

同济大学
TONGJI UNIVERSITY

王睿智

ruizhiwang@tongji.edu.cn

人工智能技术与应用

2.1 Numpy 简介

2.2 pandas 简介

2.2.1 引例

2.2.2 Pandas的数据结构

2.2.3 读写文件操作

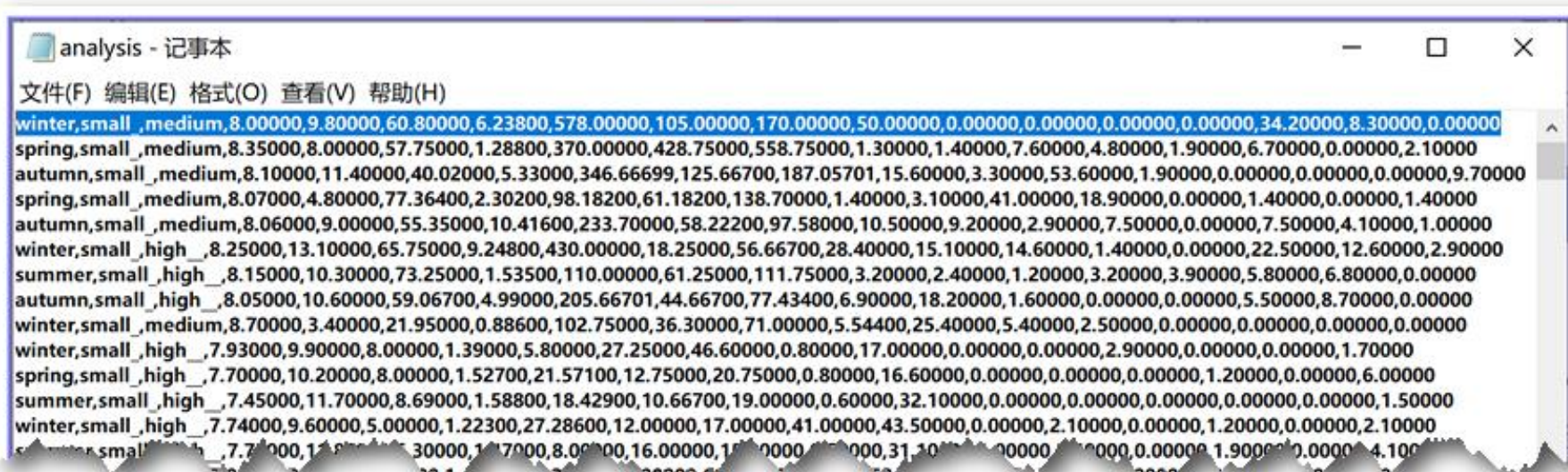
2.2.4 数据预处理

有害海藻数据来自1999年计算智能和学习(COIL)竞赛, 现可从[UCI机器学习数据库](https://archive.ics.uci.edu/) (<https://archive.ics.uci.edu/>) 可下载。本例选其中的analysis.data数据集。

该数据集有200个样本, 每个样本包含18个分量: 前11个分量是季节、河流大小、流速和8个与藻类种群分布相关的化学物质浓度; 后7个分量是不同种类有害藻类在相应水样中的频率数目。值0.0表示频率非常低。该数据集包含一些用字符串XXXXXXXX标记的空字段。

竞赛要求用这些数据训练一个预测模型, 可以根据前11个量的取值, 预测7种有害藻类的频率。

数据



```
analysis - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
winter,small_,medium,8.00000,9.80000,60.80000,6.23800,578.00000,105.00000,170.00000,50.00000,0.00000,0.00000,0.00000,0.00000,34.20000,8.30000,0.00000
spring,small_,medium,8.35000,8.00000,57.75000,1.28800,370.00000,428.75000,558.75000,1.30000,1.40000,7.60000,4.80000,1.90000,6.70000,0.00000,2.10000
autumn,small_,medium,8.10000,11.40000,40.02000,5.33000,346.66699,125.66700,187.05701,15.60000,3.30000,53.60000,1.90000,0.00000,0.00000,0.00000,9.70000
spring,small_,medium,8.07000,4.80000,77.36400,2.30200,98.18200,61.18200,138.70000,1.40000,3.10000,41.00000,18.90000,0.00000,1.40000,0.00000,1.40000
autumn,small_,medium,8.06000,9.00000,55.35000,10.41600,233.70000,58.22200,97.58000,10.50000,9.20000,2.90000,7.50000,0.00000,7.50000,4.10000,1.00000
winter,small_,high_,8.25000,13.10000,65.75000,9.24800,430.00000,18.25000,56.66700,28.40000,15.10000,14.60000,1.40000,0.00000,22.50000,12.60000,2.90000
summer,small_,high_,8.15000,10.30000,73.25000,1.53500,110.00000,61.25000,111.75000,3.20000,2.40000,1.20000,3.20000,3.90000,5.80000,6.80000,0.00000
autumn,small_,high_,8.05000,10.60000,59.06700,4.99000,205.66701,44.66700,77.43400,6.90000,18.20000,1.60000,0.00000,0.00000,5.50000,8.70000,0.00000
winter,small_,medium,8.70000,3.40000,21.95000,0.88600,102.75000,36.30000,71.00000,5.54400,25.40000,5.40000,2.50000,0.00000,0.00000,0.00000,0.00000
winter,small_,high_,7.93000,9.90000,8.00000,1.39000,5.80000,27.25000,46.60000,0.80000,17.00000,0.00000,0.00000,2.90000,0.00000,0.00000,1.70000
spring,small_,high_,7.70000,10.20000,8.00000,1.52700,21.57100,12.75000,20.75000,0.80000,16.60000,0.00000,0.00000,0.00000,1.20000,0.00000,6.00000
summer,small_,high_,7.45000,11.70000,8.69000,1.58800,18.42900,10.66700,19.00000,0.60000,32.10000,0.00000,0.00000,0.00000,0.00000,0.00000,1.50000
winter,small_,high_,7.74000,9.60000,5.00000,1.22300,27.28600,12.00000,17.00000,41.00000,43.50000,0.00000,2.10000,0.00000,1.20000,0.00000,2.10000
summer,small_,high_,7.70000,11.80000,8.30000,1.17000,8.00000,16.00000,11.00000,0.00000,31.10000,0.00000,0.00000,1.90000,0.00000,4.10000
```

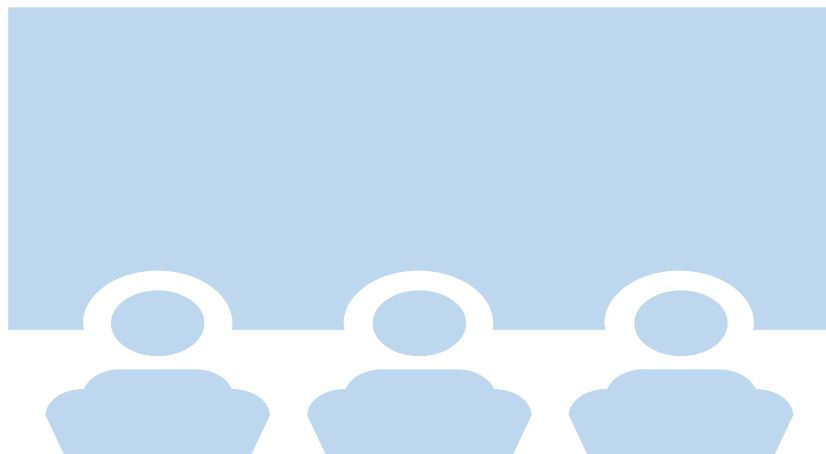

analysis.data

winter,small_,medium,8.00000,9.80000,60.80000,6.23800,578.00000,105.00000,170.00000,50.00000,0.00000,0.00000,0.00000,0.00000,34.20000,8.30000,0.00000

前11个量：季节(season)、河流大小(size)、流速(speed)、最大pH值(mxPH)、最小含氧量(mnO2)、平均氯化物含量(C1)、平均硝酸含量(NO3)、平均氨含量(NH4)、平均正磷酸盐含量(oP04)、平均磷酸盐含量(P04)、平均叶绿素含量(Ch1a)

后7个量：7种藻类频率数(a1~a7)

把数据存成CSV文件，仔细观察数据，发现了什么？请列出



思考：

- 1.有处理这种表格数据的方法？**
- 2.未知值、缺值怎么处理？**
- 3.噪声怎样处理？**

Pandas是Python的一个开源数据分析包，它建立在NumPy基础上，提供了多行多列（列数据类型可能不同）且可能缺值、含噪声的高效数据结构 and 易用分析工具。

主题：

- Pandas的数据结构
- 怎样存储、获取和更新数据？
- 怎样访问文件中的数据？
- 缺失值、异常值和重复值怎么处理？
- 怎样进行数据的分组、合并？

Series对象和DataFrame对象

创建对象、访问数据、更新数据

读写文件

数据预处理

两大核心数据结构：

- **Series**是一种类似于一维数组的对象。
它存储一组数据（类型可以不同），以及一组与之相对应的索引。
- **DataFrame**是一种多行多列的表格结构，
可存储多特征数据，既有行索引，又有列名称（也称列标签，也有称列索引）。

索引

北京奥运	2008
东京奥运	2021
巴黎奥运	2024

	num	name	age
0	1951027	张莉	19
1	1753019	李峰	20
2	1850012	童敏	20
3	2051034	吴峰	18

Pandas.Series(data, index = 索引列表)

说明:

- ① **data**: 一组数据, 支持多种数据类型, 可以是:
 - 一个Python列表(元素类型可以不同)
 - 一个NumPy数组
 - 一个Python字典, 字典的键为索引, 字典的值为data
- ② **index**: 数据对应的索引列表, 是可选参数。若data是NumPy数组或列表, 则索引长度要与该数组或列表长度一致; 若没给出index, 其默认值为整数序列, 取值0到N-1 (N为数据的长度)。


```
import pandas as pd
s1 = pd.Series(['王宏', 19], index=['姓名', '年龄'])
s1
```

元素类型不同

姓名	王宏
年龄	19
dtype: object	

```
import numpy as np
s2 = pd.Series(np.arange(3,6))
s2
```

索引自动创建

0	3
1	4
2	5
dtype: int32	

值

```
s3 = pd.Series({'北京奥运':2008, '东京奥运':2021, '巴黎奥运':2024})
s3
```

索引

北京奥运	2008
东京奥运	2021
巴黎奥运	2024
dtype: int64	

值

获取Series对象的**值数组**

对象名.values 

获取Series对象的**索引对象**

对象名.index

```
import pandas as pd
s4 = pd.Series([4,7,-5,3],index=['a','b','c','d'])
s4.values
```

s4

a	4
b	7
c	-5
d	3

输出: array([4, 7, -5, 3], dtype=int64)

```
s4.index
```

输出: Index(['a', 'b', 'c', 'd'], dtype='object')

获取Series对象的单个值

获取Series对象的一组值

对象名[索引] 索引可以是序号或 名称

对象名[切片] 或 **对象名[索引列表]**

s4

a	4
b	7
c	-5
d	3

利用序号

```
print(s4[2]) # 索引
```

```
print(s4[[0,3]]) # 索引列表
```

```
print(s4[0:3]) # 切片
```

或者

利用index名称

```
print(s4['c']) # 以键取值
```

```
print(s4['a']) # 名称索引
```

```
print(s4[['a','d']]) # 名称索引列表
```

```
print(s4['a':'d']) # 名称切片
```

-5

a	4
d	3

dtype: int64

a	4
b	7
c	-5

dtype: int64

s4[2]

s4[[0,3]]输出值和对应索引

s4[0:3]输出值和对应索引

(1) 修改值数组

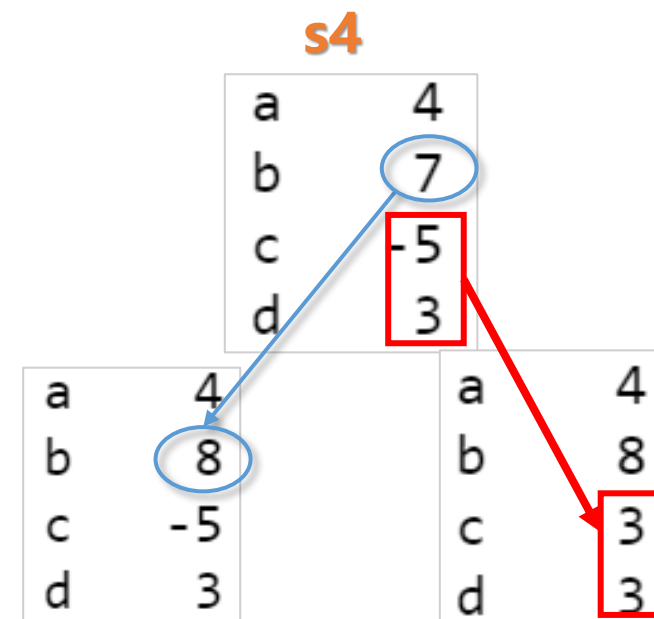
修改单个值 对象名[索引] = 新值

修改一组值 对象名[切片] = 新值

其中，索引和切片可以是序号，也可以是名称

```
s4[1]=8 # 修改一个值 或 s4['b']=8  
s4
```

```
s4[2:]=3 # 修改多个值 或 s4['c':]=3  
s4
```



(2) 修改索引对象

对象名.index = 新索引序列

```
s4.index=['Bob','Steve','Jeff','Ryan']  
s4
```

```
s4.index[0]='Ann' ❌
```

注意：不允许对Series索引对象中的元素做修改。

s4

a	4
b	7
c	-5
d	3

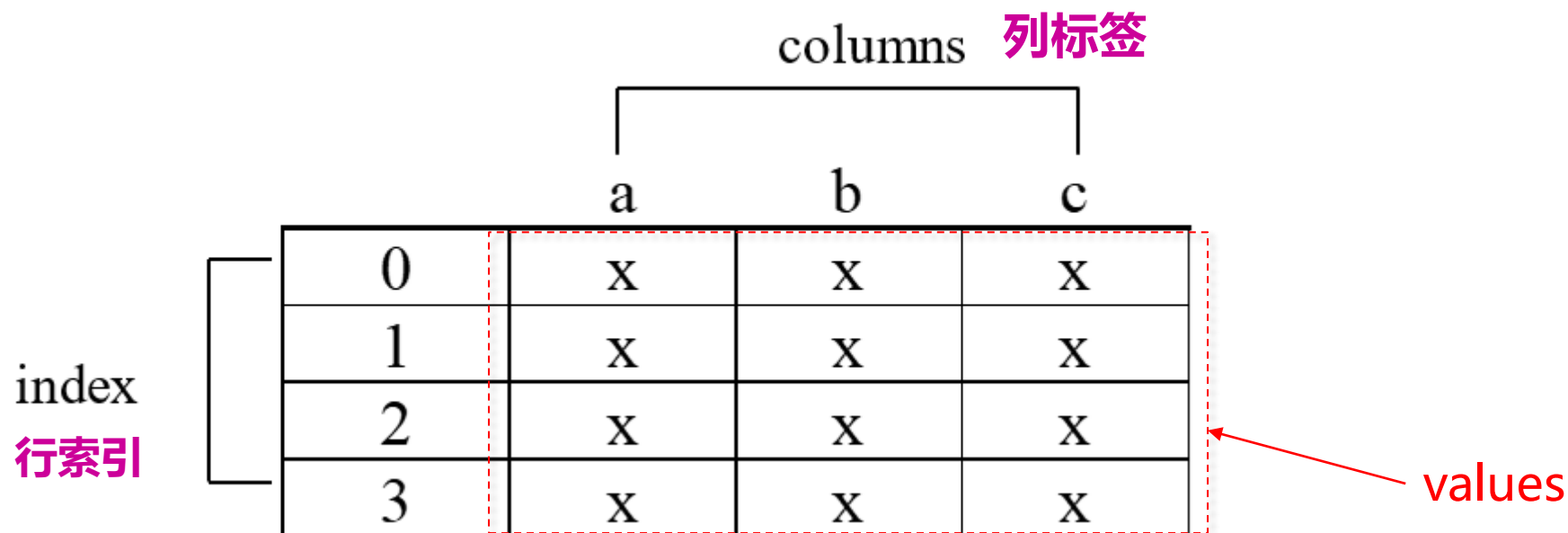
↓

s4

Bob	4
Steve	8
Jeff	3
Ryan	3

dtype: int64

- DataFrame是一种二维带标签的数据结构，各列数据类型可以不同。
- 与Series相似，也由索引和数据组成，区别是DataFrame不仅有行索引，还有列索引。列索引通常是以字符串命名的列名，称为列标签。
- DataFrame可看作Series对象组成的字典。



The diagram illustrates a DataFrame as a 2D table. The columns are labeled 'a', 'b', and 'c' at the top, with a bracket above them labeled 'columns' and '列标签' (column labels). The rows are indexed 0, 1, 2, and 3 on the left, with a bracket to their left labeled 'index' and '行索引' (row index). The table contains the value 'X' in every cell. A red dashed rectangle outlines the entire data area, with an arrow pointing to it from the label 'values' on the right.

	a	b	c
0	X	X	X
1	X	X	X
2	X	X	X
3	X	X	X

Pandas.DataFrame(**data**[, **index**=None, **columns**=None, ...])

说明:

- ① **data**: 用于创建DataFrame的数据，不可缺省。可以是
 - 等长列表或NumPy数组的字典，或者Series对象的字典。
 - 由二维列表、元组的列表、NumPy二维数组，或其他DataFrame对象。
- ② **index**: 行索引，列表类型。若没指定行索引，自动生成0~n-1（n为数据的行数）。
- ③ **columns**: 列标签的列表。若没有指定列索引，自动生成0~m-1的整数型索引，m为数据的列数。使用字典创建DataFrame对象时，默认字典的key用作列标签。

例 由等长列表的字典，来创建DataFrame对象

```
df2=pd.DataFrame({'num':['1951027','1753019','1850012','2051034'],  
                  'name':['张莉','李峰','童敏','吴峰'],  
                  'age':[19,20,20,18]})
```

df2

index没传入，
则默认0~n-1。

索引
index

df2	num	name	age
0	1951027	张莉	19
1	1753019	李峰	20
2	1850012	童敏	20
3	2051034	吴峰	18

← 列标签columns

例 由等长数组的字典，来创建DataFrame对象

```
d = {'num':np.array(['1951027','1753019','1850012','2051034']),  
      'name':np.array(['张莉','李峰','童敏','吴峰']),  
      'age':np.array([19,20,20,18])}  
df4 = pd.DataFrame(d)  
df4
```

df4	num	name	age
0	1951027	张莉	19
1	1753019	李峰	20
2	1850012	童敏	20
3	2051034	吴峰	18

例 由Series对象字典，来创建DataFrame对象

```
d = {'num':pd.Series(['1951027','1753019','1850012','2051034']),  
     'name':pd.Series(['张莉','李峰','童敏','吴峰']),  
     'age':pd.Series([19,20,20,   ])}  
df3 = pd.DataFrame(d)  
df3
```

不等长Series对象的字典
自动补齐

df3	num	name	age
0	1951027	张莉	19
1	1753019	李峰	20
2	1850012	童敏	20
3	2051034	吴峰	18

	num	name	age
0	1951027	张莉	19.0
1	1753019	李峰	20.0
2	1850012	童敏	20.0
3	2051034	吴峰	NaN

例 由二维数组创建DataFrame对象

```
np.random.seed(42)
scores = np.random.randint(60,100,(5,3))
df1 = pd.DataFrame(scores,
                    columns=['数学','人工智能','计算机视觉'])
df1
```

index没传入, 则默认0~n-1。 索引 index →

	数学	人工智能	计算机视觉
0	98	88	74
1	67	80	98
2	78	82	70
3	70	83	95
4	99	83	62

← 列标签columns

可利用DataFrame对象的属性或方法查看其信息

属性或方法	涵义
<df名>.size	查看数据框元素的总个数，返回整数
<df名>.shape	查看数据框的形状，返回元组
<df名>.index	查看数据框的 行索引 ，返回RangeIndex对象
<df名>.columns	查看数据框的 列标签 ，返回Index对象
<df名>.values	查看数据框中的 数据 ，返回二维数组
<df名>.info()	查看数据框的更多数据信息

DataFrame三要素：columns、index 和 values

df1 数学 人工智能 计算机视觉

0	98	88	74
1	67	80	98
2	78	82	70
3	70	83	95
4	99	83	62

```
print(df1.size)    # 15
```

```
print(df1.shape) # (5,3)
```

```
print(df1.columns)
```

```
Index(['数学', '人工智能', '计算机视觉'], dtype='object')
```

```
df1.values
```

```
array([[98, 88, 74],  
       [67, 80, 98],  
       [78, 82, 70],  
       [70, 83, 95],  
       [99, 83, 62]])
```

□ 选取列数据

通过列名称（即列标签）、位置或范围来指定

□ 选取行数据

通过行索引、切片、布尔索引、head()/tail()方法

□ 选取指定行和列的单元格

Pandas索引器（loc和iloc）方式

结果为DataFrame（选多列时）或Series（取单列时）对象。

若想获取某列或某几列，可把列名或列名列表放入[]中。

<df对象名>[列名]

<df对象名>[列名列表]

```
df1['人工智能']
```

```
df1[['数学', '人工智能']]
```

```
df1.人工智能 # 只适用于纯字符串列名
```

df1	数学	人工智能	计算机视觉
0	98	88	74
1	67	80	98
2	78	82	70
3	70	83	95
4	99	83	62

可通过索引器、切片和布尔索引等多种方法，选取行数据

获取行数据	说明
<df名>.head(n)	查看前n行的数据，n默认取5。
<df名>.tail(n)	查看后n行的数据，n默认取5。
<df名>.loc[indexlist, collist]	根据(名称)索引列表/切片，查询某些行、某些列的数据。省略collist，查询所有列。
<df名>.iloc[iloclist, cloclist]	根据位置序号列表/切片，查询某些行、某些列的数据。省略cloclist，查询所有列。
<df名>[start:stop:step]	利用行切片获取行子集（所有列），start、stop、step都是整数（位置序号）
<df名>[布尔索引]	利用布尔索引获取满足条件的行（所有列）。
<df名>.loc[condition, collist]	根据所构造的条件表达式，查询满足条件的数据

利用head()和tail()方法

df4	num	name	age
0	1951027	张莉	19
1	1753019	李峰	20
2	1850012	童敏	20
3	2051034	吴峰	18

`df4.head(3)`

`df4.tail(3)`

输出:

	num	name	age
0	1951027	张莉	19
1	1753019	李峰	20
2	1850012	童敏	20

	num	name	age
1	1753019	李峰	20
2	1850012	童敏	20
3	2051034	吴峰	18

通过行名获取行：利用loc属性

```
df = pd.DataFrame({'num': ['1951027', '1753019', '1850012', '2051034'],  
                  'name': ['张莉', '李峰', '童敏', '吴峰'],  
                  'age': [19, 20, 20, 18]}, index=['a', 'b', 'c', 'd'])
```

df

	num	name	age
a	1951027	张莉	19
b	1753019	李峰	20
c	1850012	童敏	20
d	2051034	吴峰	18

`df.loc['a']` # `df.loc['a', :]`

取单行所有列

```
num      1951027  
name      张莉  
age        19  
Name: a, dtype: object
```

`df.loc[['b', 'd']]`

取多行所有列

`df.loc['b': 'd']`

	num	name	age
b	1753019	李峰	20
c	1850012	童敏	20
d	2051034	吴峰	18

	num	name	age
b	1753019	李峰	20
d	2051034	吴峰	18

行名切片时，取上界

通过行号获取行：利用iloc属性

df

	num	name	age
a	1951027	张莉	19
b	1753019	李峰	20
c	1850012	童敏	20
d	2051034	吴峰	18

省略cloclist, 查询所有列。

`df.iloc[0]` 取单行

`df.iloc[[1,3]]`

`df.iloc[1:3]` 取多行

	num	name	age
b	1753019	李峰	20
c	1850012	童敏	20

```
num      1951027
name      张莉
age       19
Name: a, dtype: object
```

	num	name	age
b	1753019	李峰	20
d	2051034	吴峰	18

行号切片时, 不取上界

利用布尔索引获取满足条件的行

df

	num	name	age
a	1951027	张莉	19
b	1753019	李峰	20
c	1850012	童敏	20
d	2051034	吴峰	18

例 选取年龄20岁以下的行

```
df[df['age']<20]
```

```
# 或 df[df.age<20]
```

	num	name	age
a	1951027	张莉	19
d	2051034	吴峰	18

布尔运算符:

& 与

| 或

~ 取反

^ 异或

问题: 如何获取年龄小于均值的行?

loc和iloc属性可用于选取行、列或二者的子集

- **<对象名>.loc**[行索引列表, 列名称列表]
- **<对象名>.iloc**[行序号列表, 列序号列表]

说明:

- ① loc和iloc的语法是使用带逗号的方括号。逗号左边是待取子集的行值，右边是待取子集的列值。
- ② 行索引列表：也可以是单个索引，还可以是行切片，或者布尔索引。
- ③ 列名称列表：也可以是单列名称，还可以是列名称切片，空切片时可省略。
- ④ 行序号列表：也可以是单个行序号，还可以是行序号切片。
- ⑤ 列序号列表：也可以是单个列序号，还可以是列序号切片，空切片时可省略。

问题1: 如何获取所有学生的学号和年龄?

df

	num	name	age
a	1951027	张莉	19
b	1753019	李峰	20
c	1850012	童敏	20
d	2051034	吴峰	18

使用loc获取列子集

```
df.loc[:, ['num', 'age']]
```

使用iloc获取列子集

```
df.iloc[:, [0, 2]]
```

使用iloc获取列子集

```
df.iloc[:, ::2]
```

```
df[['num', 'age']]
```

	num	age
a	1951027	19
b	1753019	20
c	1850012	20
d	2051034	18

问题2: 如何获取年龄小于均值的学生学号和姓名?

```
df.loc[df['age'] < df['age'].mean(), ['num', 'name']]
```

df

	num	name	age
a	1951027	张莉	19
b	1753019	李峰	20
c	1850012	童敏	20
d	2051034	吴峰	18

例 获取第1行第2列的数据

```
df.loc['a', 'name'] # 使用loc
```

```
df.iloc[0,1] # 使用iloc
```

'张莉'

注意 loc和 iloc的区别



课堂练习

df

	num	name	age
--	-----	------	-----

a	1951027	张莉	19
---	---------	----	----

b	1753019	李峰	20
---	---------	----	----

c	1850012	童敏	20
---	---------	----	----

d	2051034	吴峰	18
---	---------	----	----

1. 分别用loc和iloc，写出获取第1行、最后1行，第1列、最后1列数据的语句。

```
df.loc[['a','d'],['num','age']] #使用loc
```

```
df.iloc[[0,-1],[0,-1]] #使用iloc
```

	num	age
--	-----	-----

a	1951027	19
---	---------	----

d	2051034	18
---	---------	----

2. 分别用loc和iloc，写出获取第2、3行，前2列数据的语句。

```
df.iloc[1:3,0:2] # 等价于下面
```

```
df.loc['b':'c','num':'name']
```

行名切片时，取上界

	num	name
b	1753019	李峰
c	1850012	童敏

DataFrame数据的更新

- (1) 添加、修改、删除列
- (2) 修改、删除行
- (3) 修改个别数据

添加列：对**不存在的列**赋值，将创建新列

关于赋值：

- ① 将列表或数组赋给某列时，其长度须跟数据框的长度一致，否则报错。
- ② 若将一个Series赋给某列，将自动补齐，所有空位都将被填上**缺失值NaN**。

```
df['birthday']=['2000-1-1','2001-12-1','2002-10-9','2003-4-5']
```

	num	name	age	birthday
a	1951027	张莉	19	2000-1-1
b	1753019	李峰	20	2001-12-1
c	1850012	童敏	20	2002-10-9
d	2051034	吴峰	18	2003-4-5

```
print(df['birthday'].dtype)
```

输出： object

修改列：对**现有列**赋新值，将修改该列。

df	num	name	age	birthday
a	1951027	张莉	19	2000-1-1
b	1753019	李峰	20	2001-12-1
c	1850012	童敏	20	2002-10-9
d	2051034	吴峰	18	2003-4-5

```
df['birthday']=pd.to_datetime(df['birthday'])  
print(df['birthday'])
```

```
a    2000-01-01  
b    2001-12-01  
c    2002-10-09  
d    2003-04-05  
Name: birthday, dtype: datetime64[ns]
```

删除列 `<df名>.drop(columns=[列名1,列名2,...], inplace=False)`

说明：参数 `inplace` 为 `True` 时，删除作用于原对象；否则返回一个新对象，原对象不变。

```
print(df.columns)
```

输出： `Index(['num', 'name', 'age', 'birthday'], dtype='object')`

```
X = df.drop(columns=['age'])
```

```
print(df.columns, '\n', X.columns)
```

输出： `Index(['num', 'name', 'age', 'birthday'], dtype='object')`
 `Index(['num', 'name', 'birthday'], dtype='object')`

删除行 `<df名>.drop(index = [行索引1,行索引2,...],inplace=False)`

参数inplace为True时，删除作用于原对象；否则返回一个新对象，原对象不变。行也称为记录。

X	num	name	birthday
a	1951027	张莉	2000-01-01
b	1753019	李峰	2001-12-01
c	1850012	童敏	2002-10-09
d	2051034	吴峰	2003-04-05

```
X.drop(index = ['d'],inplace=True)  
X
```

X	num	name	birthday
a	1951027	张莉	2000-01-01
b	1753019	李峰	2001-12-01
c	1850012	童敏	2002-10-09

修改行 `<df名>[n:n+1]=新数据列表` 修改行号为n的行，用新数据替换原数据。

X

	num	name	birthday
a	1951027	张莉	2000-01-01
b	1753019	李峰	2001-12-01
c	1850012	童敏	2002-10-09

`X[2:3]= ['1850012', 'XXXXXX', 'XXXXXX']`

X

	num	name	birthday
a	1951027	张莉	2000-01-01 00:00:00
b	1753019	李峰	2001-12-01 00:00:00
c	1850012	XXXXXX	XXXXXX

替换 `<df名>.replace(A,B, inplace=False)` 数据框中所有A都用B替换。
`<df名>.replace({列标签:A},B, inplace=False)` 指定列中所有A都用B替换。

	num	name	birthday
a	1951027	张莉	2000-01-01 00:00:00
b	1753019	李峰	2001-12-01 00:00:00
c	1850012	XXXXXX	XXXXXX

	num	name	birthday
a	1951027	张莉	2000-01-01 00:00:00
b	1753019	李峰	2001-12-01 00:00:00
c	1850012	NaN	0000-1-1

```
X.replace({'name': 'XXXXXX'}, np.nan, inplace=True)  
X.replace({'birthday': 'XXXXXX'}, '0000-1-1', inplace=True)
```

```
# 若将数据框中的'XXXXXX'都替换为np.nan  
newx = X.replace('XXXXXX', np.nan, inplace=True)
```

方法	涵义
<对象名>.sum()	计算和
<对象名>.mean()	计算均值
<对象名>.max()	获取最大值
<对象名>.min()	获取最小值
<对象名>.idxmax()	获取最大索引值
<对象名>.idxmin()	获取最小索引值
<对象名>.count()	计算非NaN值的个数
<对象名>.var()	样本值的方差
<对象名>.std()	样本值的标准差

一般针对某一列进行统计

```
df = pd.DataFrame({'num': ['1951027', '1753019', '1850012', '2051034'],  
                  'name': ['张莉', '李峰', '童敏', '吴峰'],  
                  'age': [19, 20, 20, 18]}, index=['a', 'b', 'c', 'd'])  
df['age'].mean() # 对age列求均值, 结果为19.25
```



课堂练习

df4如下所示, 用age列现有值的均值替代其中的NaN, 写出语句。

	num	name	age
0	1951027	张莉	19.0
1	1753019	李峰	20.0
2	1850012	童敏	20.0
3	2051034	吴峰	NaN

```
Avg = df4['age'].mean(skipna=True)
df4.replace(np.nan, Avg, inplace=True)
```

```
df4.replace(np.nan, df4['age'].mean(skipna=True), inplace=True)
```

写入csv文件

<df名>.to_csv(文件路径, sep=',', index=True, header=True)

参数:

- 文件路径, 必填。若该文件不存在, 会创建; 若已存在, 则会被覆盖;
- sep: 指定分隔符, 默认 ',' ;
- index: 是否导出行号 (或行名), 默认是True, 即导出index;
- header: 是否导出列名称, 默认是True, 即导出列名称。

```
df = pd.DataFrame({'num': ['1951027', '1753019', '1850012', '2051034'],  
                  'name': ['张莉', '李峰', '童敏', '吴峰'],  
                  'age': [19, 20, 20, 18]}, index=['a', 'b', 'c', 'd'])  
df.to_csv('student.csv', index=False, encoding='utf-8_sig')
```

num	name	age
1951027	张莉	19
1753019	李峰	20
1850012	童敏	20
2051034	吴峰	18

防止存入excel文件后中文乱码

读取csv文件

```
pd.read_csv(文件路径, sep=',', header='infer', names=None, index_col=None)
```

读取指定路径的csv文件(以逗号为分隔符), 返回一个dataframe对象。

参数:

- 文件路径: 必填, 可以是URL, 可用URL类型包括: http, ftp, s3和文件
- sep: 为分隔符, 默认为逗号。
- header: 用文件中指定行号的内容作为列名称。默认为'infer', 意思是从文件的**第一行**推测出列名。
- names: 如果文件不包含各列名称, 应显式地用**names指定列名称列表**, 并设 header=None。
- index_col: 用作行索引的列。默认为 None, 用0~N-1

```
file_data = pd.read_csv('student.csv')  
print(file_data)
```

例2.5 已知score.csv文件。该文件中每行有4列，第1列是学号，第2列~第4列分别是各题得分。先读取文件，根据各题得分计算每个考生的总分。然后将总分列和学号列合并，存入score1.csv中。

考号	选择题	填空题	设计题
10153450101	21.6	19.2	44
10153450102	20.4	17.6	36.5
10153450103	18.9	12.6	33
10153450104	28.5	12.8	32.5
10153450105	26.7	17.6	28.5
10153450106	23.1	17.4	38.5
10153450107	20.1	12.8	48.5
10153450108	13.2	19.8	32
10153450109	24.6	14.6	37.5
10153450110	23.7	15.6	42.5
10153450111	16.8	9	17
10153450112	20.1	14	28
10153450113	26.7	18	48
10153450114	22.2	13.4	39
10153450115	27	17.4	48
10153450116	24.6	15.8	39
10153450117	16.8	9	28
10153450118	22.8	16	34.5
10153450119	24.3	17.8	45
10153450120	18.9	13.4	35

```
import pandas as pd
import numpy as np
data=pd.read_csv(r'score.csv')
X=data.iloc[:,1:]
y=data.iloc[:,0]
```

X

	选择题	填空题	设计题
0	21.6	19.2	44.0
1	20.4	17.6	36.5
2	18.9	12.6	33.0
3	28.5	12.8	32.5
4	26.7	17.6	28.5
5	23.1	17.4	38.5
6	20.1	12.8	48.5
7	13.2	18.8	32.0
8	24.1	14.6	37.5

y

0	10153450101
1	10153450102
2	10153450103
3	10153450104
4	10153450105
5	10153450106
6	10153450107
7	10153450108
8	10153450109
9	10153450110
10	10153450111

```
total = pd.Series(np.sum(X.values,axis=1))
X_new = pd.DataFrame({'学号':y,'总分':total})
X_new.to_csv(r'score1.csv',index=False,
             header=True,encoding='utf-8_sig')
```

防止中文乱码

	A	B
1	学号	总分
2	10153450101	84.8
3	10153450102	74.5
4	10153450103	64.5
5	10153450104	73.8
6	10153450105	72.8
7	10153450106	79
8	10153450107	81.4
9	10153450108	65
10	10153450109	76.7



作业 2.1

考试成绩预处理

考试成绩文件score_test.csv有20行。每行有4列，第1列是学号，第2列~第4列分别是该生各题得分。有两处空缺值。

要求：

读取文件，缺失值用所在列的均值填充。然后计算每个考生的总分。最后将总分转换为三个档次的成绩：“优秀（ ≥ 85 ）、合格

（ ≥ 60 ）、不合格（ < 60 ）”，以“总评”作为列名将三档成绩加入原数据框对象中，将该对象存入score2.csv中，结果如右表所示。

score2.csv

考号	选择题	填空题	设计题	总评
10153450101	21.6	19.2	44	合格
10153450102	20.4	17.6	36.5	合格
10153450103	22.21579	12.6	33	合格
10153450104	28.5	12.8	32.5	合格
10153450105	26.7	17.6	28.5	合格
10153450106	23.1	17.4	38.5	合格
10153450107	20.1	12.8	48.5	合格
10153450108	13.2	19.8	32	合格
10153450109	24.6	14.6	37.5	合格
10153450110	23.7	15.6	42.5	合格
10153450111	16.8	15.51579	17	不合格
10153450112	20.1	14	28	合格
10153450113	26.7	18	48	优秀
10153450114	22.2	13.4	39	合格
10153450115	27	17.4	48	优秀
10153450116	24.6	15.8	39	合格
10153450117	16.8	9	28	不合格
10153450118	22.8	16	34.5	合格
10153450119	24.3	17.8	45	优秀
10153450120	18.9	13.4	35	合格