部组合到一个易于共享的文档中。



## Code anytime, anywhere

Introducing Anaconda Notebooks, a cloud-hosted notebook service that runs on any browser! Ready-to-code Anaconda environments. Hundreds of data science packages. Super snappy SSD storage. Native environment management. Preloaded extensions and sample notebooks. No configuration required — start coding from any modern web browser today.



# Jupyter notebook

## □ 使用Jupyter

- 打开开始菜单的jupyter notebook
- 浏览器会自动打开以下页面
- 选择 new Python 3 文件

localhost:8888/tree

🔍 ⚡ ☆ ▾ 百度

 jupyter

Quit

Logout

Files

Running

Clusters

Select items to perform actions on them.

0 /

Name ▾

代码列表为空, 请添加代码.

Upload

New ▾



Notebook:

Python 3

Other:

Text File

Folder

Terminal



# Jupyter notebook

□ 得到如下页面

jupyter Untitled 最新检查点: 几秒前 (更改未保存)

可信 Python 3

File Edit View Insert Cell Kernel Widgets Help

运行 代码

In [ ]:

# Jupyter notebook

- Jupyter notebook有两种模式：命令模式、编辑模式
- 编辑模式中允许往代码块中键入代码或文本
- 命令模式中允许输入宏观命令
- 在命令模式中按 H键，可以查看详细快捷键



The screenshot shows a Jupyter Notebook interface with a '快捷键' (Keyboard Shortcuts) dialog box open. The dialog box contains the following text:

Jupyter笔记本有两种不同的键盘输入模式。编辑模式允许您将代码或文本输入到一个单元格中，并通过一个绿色的单元格来表示。命令模式将键盘与笔记本级命令绑定在一起，并通过一个灰色的单元格边界显示，该边框为蓝色的左边框。

命令行模式(按 **Esc** 生效)

编辑快捷键

<b>F</b> : 查找并且替换	<b>Shift-T</b> : 扩展下面选择的代码块
<b>Ctrl-Shift-F</b> : 打开命令配置	<b>Shift-J</b> : 扩展下面选择的代码块
<b>Ctrl-Shift-P</b> : 打开命令配置	<b>A</b> : 在上面插入代码块
<b>Enter</b> : 进入编辑模式	<b>B</b> : 在下面插入代码块
<b>P</b> : 打开命令配置	<b>X</b> : 剪切选择的代码块
<b>Shift-Enter</b> : 运行代码块, 选择下面的代码块	<b>C</b> : 复制选择的代码块
<b>Ctrl-Enter</b> : 运行选中的代码块	<b>Shift-V</b> : 粘贴到上面
<b>Alt-Enter</b> : 运行代码块并且插入下面	<b>V</b> : 粘贴到下面



# Jupyter notebook

- 命令模式 (按 Esc 生效)
  - 命令模式时, 代码块是蓝色的
  - **Shift + Enter**: 运行当前代码块, 并选择下面的代码块
  - **X**: 剪切选择的代码块
  - **C**: 复制选择的代码块
  - **V**: 粘贴到下方代码块
  - **D+D**: 删除当前代码块
  - **Z**: 撤销删除

jupyter example (autosaved) 注销

文件 编辑 查看 插入 单元格 服务 Widgets 帮助 Not Trusted | Python 3 (ipykernel) ○

📄 + ↶ 📄 ↷ ⬆ ⬇ ▶ Run ■ ↺ ▶▶ Code ▼ 🗨

```
In [1]: print('Hello, World!')
```

Hello, World!

```
In [3]: counter = 100
miles = 1000.0
```



# Jupyter notebook

- 编辑模式(按 Enter 生效)
  - 编辑模式时，代码块是绿色的
  - Tab：代码补全或缩进
  - Ctrl+Z：撤销
  - Ctrl+/: 注释代码

The screenshot displays the Jupyter Notebook interface. At the top, the title bar reads "jupyter example (autosaved)" and includes a "注销" (Logout) button. Below the title bar is a menu bar with options: 文件 (File), 编辑 (Edit), 查看 (View), 插入 (Insert), 单元格 (Cell), 服务 (Services), Widgets, and 帮助 (Help). To the right of the menu bar, there is a "Not Trusted" warning, a pencil icon, and the text "Python 3 (ipykernel)".

The main area shows two code cells. The first cell, labeled "In [1]:", contains the code `print('Hello, World!')` and has executed, displaying the output "Hello, World!". This cell is highlighted with a red border. The second cell, labeled "In [3]:", contains the code `counter = 100` and `miles = 1000.0`.



# 1. Python基础

## □ 打印函数

```
In [1]: ▶ print("Hello World!")  
Hello World!
```

## □ 数字

```
In [2]: ▶ hello = "Hello World!"  
print(hello)  
Hello World!
```

```
In [3]: ▶ counter = 100 # 整型变量  
miles = 1000.0 # 浮点型变量
```

```
In [4]: ▶ print(miles + 4) # 加法  
print(3 * 7) # 乘法  
print(2 / 4) # 除法  
print(17 % 3) # 取余  
print(2 ** 5) # 乘方
```

```
1004.0  
21  
0.5  
2  
32
```

Python无需数据类型声明!

Java语言

```
01. public static void main(String[] args) {  
02.     byte a = 20; // 声明一个byte类型的变量并赋予初始值为20  
03.     short b = 10; // 声明一个short类型的变量并赋予初始值为10  
04.     int c = 30; // 声明一个int类型的变量并赋予初始值为30  
05.     long d = 40; // 声明一个long类型的变量并赋予初始值为40  
06.     long sum = a + b + c + d;  
07.     System.out.println("20+10+30+40=" + sum);  
08. }
```

实例

C语言

```
int i = 10;  
float f = 3.14;  
double d = i + f; // 隐式将int类型转换为double类型
```



# Python基础

## □ 字符串

```
In [5]: ▶ string = 'University of Science and Technology of China'

print(string)           # 输出字符串
print(string[0])       # 输出字符串第一个字符
print(string[-1])      # 输出字符串最后一个字符
print(string[10:])     # 输出从第11个开始的所有字符
print(string + " test") # 拼接字符串
print(string.split())  # 以空格为分隔符，分割字符串

University of Science and Technology of China
U
a
  of Science and Technology of China
University of Science and Technology of China test
['University', 'of', 'Science', 'and', 'Technology', 'of', 'China']
```



# Python基础

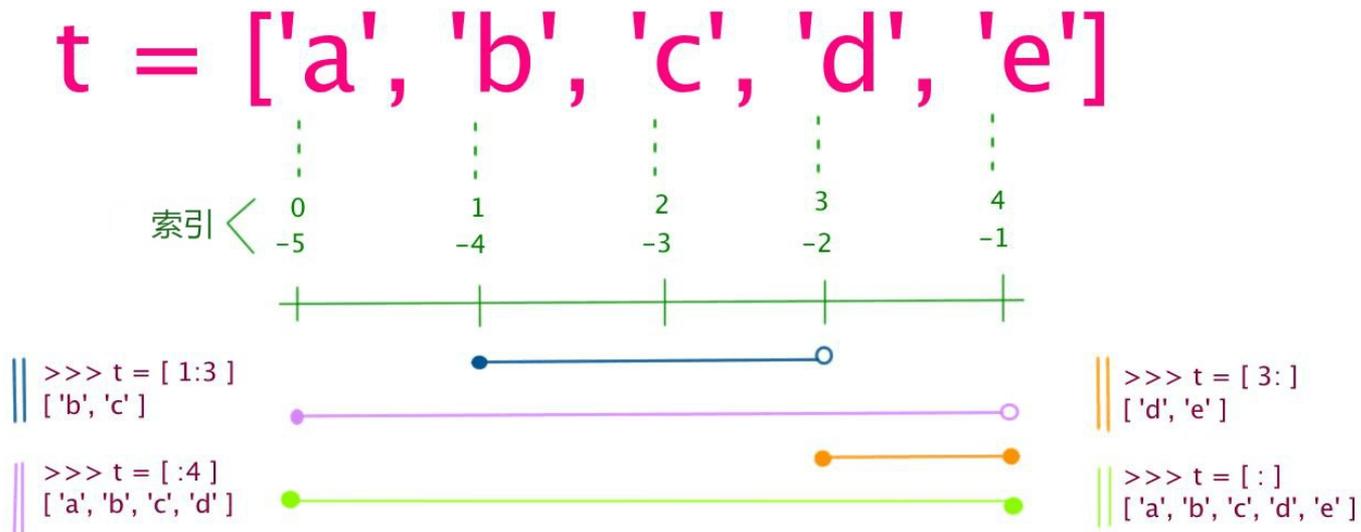
## □ 列表

□ 不同于C语言，python中的列表可以放任意类型的变量。

```
In [6]: ▶ a = [1, 2, 4, 'python', 8.9]
```

□ 列表截取

## Python 列表截取





# Python基础

## □ 列表操作 (\*)

```
In [8]: ▶ print(len(a))           # 输出列表长度  
         print(a[0])             # 输出列表第一个元素  
         print(a[-1])           # 输出列表最后一个元素  
         print(a[1:])           # 输出从第2个开始的所有元素  
         print(a + [-1, -2, -3]) # 拼接列表
```

```
5  
1  
8.9  
[2, 4, 'python', 8.9]  
[1, 2, 4, 'python', 8.9, -1, -2, -3]
```

```
In [9]: ▶ a.append(89)          # 在列表末尾添加新元素  
         a.pop()               # 移除列表中的最后一个元素  
         a.index(2)            # 寻找某个元素, 返回索引
```

```
Out[9]: 1
```



# Python基础

## □ 循环

### C语言

```
int a[5] = {1, 2, 4, 5, 7};  
  
for (int i=0; i<5; i++){  
    printf("%d\n", a[i]);  
}
```

```
In [10]: ▶ a = [1, 2, 4, 'python', 8.9]  
for i in a:  
    print(i)
```

```
1  
2  
4  
python  
8.9
```

```
In [11]: ▶ cnt = 0  
while cnt < len(a):  
    print(a[cnt])  
    cnt += 1
```

```
1  
2  
4  
python  
8.9
```



# Python基础

## □ 判断

```
In [12]: if 'python' == 'Python':  
         print(True)
```

```
In [13]: if 1 == 1.0:  
         print(True)  
         else:  
         print(False)
```

True

```
In [14] int age;  
scanf("Enter your age%d", &age);  
printf("\n");  
if(age <= 0)  
    printf("Are you kidding me!");  
else if(age >= 1 && age <= 110)  
    printf("fine.");  
else  
    printf("Are you kidding me.");
```

C语言



# Python基础

- Tuple (元组)
- Dictionary (字典)
- Set (集合)
- 函数

剩下的内容，请同学们参考

<https://docs.python.org/3/tutorial/>

<https://www.runoob.com/python3/python3-tutorial.html>



# Python基础

## □ Python练习

<https://code.bdaa.pro/exercise/ck77fhxo10219vdmgw66huhdw/coding>

<https://code.bdaa.pro/exercise/ck7h940vj0233y8mgy1hy6100/coding>

<https://code.bdaa.pro/exercise/ckcu68q4x5595stmgrhiawxj/coding>

CODIA 资源库 👤 ▼

### 最长平衡串

贡献者 [kaoyan](#) 提交情况  
0 / 0

**题目描述**

给定只含01的字符串，找出最长平衡子串的长度（平衡串：包含0和1的个数相同）。

来源：2019北邮机试题  
知识点：动态规划

**输入描述**

第一行输入串长 $n(1 \leq n \leq 100000)$   
第二行输入字符串

**输出描述**

输出最长的平衡子串的长度。

**样例输入**

```
8
11011011
```

**样例输出**

```
4
```

1

Python 提交



## 2. Numpy

<https://numpy.org/doc/stable/index.html>

- Python对高维矩阵的支持并不好
- Numpy提供了对矩阵的高效操作
- NumPy是Python中科学计算的基本软件包



# Numpy

## □ Python中调用Numpy

```
import numpy as np
```

## □ 生成一个4\*5的矩阵

```
In [2]: ▶ np.arange(20)
```

```
Out[2]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

```
In [3]: ▶ np.arange(20).reshape(4, 5)
```

```
Out[3]: array([[ 0,  1,  2,  3,  4],
               [ 5,  6,  7,  8,  9],
               [10, 11, 12, 13, 14],
               [15, 16, 17, 18, 19]])
```

```
In [4]: ▶ a = np.arange(20).reshape(4, 5)
```



# Numpy

## □ np的ndarray对象

```
In [5]: ▶ type(a)
```

```
Out[5]: numpy.ndarray
```

```
In [6]: ▶ a.ndim
```

```
Out[6]: 2
```

```
In [7]: ▶ a.shape
```

```
Out[7]: (4, 5)
```

```
In [8]: ▶ a.dtype
```

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



# Numpy

## □ 矩阵索引

```
In [9]: ▶ a
```

```
Out[9]: array([[ 0,  1,  2,  3,  4],
               [ 5,  6,  7,  8,  9],
               [10, 11, 12, 13, 14],
               [15, 16, 17, 18, 19]])
```

**获取第二行，第二列的元素**

```
In [10]: ▶ a[1, 1]
```

```
Out[10]: 6
```

每个维度一个索引值，逗号分割

**获取最后一行，最后一列的元素**

```
In [11]: ▶ a[-1, -1]
```

```
Out[11]: 19
```



# Numpy

## □ 矩阵切片

```
In [9]: ▶ a
```

```
Out[9]: array([[ 0,  1,  2,  3,  4],
               [ 5,  6,  7,  8,  9],
               [10, 11, 12, 13, 14],
               [15, 16, 17, 18, 19]])
```

**获取第一行的元素，与 a[0] 等价**

```
In [12]: ▶ a[0, :]
```

```
Out[12]: array([0, 1, 2, 3, 4])
```

单独的冒号代表取该维度所有元素

**获取第二列的元素**

```
In [13]: ▶ a[:, 1]
```

```
Out[13]: array([ 1,  6, 11, 16])
```



# Numpy

## □ 矩阵切片

```
In [9]: ▶ a
```

```
Out[9]: array([[ 0,  1,  2,  3,  4],
               [ 5,  6,  7,  8,  9],
               [10, 11, 12, 13, 14],
               [15, 16, 17, 18, 19]])
```

**第二行到第三行中, 后四列的元素**

```
In [14]: ▶ a[1:3, 1:]
```

```
Out[14]: array([[ 6,  7,  8,  9],
                [11, 12, 13, 14]])
```

**第二行到第四行中, 前四列的元素**

```
In [15]: ▶ a[1:4, :-1]
```

```
Out[15]: array([[ 5,  6,  7,  8],
                [10, 11, 12, 13],
                [15, 16, 17, 18]])
```



# Numpy

## □ 矩阵更改

```
In [9]: ▶ a
```

```
Out[9]: array([[ 0,  1,  2,  3,  4],
               [ 5,  6,  7,  8,  9],
               [10, 11, 12, 13, 14],
               [15, 16, 17, 18, 19]])
```

```
In [16]: ▶ a[1] = 1
```

```
In [17]: ▶ a
```

```
Out[17]: array([[ 0,  1,  2,  3,  4],
                [ 1,  1,  1,  1,  1],
                [10, 11, 12, 13, 14],
                [15, 16, 17, 18, 19]])
```

```
In [18]: ▶ a[0:2, 2:] = 2
```

```
In [19]: ▶ a
```

```
Out[19]: array([[ 0,  1,  2,  2,  2],
                [ 1,  1,  2,  2,  2],
                [10, 11, 12, 13, 14],
                [15, 16, 17, 18, 19]])
```



# Numpy

## □ 矩阵操作-加

```
In [20]: ▶ a + 1
```

```
Out[20]: array([[ 1,  2,  3,  3,  3],  
                [ 2,  2,  3,  3,  3],  
                [11, 12, 13, 14, 15],  
                [16, 17, 18, 19, 20]])
```

```
In [21]: ▶ a + a
```

```
Out[21]: array([[ 0,  2,  4,  4,  4],  
                [ 2,  2,  4,  4,  4],  
                [20, 22, 24, 26, 28],  
                [30, 32, 34, 36, 38]])
```



# Numpy

## □ 矩阵操作-乘 (\*)

```
In [19]: ▶ a
```

```
Out[19]: array([[ 0,  1,  2,  2,  2],
                [ 1,  1,  2,  2,  2],
                [10, 11, 12, 13, 14],
                [15, 16, 17, 18, 19]])
```

```
In [22]: ▶ a * 2
```

```
Out[22]: array([[ 0,  2,  4,  4,  4],
                [ 2,  2,  4,  4,  4],
                [20, 22, 24, 26, 28],
                [30, 32, 34, 36, 38]])
```

```
In [23]: ▶ a * [0.1, 1, 2, 4, 0] ← 广播机制(broadcasting)
```

```
Out[23]: array([[ 0. ,  1. ,  4. ,  8. ,  0. ],
                [ 0.1,  1. ,  4. ,  8. ,  0. ],
                [ 1. , 11. , 24. , 52. ,  0. ],
                [ 1.5, 16. , 34. , 72. ,  0. ]])
```



# Numpy

## □ 矩阵操作-乘 (\*)

### □ 矩阵乘法

```
In [24]: ▶ b = np.arange(15).reshape((5, 3))  
np.dot(a, b)
```

```
Out[24]: array([[ 57,   64,   71],  
                [ 57,   65,   73],  
                [390,  450,  510],  
                [540,  625,  710]])
```



# Numpy

## □ 矩阵操作-平方&三角函数

```
In [25]: ▶ a ** 2
```

```
Out[25]: array([[ 0,  1,  4,  4,  4],
                [ 1,  1,  4,  4,  4],
                [100, 121, 144, 169, 196],
                [225, 256, 289, 324, 361]], dtype=int32)
```

```
In [26]: ▶ np.sin(a)
```

```
Out[26]: array([[ 0.          ,  0.84147098,  0.90929743,  0.90929743,  0.90929743],
                [ 0.84147098,  0.84147098,  0.90929743,  0.90929743,  0.90929743],
                [-0.54402111, -0.99999021, -0.53657292,  0.42016704,  0.99060736],
                [ 0.65028784, -0.28790332, -0.96139749, -0.75098725,  0.14987721]])
```



# Numpy

## □ 矩阵操作-最大、最小、求和

```
In [27]: ▶ np.max(a)
```

```
Out[27]: 19
```

```
In [28]: ▶ np.min(a)
```

```
Out[28]: 0
```

```
In [29]: ▶ np.sum(a)
```

```
Out[29]: 160
```



# Numpy

## □ 矩阵操作-求和 (\*)

In [19]: ▶ a

```
Out[19]: array([[ 0,  1,  2,  2,  2],
                [ 1,  1,  2,  2,  2],
                [10, 11, 12, 13, 14],
                [15, 16, 17, 18, 19]])
```

### 以行为单位求和

In [30]: ▶ np.sum(a, axis=1)

```
Out[30]: array([ 7,  8, 60, 85])
```

### 以列为单位求和

In [31]: ▶ np.sum(a, axis=0)

```
Out[31]: array([26, 29, 33, 35, 37])
```



# Numpy

## □ 其他创建矩阵的方法

```
In [32]: ▶ b = np.array([[1, 5, 6.6],  
                        [9, 12, 5],  
                        [7, 8, 9]])
```

## □ 随机数初始化 (\*)

```
In [33]: ▶ np.random.rand(5, 3)
```

```
Out[33]: array([[0.86122248, 0.74729824, 0.14051075],  
               [0.25477071, 0.81669717, 0.65898755],  
               [0.45779822, 0.57063726, 0.07231267],  
               [0.38817019, 0.77935654, 0.90992532],  
               [0.48886097, 0.31464617, 0.81755649]])
```

```
In [34]: ▶ np.random.randn(5, 3)
```

```
Out[34]: array([[ 0.83656965, -0.49004958, -0.30325797],  
               [ 0.94391992, -2.3391026 ,  1.82955214],  
               [-1.22674497,  0.26722855, -1.05957563],  
               [-1.21578264,  2.0722414 ,  0.34791943],  
               [-0.02787046, -0.03013023, -0.24642746]])
```

# Numpy

- 更多方法可以参考官方文档



[https://numpy.org/doc/stable/user/absolute\\_beginners.html](https://numpy.org/doc/stable/user/absolute_beginners.html)

- 试一试：
  - 随机生成一个3x3的矩阵，并以行为单位进行求和。



# Numpy

## □ 试一试：

- 随机生成一个3x3的矩阵，并以行为单位进行求和。

```
# 随机生成一个3x3的矩阵，包含浮点数
random_matrix = np.random.rand(3, 3)

# 以行为单位求和
random_row_sums = random_matrix.sum(axis=1)

random_matrix, random_row_sums
```

生成的随机 3x3 矩阵如下：

$$\begin{bmatrix} 0.2033 & 0.7396 & 0.6996 \\ 0.2585 & 0.6488 & 0.8332 \\ 0.2015 & 0.0875 & 0.6387 \end{bmatrix}$$

每一行的和分别是：1.6425, 1.7405, 0.9278 [↗](#)

结果

```
(array([[0.20334485, 0.73957265, 0.69956596],
       [0.25854449, 0.64875472, 0.83317836],
       [0.20146653, 0.08753946, 0.63874498]]),
 array([1.64248346, 1.74047758, 0.92775098]))
```



# Numpy

- 试一试：
  - 随机生成一个3x3的整数矩阵，并以行为单位进行求和。



# Numpy

## □ 试一试：

- 随机生成一个3x3的整数矩阵，并以行为单位进行求和。

```
import numpy as np

# 随机生成一个3x3的整数矩阵
matrix = np.random.randint(1, 100, (3, 3))

# 以行为单位进行求和
row_sums = np.sum(matrix, axis=1)

matrix, row_sums
```

结果

```
(array([[56, 34, 33],
        [56, 36, 92],
        [48, 37, 24]]),
 array([123, 184, 109]))
```

生成的 3x3 整数矩阵如下：

$$\begin{bmatrix} 56 & 34 & 33 \\ 56 & 36 & 92 \\ 48 & 37 & 24 \end{bmatrix}$$

每一行的和分别为：

- 第一行：123
- 第二行：184
- 第三行：109 [\[-\]](#)



# 目录

- Python、Jupyter notebook 简介
- Python 基础
- 矩阵运算示例：numpy 库
- 数据分析与特征处理示例
  - pandas 库
  - sklearn 库
    - 特征工程
    - 模型训练
  - 扩展：文本数据的分析
  - 扩展：深度学习库



### 3. 数据分析与特征处理示例

- 本次教程将结合一个具体案例，教大家如何处理数据、分析数据、提取特征、数据可视化。
- 本次教程将会用到pandas库和sklearn库。



# 第三方库的安装

## □ Python: 打开命令行

- pip install numpy
- pip install pandas
- pip install scikit-learn

## □ Anaconda: 打开 Anaconda Prompt

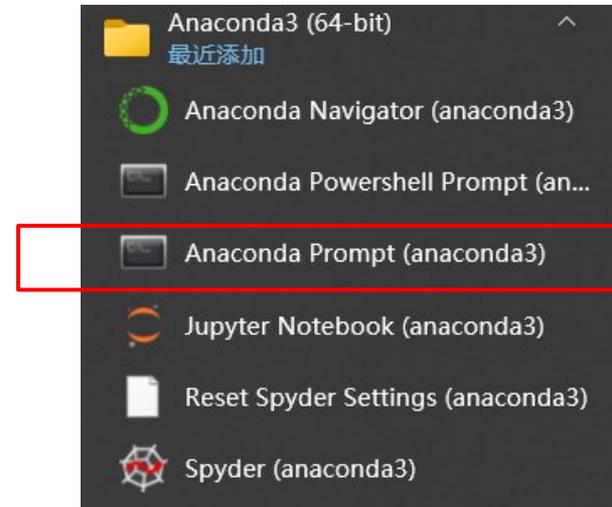
- pip install xxx (库名)
- 可选择清华镜像下载路径

■ pip install -i <https://mirrors.tuna.tsinghua.edu.cn/simple> xxx(库名)

## □ 另一种方法 (限windows用户)

- <https://www.lfd.uci.edu/~gohlke/pythonlibs/>  
下载对应的 wheel 文件

- 打开命令行, 输入 pip install + 对应 wheel 文件名





# 数据分析

- 数据分析环节，我们使用泰坦尼克号数据集
- 数据的下载地址为：

<https://www.kaggle.com/c/titanic/data>



GettingStarted Prediction Competition

## Titanic - Machine Learning from Disaster

Start here! Predict survival on the Titanic and get familiar with ML basics



Kaggle · 15,694 teams · Ongoing

Overview Data Code Discussion Leaderboard Rules

Join Competition





# 3.1 Pandas-数据分析

## □ 数据读取

### □ 命令: pd.read\_csv()

```
In [1]: ▶ import numpy as np           # 导入第三方库
import pandas as pd
```

```
In [2]: ▶ train = pd.read_csv(r"D:\Downloads\Data\train.csv")
test = pd.read_csv(r"D:\Downloads\Data\test.csv")
```

- PassengerId: ID
- Survived: 存活与否, 0 = No, 1 = Yes
- Pclass: 客舱等级, 1 = 1st, 2 = 2nd, 3 = 3rd
- Name: 姓名
- Sex: 性别
- Age: 当时的年龄
- SibSp: 泰坦尼克号上的兄弟姐妹/配偶数
- Parch: 泰坦尼克号上的父母/孩子数
- Ticket: 船票编号
- Fare: 票价
- Cabin: 客舱编号
- Embarked: 上船的港口编号, C = Cherbourg, Q = Queenstown, S = Southampton

## □ 数据总览

```
In [3]: print(train.shape)
train.head()
(891, 12)
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S



# Pandas-数据分析

## □ 数字特征总览（平均值、方差等指标）

□ 命令：`describe()`

### 数字特征

找一找：乘客的平均年龄是多少？

```
In [5]: train.describe()
```

Out[5]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200



# Pandas-数据分析

## □ 缺失值分析 (\*)

□ 命令: `isna()`

```
In [4]: ▶ train.isna().sum()
```

```
Out[4]: PassengerId    0 → 无缺失  
Survived              0  
Pclass                0  
Name                  0  
Sex                   0  
Age                   177 → 有177行缺失了该值  
SibSp                 0  
Parch                 0  
Ticket                0  
Fare                  0  
Cabin                 687  
Embarked              2  
dtype: int64
```

## □ 缺失值处理

□ 填充 `fillna()`

□ 丢弃 `dropna()`

▷

```
1 # 填充某一列的空值  
2 train["Age"] = train["Age"].fillna(0)  
3 train["Age"].isna().sum()
```

[8]

```
1 # 丢弃含有空值的行  
2 train = train.dropna(axis='index')  
3 train.isna().sum()
```

[9]



# Pandas-数据分析

## □ 特征取值统计

□ 命令: `nunique()`

```
1 # 乘客ID都是不同的
2 train.PassengerId.nunique() == train.shape[0]
[41] ✓ 0.2s
... True
```

□ 命令: `unique()`

```
1 # 船舱等级 有1, 2, 3级
2 train.Pclass.unique()
[40] ✓ 0.1s
... array([3, 1, 2], dtype=int64)
```

□ 命令: `value_counts()`

```
1 # 存活的人有342, 死亡的有549
2 train.Survived.value_counts()
[31] ✓ 0.5s
...
0    549
1    342
Name: Survived, dtype: int64
```

# Pandas-数据分析

## □ 特征取值可视化

□ plot()

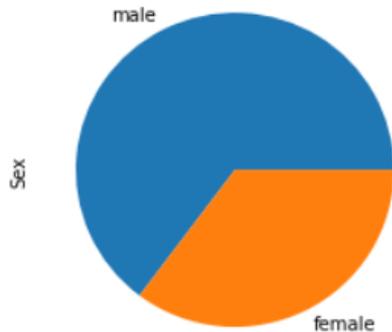
□ hist()

```
1 # 男女比  
2 train.Sex.value_counts().plot(kind='pie')
```

[24] ✓ 0.1s

... <AxesSubplot:ylabel='Sex'>

</>

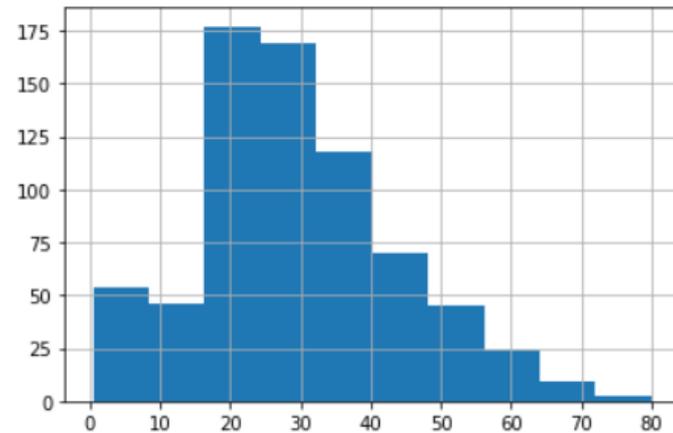


```
1 # 年龄分布直方图  
2 train.Age.hist(bins=10)
```

[28] ✓ 0.2s

... <AxesSubplot:>

</>





# Pandas-可视化分析

## □ 可视化分析 (\*)

□ 命令: `groupby()`

□ 命令: `mean()`

□ 命令: `sort_values()`

可以看到船舱等级越高, 存活率越高。

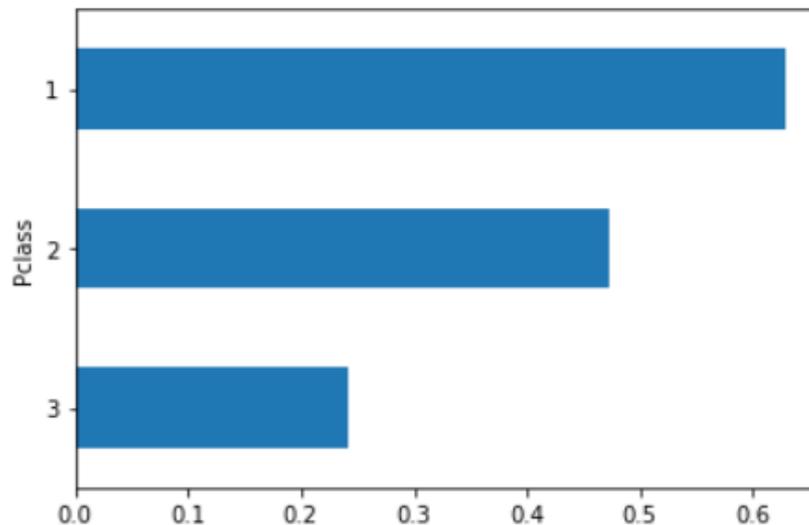
## 客舱等级分布

```
In [8]: ▶ train.Pclass.value_counts()
```

```
Out[8]: 3    491
         1    216
         2    184
         Name: Pclass, dtype: int64
```

```
In [9]: ▶ train.groupby('Pclass')['Survived'].mean().sort_values().plot(kind='barh')
```

```
Out[9]: <AxesSubplot:ylabel='Pclass'>
```



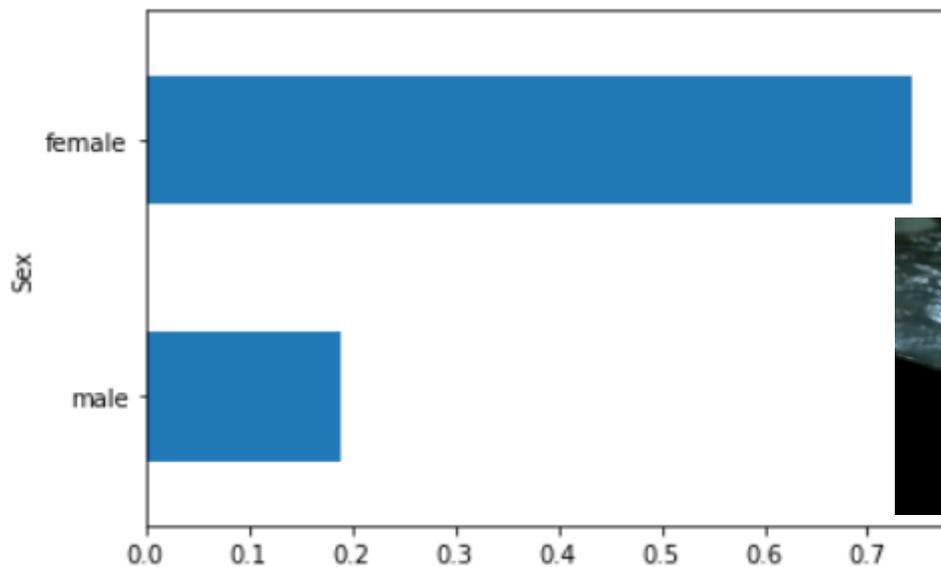
# Pandas-可视化分析

## □ 可视化分析

通过下面的图，可以看到女性的存活率高于男性。

```
In [11]: ▶ train.groupby('Sex')['Survived'].mean().sort_values().plot(kind='barh')
```

```
Out[11]: <AxesSubplot:ylabel='Sex'>
```



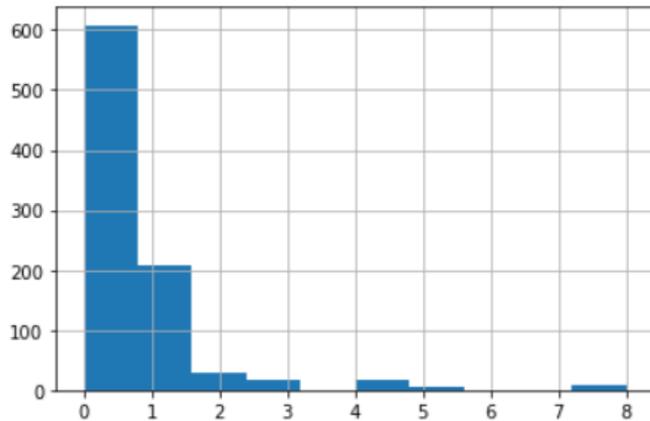
# Pandas-可视化分析

## □ 可视化分析

### SibSp分布

```
In [13]: ▶ train.SibSp.hist()
```

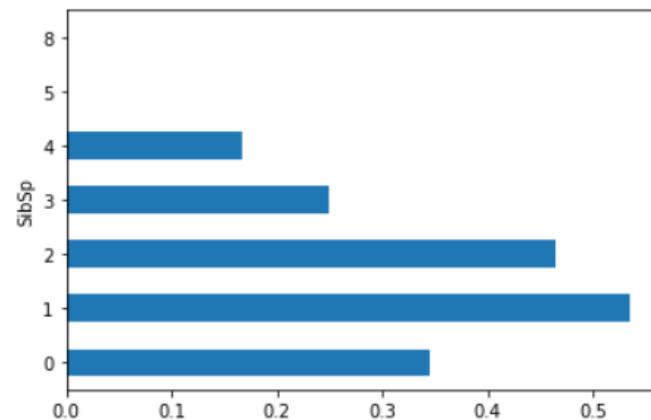
```
Out[13]: <AxesSubplot:>
```



### 兄弟姐妹数适中的乘客生存率更高

```
In [14]: ▶ train.groupby('SibSp')['Survived'].mean().plot(kind='barh')
```

```
Out[14]: <AxesSubplot:ylabel='SibSp'>
```





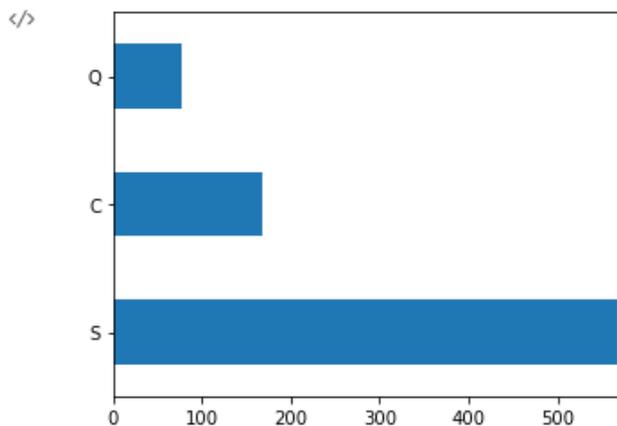
# Pandas-可视化分析

## □ 可视化分析

```
1 train.Embarked.value_counts().plot(kind='barh')
```

[16] ✓ 0.1s

... <AxesSubplot:>

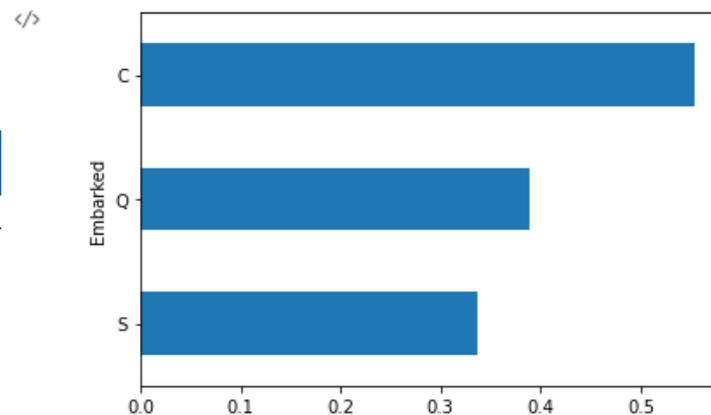


```
1 # 商船的港口为C的乘客存活率更高
```

```
2 train.groupby('Embarked')['Survived'].mean().sort_values().plot(kind='barh')
```

[17] ✓ 0.1s

... <AxesSubplot:ylabel='Embarked'>





# Pandas-可视化分析

## □ 特征取值统计

- 试一试：‘Cabin’列应该用什么方法分析？
  - 数值化数据 vs 类别数据
  - 缺失值



# Pandas-可视化分析

## □ 特征取值统计

□ 试一试：‘Cabin’ 列应该用什么方法分析？

■ 数值化数据 vs 类别数据

```
train['Cabin'].head(10)
```

```
In [6]: train['Cabin'].head(10)
```

```
Out[6]: 0      NaN  
1      C85  
2      NaN  
3     C123  
4      NaN  
5      NaN  
6     E46  
7      NaN  
8      NaN  
9      NaN
```

```
Name: Cabin, dtype: object
```

■ 缺失值

```
train['Cabin'] = train ['Cabin'].fillna( 'U' )
```



# Pandas-特征处理

## □ 文本特征的处理

```
train.head()
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Ticket_num	
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	NaN	S	21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	C85	C	17599
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	C123	S	113803
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	NaN	S	373450

比如说我们想要Ticket那一系列中的数字信息

```
▶ def process_Ticket(string):  
    try:  
        num = string.split()[-1]  
        num = int(num)  
    except:  
        num = -1  
    return num
```

```
▶ train['Ticket_num'] = train['Ticket'].apply(process_Ticket)
```

```
如: string str = "aa.bb.cc.dd";  
    string[] strArray = str.Split('.');
```

```
string[] {"aa", "bb", "cc", "dd"}
```



# Pandas-特征处理

- 类别特征的处理
  - 处理为数字

Sex那一列中的 'male'和 'female'是字符串，不能直接送给模型，因此要预处理。

```
In [21]: ▶ train['Sex'] = train['Sex'].map({'male': 1, 'female': 0})
```

```
train.head()
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Ticket_num	
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	NaN	S	21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	C85	C	17599
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	C123	S	113803
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	NaN	S	373450



# Pandas-特征处理

## 类别特征的处理 (\*)

处理为one-hot向量

命令: `pd.get_dummies()`

```
1 tmp = pd.get_dummies(data=tmp, columns=['Pclass'])
2 tmp.head()
```

[34] ✓ 0.1s

PassengerId	Survived	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Pclass_1	Pclass_2	Pclass_3
1	2	Cummings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	1	0	0
3	4	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	1	0	0
6	7	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S	1	0	0
10	11	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S	0	0	1
11	12	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S	1	0	0



# Pandas-特征处理

## □ 数值特征的处理

Fare这个特征，范围比较大，可以取一个 log。

```
In [24]: ▶ train['Fare_log'] = np.log(train['Fare'] + 0.0001) # 防止 log(0)
del train['Fare'] # 删除 Fare列
```

比如说我们想增加一列特征，记录每个在船上的亲戚数量。

```
In [25]: ▶ train['relative'] = train['SibSp'] + train['Parch']
```

```
In [26]: ▶ train[['Fare_log', 'SibSp', 'Parch', 'relative']].head(6)
```

Out[26]:

	Fare_log	SibSp	Parch	relative
0	1.981015	1	0	1
1	4.266663	1	0	1
2	2.070035	0	0	0
3	3.972179	1	0	1
4	2.085685	0	0	0
5	2.135160	0	0	0



# Pandas-特征处理

## □ 数值特征的处理

比如说我们想还增加一列特征，记录每个船舱等级的平均年龄。

```
In [27]: ▶ train['Pclass_Age_mean'] = train.groupby('Pclass')['Age'].transform('mean')
```

```
In [28]: ▶ train[['Pclass', 'Age', 'Pclass_Age_mean']].head(10)
```

Out[28]:

	Pclass	Age	Pclass_Age_mean
0	3	22.0	25.140620
1	1	38.0	38.233441
2	3	26.0	25.140620
3	1	35.0	38.233441
4	3	35.0	25.140620
5	3	NaN	25.140620
6	1	54.0	38.233441
7	3	2.0	25.140620
8	3	27.0	25.140620
9	2	14.0	29.877630



# Pandas-特征处理

- 更多方法可以参考官方文档
  - [https://pandas.pydata.org/docs/user\\_guide/index.html](https://pandas.pydata.org/docs/user_guide/index.html)





# Sklearn-特征处理

## □ □ 2. 数据集的预处理

### a). 数据的标准化

$$x' = \frac{x - \bar{X}}{S}$$

其中 $\bar{X}$ 为均值， $S$ 为标准差

```
In [3]: ▶ from sklearn.preprocessing import StandardScaler
StandardScaler().fit_transform(iris.data)

Out[3]: array([[ -9.00681170e-01,  1.01900435e+00, -1.34022653e+00,
                -1.31544430e+00],
                [-1.14301691e+00, -1.31979479e-01, -1.34022653e+00,
                -1.31544430e+00],
                [-1.38535265e+00,  3.28414053e-01, -1.39706395e+00,
                -1.31544430e+00],
                [-1.50652052e+00,  9.82172869e-02, -1.28338910e+00,
                -1.31544430e+00],
                [-1.02184904e+00,  1.24920112e+00, -1.34022653e+00,
                -1.31544430e+00],
```



# Sklearn-特征处理

## □ □ 2. 数据集的预处理

b). 数据的归一化(规则为L2公式如下):

$$x' = \frac{x}{\sqrt{\sum_j^m x[j]^2}}$$

对特征矩阵的行处理数据，其中m为向量的维度

```
In [4]: ▶ from sklearn.preprocessing import Normalizer
```

```
Normalizer().fit_transform(iris.data)
```

```
Out[4]: array([[0.80377277, 0.55160877, 0.22064351, 0.0315205 ],
 [0.82813287, 0.50702013, 0.23660939, 0.03380134],
 [0.80533308, 0.54831188, 0.2227517 , 0.03426949],
 [0.80003025, 0.53915082, 0.26087943, 0.03478392],
 [0.790965 , 0.5694948 , 0.2214702 , 0.0316386 ],
 [0.78417499, 0.5663486 , 0.2468699 , 0.05808704],
 [0.78010936, 0.57660257, 0.23742459, 0.0508767 ],
 [0.80218492, 0.54548574, 0.24065548, 0.0320874 ],
 [0.80642366, 0.5315065 , 0.25658935, 0.03665562],
 [0.81803119, 0.51752994, 0.25041771, 0.01669451],
```



# Sklearn-特征处理

## □ 3. 特征的选择

□ a). **Filter**: 过滤法，按照发散性或者相关性对各个特征进行评分，设定阈值或者待选择阈值的个数，选择特征。

- **方差选择法**: 使用**方差**做为Filter的特征评价函数，先要计算各个特征的方差，然后根据 阈值选择方差大于阈值的特征。使用 sklearn 通过方差选择法来选择特征的
- 相关系数法、卡方检验、互信息法 ...

```
In [6]: iris.data
```

```
Out[6]: array([[5.1, 3.5, 1.4, 0.2],  
              [4.9, 3. , 1.4, 0.2],  
              [4.7, 3.2, 1.3, 0.2],  
              [4.6, 3.1, 1.5, 0.2],  
              [5. , 3.6, 1.4, 0.2],  
              [5.4, 3.9, 1.7, 0.4],  
              [4.6, 3.4, 1.4, 0.3],  
              [5. , 3.4, 1.5, 0.2],  
              [4.4, 2.9, 1.4, 0.2],
```



```
In [5]: from sklearn.feature_selection import VarianceThreshold
```

```
# 方差选择法，返回值为特征选择后的数据  
# 参数为方差的阈值  
VarianceThreshold(threshold=3).fit_transform(iris.data)
```

```
Out[5]: array([[1.4],  
              [1.4],  
              [1.3],  
              [1.5],  
              [1.4],  
              [1.7],
```



# Sklearn-特征处理

- 3. 特征的选择
- b). Wrapper: 包装法，根据目标函数（通常是预测效果评分），每次选择若干特征，或者排除若干特征。
  - **递归特征消除法：** 对一个基模型来进行多轮训练，每轮训练后，消除若干权值系数特征，再基于新的特征集进行下一轮训练

```
In [7]: ▶ from sklearn.feature_selection import RFE
        from sklearn.linear_model import LogisticRegression

        # 递归特征消除法，返回特征选择后的数据
        # 参数为基模型estimator与选择的特征个数n_features_to_select
        RFE(estimator=LogisticRegression(),n_features_to_select=2).fit_transform(iris.data, iris.target)
```

```
Out[7]: array([[1.4, 0.2],
               [1.4, 0.2],
               [1.3, 0.2],
               [1.5, 0.2],
               [1.4, 0.2],
               [1.7, 0.4],
               [1.4, 0.3],
               [1.5, 0.2],
```



# Sklearn-特征处理

- 3. 特征的选择
- c). Embedded: 嵌入式方法，使用某些机器学习的算法和模型进行训练，得到各个特征的权值系数，根据系数从大到小选择特征
  - 基于树模型的特征选择法有决策树、随机森林和GBDT等方法。在这里以 **GBDT** 来选择特征为例，具体 sklearn 的实现代码如下所示：

```
In [8]: ▶ from sklearn.feature_selection import SelectFromModel
        from sklearn.ensemble import GradientBoostingClassifier

        # 基模型选择GBDT
        SelectFromModel(GradientBoostingClassifier()).fit_transform(iris.data, iris.target)
```

```
Out[8]: array([[1.4, 0.2],
               [1.4, 0.2],
               [1.3, 0.2],
               [1.5, 0.2],
               [1.4, 0.2],
               [1.7, 0.4],
               [1.4, 0.3],
               [1.5, 0.2],
```



# Sklearn-特征处理

## 3. 特征的降维

- 当特征选择完成后，可以直接训练模型了，但是可能由于特征矩阵过大，导致计算量大，训练时间长的的问题，因此降低特征矩阵维度也是必不可少的。

### a). 主成分分析（PCA）



```
1 from sklearn.decomposition import PCA
2
3 pca = PCA(n_components=2)
4 # 主成分分析法, 返回降维后的数据
5 data = pca.fit_transform(iris.data)
6 print(data)
7
8 print(pca.explained_variance_ratio_) # 特征权重
9 print(pca.singular_values_) # 特征值
```



```
[[ -2.68412563  0.31939725]
 [ -2.71414169 -0.17700123]
 [ -2.88899057 -0.14494943]
 [ -2.74534286 -0.31829898]
 [ -2.72871654  0.32675451]
 ...
 [ 1.90094161  0.11662796]
 [ 1.39018886 -0.28266094]]
[0.92461872 0.05306648]
[25.09996044  6.01314738]
```

### b). 线性判别分析法（LDA）



```
1 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
2
3 # 线性判别分析法, 返回降维后的数据
4 LDA(n_components=2).fit_transform(iris.data, iris.target)
```



# Sklearn-训练模型

## □ 1. 分割训练、测试集

```
▶ from sklearn.model_selection import train_test_split

# 划分训练、测试集
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=0)
```

## □ 2. 选择模型

- 1) 线性回归
- 2) logistic回归
- 3) 决策树分类
- 4) SVM

```
▶ ▾
1 from sklearn import linear_model
2 lin_reg = linear_model.LinearRegression() # 线性回归
3 log_reg = linear_model.LogisticRegression() # 逻辑回归
[16] ✓ 0.3s
```

```
1 from sklearn.tree import DecisionTreeClassifier
2 from sklearn.svm import LinearSVC
3
4 decision_tree_classifier = DecisionTreeClassifier() # 决策树分类
5 svc = LinearSVC() # 支持向量机
```



# Sklearn-训练模型

## □ 3. 训练模型

```
1 lin_reg.fit(X_train, y_train)
2
3 print(f'intercept: {lin_reg.intercept_: .2f}') # 偏移 (截距)
4 print('coef: ', lin_reg.coef_) # 系数 (斜率)
```

[39] ✓ 0.5s

... intercept: 0.16

coef: [-0.10627533 -0.0397204 0.22894234 0.61123074]

## □ 4. 测试模型

```
1 test_predict = lin_reg.predict(X_test)
```

[40] ✓ 0.4s

## □ 5. 模型评估

```
1 lin_reg.score(X_test, y_test)
```

[41] ✓ 0.4s

... 0.9055032992676105



# Sklearn-训练模型

- 进阶
  - 交叉验证
  - 参数搜索
  - 模型集成
  - ...
- 其他模型

```
1 def model_train(model, model_name, kfold=5):
2     oof_preds = np.zeros((train.shape[0]))
3     test_preds = np.zeros(test.shape[0])
4     # K折划分
5     skf = KFold(n_splits=kfold)
6
7     print(f"Model = {model_name}")
8     for k, (train_index, test_index) in enumerate(skf.split(train, label)):
9         x_train, x_test = train.iloc[train_index, :], train.iloc[test_index, :]
10        y_train, y_test = label.iloc[train_index], label.iloc[test_index]
11        # 训练
12        model.fit(x_train, y_train)
13        # 验证
14        y_pred = model.predict_proba(x_test)[: , 1]
15        oof_preds[test_index] = y_pred.ravel()
16        auc = roc_auc_score(y_test, y_pred)
17        print("- KFold = %d, val_auc = %.4f" % (k, auc))
18        # 预测
19        test_fold_preds = model.predict_proba(test)[: , 1]
20        test_preds += test_fold_preds.ravel()
21
22    print("Overall Model = %s, AUC = %.4f" % (model_name, roc_auc_score(label, oof_preds)))
23    return test_preds / kfold # 取K个模型的平均预测概率
```

[6] ✓ 0.3s

Python

## scikit-learn

*Machine Learning in Python*

更多用法参考sklearn官方文档: <https://scikit-learn.org/stable/index.html>

# 扩展：文本数据的分析



## □ 案例：议论文结构识别

### □ 赛题官网：

<https://www.kaggle.com/competitions/feedback-prize-2021>

### □ 作文数据：

[https://www.kaggle.com/datasets/longhuqin/myfeedbackdata?select=train\\_essay.csv](https://www.kaggle.com/datasets/longhuqin/myfeedbackdata?select=train_essay.csv)

### □ 数据读取

```
1 train_df = pd.read_csv("train_essay.csv")
2 print(train_df.shape)
3 train_df.head()
```

[9] ✓ 0.5s

... (15594, 3)

id	text
0000D23A521A	Some people believe that the so called "face" o...
00066EA9880D	Driverless cars are exactly what you would exp...
000E6DE9E817	Dear: Principal\n\nI am arguing against the po...
001552828BD0	Would you be able to give your car up? Having ...
0016926B079C	I think that students would benefit from learn...



# 扩展：文本数据的分析

## □ 文本聚类分析

(词袋)  
Tf 编码

文本编码 - Tfidf

进阶版

词表:	1 我;	2 爱;	3 爸;	4 妈;	5 中国
我爱中国	->	1, 1, 0, 0, 1			
爸爸妈妈爱我	->	1, 1, 2, 2, 0			
爸爸妈妈爱中国	->	0, 1, 2, 2, 1			

[3]

```
1 tfidf = TfidfVectorizer(binary=True, max_features=25_000)
2 text_embeddings = tfidf.fit_transform(train_df["text"]).toarray()
```

✓ 4.3s

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

## √ 降维 - t-SNE

- 其他降维方法: PCA

[4]

```
1 ts = TSNE(n_components=2)
2 embed_2d = ts.fit_transform(text_embeddings)
```

✓ 46.7s

```
array([[ 14.929427, -47.857616 ],
       [-2.5265694,  57.226536 ],
       [-3.2941248, -30.974361 ],
       ...,
       [ 18.113998, -51.366302 ],
       [-14.548886,  18.839033 ],
       [-22.015995,  23.30858  ]], dtype=float32)
```

# 扩展：文本数据的分析

## □ 文本聚类分析

### 聚类 - K-means

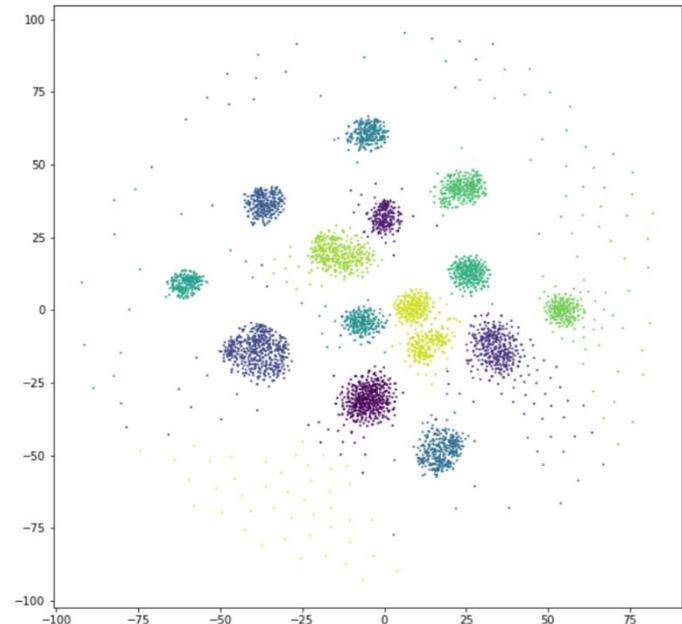
```
1 kmeans = KMeans(n_clusters= 15)
2 kmeans.fit(embed_2d)
3
4 label = kmeans.labels_.tolist()
```

[5] ✓ 0.6s

### 可视化

```
1 plt.figure(figsize=(10,10))
2 plt.scatter(embed_2d[:,0], embed_2d[:,1], s=1, c=label)
3 plt.show()
```

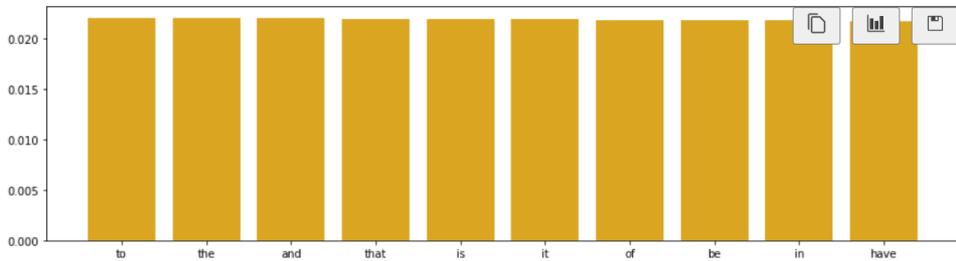
[6] ✓ 0.6s



# 扩展：文本数据的分析

## 高频词分析

```
1 import jieba.analyse as analyse
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 top_num = 10
6
7 # 求整个语料库的 TFIDF 值
8 mm = np.mean(text_embeddings, axis=0 )
9 # TopK TFIDF值对应的词表索引
10 ii = np.argsort(mm)[-top_num:][::-1].tolist()
11 # 求索引对应的词
12 top_words = [(tfidf.get_feature_names()[i], mm[i]) for i in ii]
```

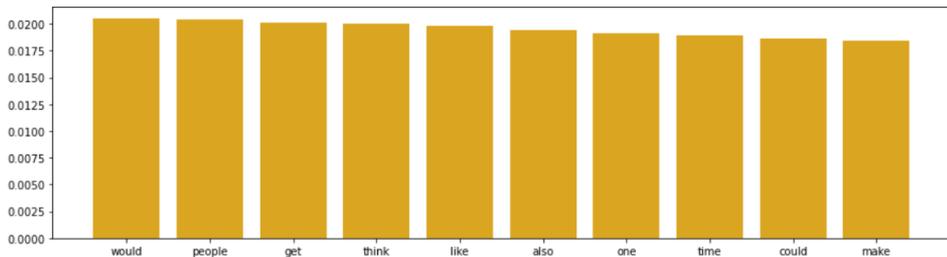


问题：直接统计得到的是无意义的停用词

↓ 去除停用词

## 其他分析与处理

- 长度分析、去除停用词
- 语法特征挖掘
- 主题分析 LDA





# 扩展：深度学习主流框架

- Tensorflow官网：<https://tensorflow.google.cn/>
  - pip install tensorflow
  - 可参考<https://tensorflow.google.cn/install/>
- Pytorch官网：<https://pytorch.org/>
  - 根据设备配置找到安装命令
  - 建议使用清华镜像下载(去掉-f及之后的内容，使用37页的-i命令)

PyTorch Build	Stable (1.10)	Preview (Nightly)	LTS (1.8.2)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 10.2	CUDA 11.3	ROCm 4.2 (beta)	CPU
Run this Command:	<pre>pip3 install torch==1.10.0+cu102 torchvision==0.11.1+cu102 torchaudio==0.10.0+cu102 -f https://download.pytorch.org/whl/cu102/torch_stable.html</pre>			



# 扩展：深度学习NLP库

## □ 扩展

### □ NLP库

- jieba <https://github.com/fxsjy/jieba>
- spacy <https://spacy.io/usage/linguistic-features>
- nltk <https://www.nltk.org/index.html>

### □ 词向量

<https://radimrehurek.com/gensim/models/word2vec.html>

### □ 预训练模型

[https://huggingface.co/docs/transformers/model\\_doc/bert?highlight=bertpretrainedmodel#transformers.BertModel](https://huggingface.co/docs/transformers/model_doc/bert?highlight=bertpretrainedmodel#transformers.BertModel)

fxsjy / jieba

spaCy

NLTK

 **GENSIM**  
topic modelling for humans



**Hugging Face**