

同济大学
TONGJI UNIVERSITY

王睿智

ruizhiwang@tongji.edu.cn

人工智能技术与应用

(1) 产生0~100之间的随机整数数组, 形状为(3,4), 写出语句。

```
a = np.random.randint(0,101,(3,4))
```

(2) 求该数组的均值, 并将数组中<20或>95的元素值用均值替换。

```
a[(a<20) | (a>95)] = np.mean(a)
```

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。



把数据存成CSV文件，仔细观察数据，发现了什么？请列出

- 数据是多行、多列的表格数据，每列有各自的数据类型。
- 数据中有“XXXXXXXX”，表示未知值。未知值会对后期分析产生影响。
- NO3列有的数据带两个小数点，这是噪声数据。
- 最后一列（a7）有空白。
- 有个别行存在多列未知值。



analysis.data

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？

- 数据预处理是数据分析过程的关键环节，它能提高数据的质量。
- 预处理主要包括：
 - **数据清洗** 去除重复值、缺失值处理
 - 数据替换 整体替换、个别修改（已学）
 - **数据合并** 记录(行)合并、字段(列)合并
 - **数据转换** 独热编码、数据标准化等
 - **数据抽取** 抽取记录（行）、抽取字段（列）、随机抽取

数据清洗：

- 是整个数据分析过程的关键环节。
- 目的是提高数据质量，将脏数据（指与数据分析任务无关的数据、格式非法的数据、不在指定范围内的数据）清洗干净，使原数据具有完整性、唯一性、合法性、一致性等特点。

Pandas中常见的数据清洗操作：

- 重复值处理
- 缺失值处理
- 异常值处理
- 统一数据格式

。 。 。 。 。 。

■ 重复值处理

重复值是指表中出现两行或多行数据完全相同的数据。
如右图，name张帆的行出现两次。

Pandas提供`uplicated()`和`drop_duplicates()`用于去重。
两方法配合使用：

首先，用`uplicated()` 检查是否有重复行。
然后，用`drop_duplicates()` 移除重复行。

	age	name
0	20	张帆
1	19	王磊
2	18	邓敏
3	18	刘佳
4	20	张帆

检查重复行 `<df名>.duplicated(subset=None,keep='first')`

删去重复行 `<df名>.drop_duplicates(subset=None,inplace=False)`

参数:

- keep: 默认为first, 从前向后查找, 除第一次出现外, 其余相同的被标记为重复。返回一个布尔型的Series对象, 重复则标记为True, 不重复则标记为False。
- subset: 待识别的列标签或列标签序列, 默认识别所有的列标签。



课堂实例

例 重复值处理

	age	name
0	20	张帆
1	19	王磊
2	18	邓敏
3	18	刘佳
4	20	张帆

```
import pandas as pd
student = pd.DataFrame({'age':[20,19,18,18,20],
                        'name':['张帆','王磊','邓敏','刘佳','张帆']})
```

```
student.duplicated() # 检查重复行
```

```
0    False
1    False
2    False
3    False
4     True
dtype: bool
```

```
newst=student.drop_duplicates() # 返回去重结果, student不变
newst
```

`student.drop_duplicates('age')` 结果?

	age	name
0	20	张帆
1	19	王磊
2	18	邓敏
3	18	刘佳

■ 缺失值处理

- **缺失值**：未知、不确定或将在以后添加的**空缺数据**。

- **Pandas中使用NaN表示缺失值**

NaN 来自NumPy库，

NumPy中，缺失值有三种表示：NaN、NAN、nan，三者是等同的

- **缺失值来源有两个：**

一是包含缺失值的数据集；

二是数据整理过程，如合并数据，用户输入数据



课堂实例

例 读入包含缺失值的数据文件

```
import pandas as pd  
data1 = pd.read_csv('student2.csv')  
data1
```

	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	18.0
4	2101001	NaN	NaN

student2.csv

num	name	age
1951027	张莉	19
1753019	李峰	20
1850012	童敏	20
2051034	吴峰	18
2101001		

自动将其编码为缺失值



课堂实例

例 读入包含缺失值的数据文件

```
data3 = pd.read_csv('student3.csv',  
                    na_values=['????'],  
                    keep_default_na = False)
```

手动指定缺失值

data3

num	name	age
1951027	张莉	19
1753019	李峰	20
1850012	童敏	20
2051034	吴峰	18
2101001	????	????

student3.csv

	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	18.0
4	2101001	NaN	NaN



课堂实例

例 用户输入包含缺失值

```
from numpy import NaN  
df1 = pd.DataFrame({'num': ['1951027', '1753019', '1850012', '2051034'],  
                    'name': ['张莉', '李峰', NaN, '吴峰'],  
                    'address': ['嘉定', '沪西', NaN, NaN],  
                    'age': [19, NaN, 20, NaN]})
```

df1

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

■ 缺失值的方法:

- 发现缺失值: **isnull()**和**notnull()**

用于检查DataFrame对象中**是否**有缺失值

- 处理缺失值: **fillna()**和**dropna()**

用于**填充**和**删除**数据中的缺失值

- 发现和统计缺失值

Pandas 提供了检查缺失值常用方法。表中对象是指DataFrame或Series对象

方法	涵义
<对象名>.isnull()	检查对象中是否存在NaN，一旦发现则将这个位置标记为True，否则为False。返回一个布尔型DataFrame对象或Series对象。
<对象名>.isnull().any()	检查对象的各个列是否存在NaN元素，存在的列返回True，否则为False。返回一个布尔型Series对象或布尔值。
<对象名>.isnull().sum()	统计各列缺失值个数



课堂实例

例 发现和统计缺失值

```
ebola = pd.read_csv('country_timeseries.csv')  
print(ebola.columns)
```

```
Index(['Date', 'Day', 'Cases_Guinea', 'Cases_Liberia', 'Cases_SierraLeone',  
      'Cases_Nigeria', 'Cases_Senegal', 'Cases_UnitedStates', 'Cases_Spain',  
      'Cases_Mali', 'Deaths_Guinea', 'Deaths_Liberia', 'Deaths_SierraLeone',  
      'Deaths_Nigeria', 'Deaths_Senegal', 'Deaths_UnitedStates',  
      'Deaths_Spain', 'Deaths_Mali'],  
      dtype='object')
```

```
ebola.isnull() # 122行18列
```

```
ebola.isnull().any() # 查看哪些列有缺失值
```

```
ebola.isnull().sum() # 统计各列缺失值个数
```

```
ebola['Cases_Spain'].isnull().any() # 查看'Cases_Spain'有无缺失值
```

Date	False
Day	False
Cases_Guinea	True
Cases_Liberia	True
Cases_SierraLeone	True
Cases_Nigeria	True
Cases_Senegal	True
Cases_UnitedStates	True
Cases_Spain	True
Cases_Mali	True
Deaths_Guinea	True
Deaths_Liberia	True
Deaths_SierraLeone	True
Deaths_Nigeria	True
Deaths_Senegal	True
Deaths_UnitedStates	True
Deaths_Spain	True
Deaths_Mali	True

- 清理缺失数据: `fillna()`、`dropna()`

方法

涵义

<对象名>.fillna(value=None,method=None,axis=None,inplace=False) 把缺失值重新编码为其他值

- ① value: 用于填充的数值。可以是标量、字典等。
- ② method: 表示填充方式，默认为None。可取值：
 - ffill: 用缺失值前面的一个值替代缺失值。
 - bfill: 用缺失值后面的一个值替代缺失值。
- ③ axis: 指定填充行或列，0指定行,1指定列。默认0。

注意:
method参数不能与value参数同时使用。



课堂实例

例 填充缺失值

```
df1 = pd.DataFrame({'num': ['1951027', '1753019', '1850012', '2051034'],  
                    'name': ['张莉', '李峰', None, '吴峰'],  
                    'address': ['嘉定', '沪西', None, None],  
                    'age': [19, NaN, 20, NaN]})
```

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

(1) 对address列，用缺值前一行的有效值填充

```
df1['address'].fillna(method='ffill', axis=0, inplace=True)  
df1
```

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

如果一列以缺失值开始，ffill会出现什么情况？

如果一列以缺失值结束，bfill会出现什么情况？



课堂实例

例 填充缺失值

(2) 对age列的缺失值，用该列现有值的均值来填充

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

```
df1.fillna({'age':df1['age'].mean(skipna=True)},inplace=True)  
df1
```

	num	name	address	age
0	1951027	张莉	嘉定	19.0
1	1753019	李峰	沪西	19.5
2	1850012	NaN	沪西	20.0
3	2051034	吴峰	沪西	19.5

- 清理缺失数据

方法

涵义

`<对象名>.dropna(axis=0, how='any', thresh=None, inplace=False)`

删除缺失值

- ① axis: 确定过滤行还是列; 0指行, 1指列。默认为0。
- ② how: 确定过滤的标准。取值为'all', 表示行或列数据全为NaN才删除该行或列; 取值'any', 若存在NaN值则删除该行或列, 默认为'any'。
- ③ thresh: 指定行或列中非NaN个数的阈值, 小于此阈值则删除。



课堂实例

例 删除缺失值

```
df1 = pd.DataFrame(
    {'num': ['1951027', '1753019', '1850012', '2051034', NaN],
     'name': ['张莉', '李峰', NaN, '吴峰', NaN],
     'address': ['嘉定', '沪西', NaN, NaN, NaN],
     'age': [19, NaN, 20, NaN, NaN],
     'birthday': [NaN, NaN, NaN, NaN, NaN]})
```

	num	name	address	age	birthday
0	1951027	张莉	嘉定	19.0	NaN
1	1753019	李峰	沪西	NaN	NaN
2	1850012	NaN	NaN	20.0	NaN
3	2051034	吴峰	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN

(1) 删除全NaN行和列

```
df1.dropna(how='all', inplace=True) # 删除行
df1
```

```
df1.dropna(how='all', axis=1, inplace=True)
df1
```

	num	name	address	age	birthday
0	1951027	张莉	嘉定	19.0	NaN
1	1753019	李峰	沪西	NaN	NaN
2	1850012	NaN	NaN	20.0	NaN
3	2051034	吴峰	NaN	NaN	NaN

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



课堂实例

例 删除缺失值

(2) 删除非NaN个数少于3的行

```
df2 = df1.dropna(axis=0, thresh=3)  
df2
```

	num	name	address	age
0	1951027	张莉	嘉定	19.0
1	1753019	李峰	沪西	NaN

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

(3) name列有缺失值，若该列与后期分析无关，也可删去

```
df4 = df2.drop(columns='name')  
df4
```

	num	address	age
0	1951027	嘉定	19.0
1	1753019	沪西	NaN

- **异常值**，是指样本中的个别值，其值明显偏离它（或他们）所属样本的其余观测值。这些值是不合理的，应予以剔除。
- **检测方法**：3 σ 原则、箱线图法、Z-score法等。
 - **3 σ 原则**：假定数据服从正态分布，99.7%数据集中在 $(\mu-3\sigma, \mu+3\sigma)$ ，超出这个范围的值视为异常值，应予以剔除。
 - **箱线图法**：异常值被定义为小于 $QL-1.5*IQR$ 或大于 $QU+1.5*IQR$ 的值，即箱型图上的离散点。（QL：下四分位数。QU：上四分位数。IQR：四分位数间距， $IQR=QU-QL$ ）
 - **Z-score法**：假定数据服从高斯分布，异常值是分布在尾部的数据点，其归一化 $z_i = (x_i - \mu)/\sigma$ 后，满足 $|z_i| > z_{thr}$ 。阈值 z_{thr} 一般设置为2.5、3.0或3.5。

例2.6 基于 3σ 原则检测异常值

自定义一个基于 3σ 原则的函数，来检查一组数据中是否存在异常值。

	math	english	computer
0	78	89	65
1	67	87	209
2	89	91	87
3	56	67	76
4	91	77	56
5	89	80	70
6	67	65	67
7	56	78	65
8	89	90	87
9	78	67	45
10	56	89	56
11	86	190	78

```
import numpy as np
import pandas as pd
def three_sigma(ser1): # ser1 表示传入DataFrame的某一列
    mean_value=ser1.mean() # 求平均值
    std_value = ser1.std() # 求标准差
    # ser1中的数值小于 $\mu-3\sigma$ 或大于 $\mu+3\sigma$ 均为异常值。发现异常值则标注为True，否则标注为False
    rule = ((mean_value-3*std_value)>ser1)|((ser1.mean() + 3*ser1.std())<ser1)
    idx = np.arange(ser1.shape[0])[rule] # 获取异常值的位置索引
    outlier = ser1.iloc[idx] # 获取异常值数据
    print(outlier.name,outlier.values)
    return outlier
```

```
df = pd.read_csv('outlier.csv',index_col=0)
```

```
for col in df.columns:
    three_sigma(df[col])
```

`df.apply(three_sigma,axis=0)`



pandas DataFrame/Series的apply方法

<df对象名>.apply(函数名,axis=0, **kwargs)

把函数同时“应用于” DataFrame的每一列(整个列) 或每一行(整个行)。

axis=0, 按列应用函数; axis=1, 按行应用函数。axis默认为0。

**kwargs 是函数的关键字参数。

```
for col in df.columns:  
    <某函数名>(df[col],该函数关键字参数)
```

它类似于一个跨每列或每行的for循环，并同时调用apply中的函数。

↓

```
df.apply(<某函数名>,该函数关键字参数,axis=0)
```

注意： 当把函数传递给apply时，不要在函数后面加上圆括号。

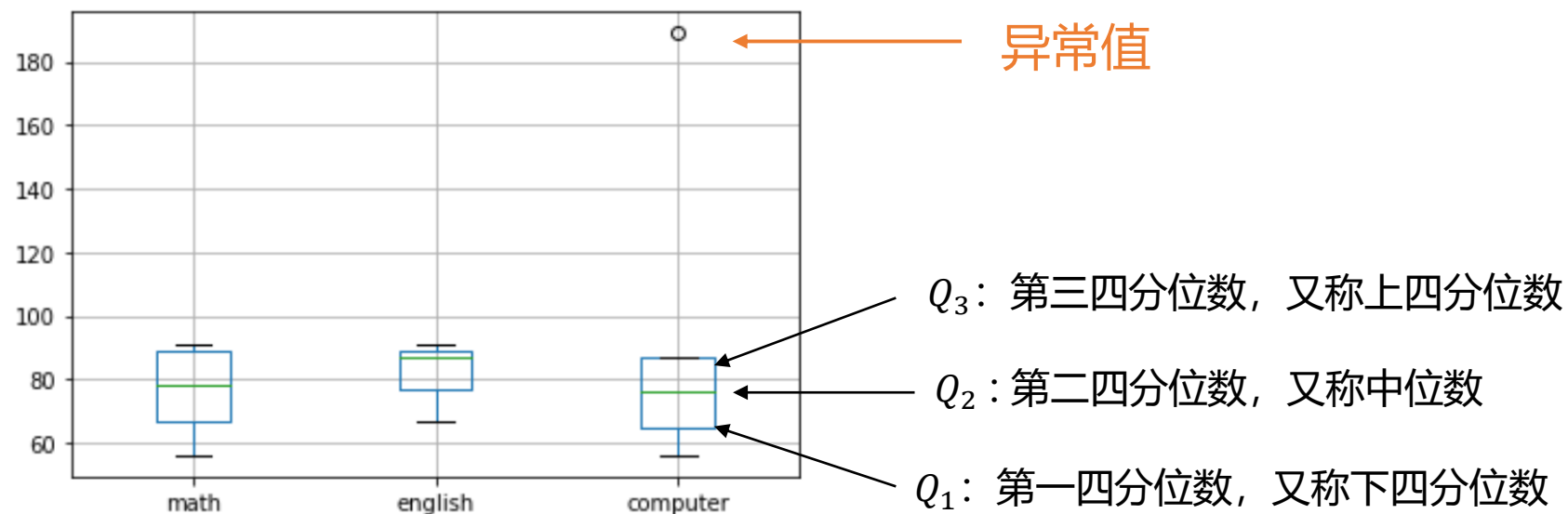


课堂实例

例 基于箱型图的异常值检测

箱线图是一种显示一组数据分散情况的统计图。Pandas中提供boxplot()方法，用于绘制箱线图。<数据框对象>.boxplot(column=None)

```
import pandas as pd
df = pd.DataFrame({'math':[78,67,89,56,91],
                  'english':[89,87,91,67,77],
                  'computer':[65,189,87,76,56]})
df.boxplot(column=['math','english','computer'])
```



检出异常值后，常采用四种方法处理：

- (1) 用指定的值替换异常值。
- (2) 直接删除含有异常值的记录。
- (3) 不处理，直接在有异常值的数据上进行后续分析。
- (4) 视为缺失值，采用缺失值处理方法处理。

用replace()方法替换。格式：

```
<对象名>.replace(to_replace=None,value=None,  
                  inplace=False,method='pad')
```

参数：

- ① to_replace: 待替换的数据。取值为数值、字符串等。若是“数值”，则对象中所有值等于to_replace的数据将被value参数值所替换；若取“字符串”，则与to_replace相匹配的数据将被value参数值所替换。
- ② value: 用来替换任何匹配to_replace的值，默认值None。
- ③ method: 替换时使用的方法。pad/ffill表示向前填充（按索引顺序，用其前的值填充），bfill表示向后填充（按索引顺序，用其后的值填充）。

注意：method参数不能与value参数同时使用。



课堂实例

例 异常值处理

```
import pandas as pd
df = pd.DataFrame({'math': [78, 67, 89, 56, 91],
                   'english': [89, 87, 91, 67, 77],
                   'computer': [65, 189, 87, 76, 56]})
```

```
df['computer'].replace(to_replace=189, value=89, inplace=True)
df
```

	math	english	computer
0	78	89	65
1	67	87	89
2	89	91	87
3	56	67	76
4	91	77	56



课堂实例

例 噪声数据处理

```
import pandas as pd
from numpy import NaN
df = pd.DataFrame({'num': ['1951027', '1753019', '1850012', '2051034'],
                   'name': ['张莉', '李峰', '???', '吴峰'],
                   'address': ['嘉定', '沪西', '???', '???'],
                   'age': [19, '???', 20, '???']})
```

```
#将age列中出现的'???'替换为NaN
df['age'].replace('???',NaN,inplace=True)
df
```

	num	name	address	age
0	1951027	张莉	嘉定	19.0
1	1753019	李峰	沪西	NaN
2	1850012	???	???	20.0
3	2051034	吴峰	???	NaN

```
# 将其他'???'替换为NaN
df.replace('???',NaN,inplace=True)
df
```

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

■ 更改数据类型

数据处理时，可能出现数据类型不一致的情况。如

```
df1 = pd.DataFrame({'num': ['1951027', '1753019', '1850012', '2051034'],  
                    'name': ['张莉', '李峰', '童敏', '吴峰'],  
                    'address': ['嘉定', '沪西', '四平', '四平'],  
                    'age': [19, 18, NaN, '19']})
```

age列，用该列现有值的均值来填充缺失值

```
df1.fillna({'age': df1['age'].mean(skipna=True)}, inplace=True)
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

报错!!!

```
type(df1['age'].dtype) # 查看age列中数据类型
```

```
numpy.dtype[object_]
```


用`astype()`，将Pandas的object类型数据强制转换为指定类型

格式：`<对象名>.astype(dtype)`

说明：

① 参数dtype，表示要转换到的数据类型。常用的有：

int64 (整型)、

float64 (带小数的数字)、

object (对应Python的字符串)

② 返回一个转换后的新对象，原对象不变。

	num	name	address	age
0	1951027	张莉	嘉定	19
1	1753019	李峰	沪西	18
2	1850012	童敏	四平	NaN
3	2051034	吴峰	四平	19

NaN是浮点数。

	num	name	address	age
0	1951027	张莉	嘉定	19
1	1753019	李峰	沪西	18
2	1850012	童敏	四平	18.666667
3	2051034	吴峰	四平	19

	num	name	address	age
0	1951027	张莉	嘉定	19
1	1753019	李峰	沪西	18
2	1850012	童敏	四平	18
3	2051034	吴峰	四平	19

age数据类型强转为float, 然后用该列现有值的均值来填充缺失值

```
df1.fillna({'age':df1['age'].astype('float').mean(skipna=True)},inplace=True)
df1
```

age数据类型强转为int

```
df1['age']=df1['age'].astype(dtype='int')
df1
```

例 已知两个不同的数据框df1和df2，可否合并在一起构成df3？

df1	A	B
1	a1	b1
2	a2	b2
3	a3	b3

df2	A	B
4	a4	b4
5	a5	b5

合并

df3	A	B
1	a1	b1
2	a2	b2
3	a3	b3
4	a4	b4
5	a5	b5

通过Pandas 提供的
concat可以实现！

Pandas 提供了多种合并数据集的方法，包括：

1. 通过**concat()函数**，简单地堆叠数据
2. 通过merge()方法，根据主键合并数据
3. 通过join()方法，通过行索引合并数据

格式: `pd.concat(objs, axis=0, join='outer')`

`pd.concat()`函数沿着一个轴将多个对象进行堆叠（这与`np.concatenate`类似），与`np.concatenate`不同的是，`pd.concat()`在合并时会保留索引，即使出现重复。

默认：按0轴堆叠、采用外连接方式合并数据。

说明:

- ① `objs`: 待合并的多个对象构成列表。
- ② `axis`: 表示连接的轴向。1表示沿1轴堆叠，0表示沿0轴堆叠，默认为0。
- ③ `join`: 表示连接方式。`inner`表示内连接合并（对所有输入列交集合并），`outer`表示外连接合并（对所有输入列并集合并，数据缺失的位置用NaN补齐），默认为`outer`。



课堂实例

例 纵向堆叠

```
df1 = pd.DataFrame({'A': ['a1', 'a2', 'a3'],  
                    'B': ['b1', 'b2', 'b3']}, index=[1, 2, 3])  
df2 = pd.DataFrame({'A': ['a4', 'a5'],  
                    'B': ['b4', 'b5']}, index=[4, 5])
```

df1

	A	B
1	a1	b1
2	a2	b2
3	a3	b3

df2

	A	B
4	a4	b4
5	a5	b5

```
df3 = pd.concat([df1, df2])
```

df3

df3

	A	B
1	a1	b1
2	a2	b2
3	a3	b3
4	a4	b4
5	a5	b5

列名相同，直接合并即可



课堂实例

例 纵向堆叠与内连接

```
df1 = pd.DataFrame({'A': ['a1', 'a2', 'a3'],  
                    'B': ['b1', 'b2', 'b3'],  
                    'C': ['c1', 'c2', 'c3']}, index=[1, 2, 3])
```

df1

```
df2 = pd.DataFrame({'B': ['b4', 'a5'],  
                    'C': ['c4', 'c5'],  
                    'D': ['d4', 'd5']}, index=[4, 5])
```

df2

```
df3 = pd.concat([df1, df2], join='inner', axis=0)
```

df3

df1	A	B	C
1	a1	b1	c1
2	a2	b2	c2
3	a3	b3	c3

df2	B	C	D
4	b4	c4	d4
5	a5	c5	d5

df3	B	C
1	b1	c1
2	b2	c2
3	b3	c3
4	b4	c4
5	a5	c5

列名不完全相同时，设置join参数来解决



课堂实例

例 纵向堆叠与外连接

df1	A	B	C
1	a1	b1	c1
2	a2	b2	c2
3	a3	b3	c3

df2	B	C	D
4	b4	c4	d4
5	a5	c5	d5

```
df4 = pd.concat([df1,df2],join='outer',axis=0)  
df4
```

df4	A	B	C	D
1	a1	b1	c1	NaN
2	a2	b2	c2	NaN
3	a3	b3	c3	NaN
4	NaN	b4	c4	d4
5	NaN	a5	c5	d5



课堂实例

例 横向堆叠与外连接

```
df1 = pd.DataFrame({'A': ['a1', 'a2', 'a3'],  
                    'B': ['b1', 'b2', 'b3']}, index=[1, 2, 3])
```

df1

df1

	A	B
1	a1	b1
2	a2	b2
3	a3	b3

```
df2 = pd.DataFrame({'C': ['c1', 'c2', 'c3', 'c4'],  
                    'D': ['d1', 'd2', 'd3', 'd4']},  
                    index=[1, 2, 3, 4])
```

df2

df2

	C	D
1	c1	d1
2	c2	d2
3	c3	d3
4	c4	d4

```
df3 = pd.concat([df1, df2], join='outer', axis=1)
```

df3

df3

	A	B	C	D
1	a1	b1	c1	d1
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	NaN	NaN	c4	d4

场景1：海藻数据analysis.dat中，存在这样的列，他们的取值反映了变量的不同类别，

如 season (spring,summer,autumn,winter) ,

size (small,medium,large)

speed (low, medium, high)



analysis.data


- 这种取值为类别的数据，常称为**分类变量**（也称**分类特征**，或称**类别特征**）。
- 分类变量的取值是一组固定值，这些值之间没有大小关系，不能直接用于模型学习，需要进行**数据转换**。
- 类别型数据，可采用**one-hot编码**（也称**k取一编码**）进行编码转换。

■ one-hot编码

思想：使用二进制特征表示，来解释分类变量的所有可能取值。

即，将一个分类变量替换为一个或多个新特征，新特征值为0和1，使得原分类变量的每个取值对应一个新的二进制特征，新特征有时被称为**哑变量**。

如，上海各区房价数据

	neighborhood	price	rooms		price	rooms	neighborhood_嘉定	neighborhood_宝山	neighborhood_杨浦	neighborhood_静安	
0	杨浦	850000	1		0	850000	1	0	0	1	0
1	宝山	3000000	2		1	3000000	2	0	1	0	0
2	嘉定	2600000	2		2	2600000	2	1	0	0	0
3	静安	10000000	3		3	10000000	3	0	0	0	1

Pandas的`get_dummies()`函数可对类别特征进行one-hot编码。

`pd.get_dummies(data, prefix=None, dummy_na=False, columns=None, drop_first=False,...)` 将类别特征转换为哑变量

参数:

- ① *data* : 表示待转换的数据。类型为DataFrame, 或array, Series对象。
- ② *prefix*: 表示新列名的前缀, 默认为None。
- ③ *dummy_na*: 是否为类别型数据中的NaN值添加一列, 默认为False。
- ④ *columns* : 表示DataFrame要编码的列名列表, 默认为None。若None, 则对所有类别特征都进行one-hot编码。
- ⑤ *drop_first*: bool。是否删去第一级, 得到k个类别的k-1个哑变量。默认False。



课堂实例

例 类别特征one-hot编码

```
import pandas as pd
df1 = pd.DataFrame({'neighborhood': ['杨浦', '宝山', '嘉定', '静安'],
                    'rooms': [1, 2, 2, 3],
                    'price': [850000, 3000000, 2600000, 10000000]})
```

df1

	neighborhood	rooms	price
0	杨浦	1	850000
1	宝山	2	3000000
2	嘉定	2	2600000
3	静安	3	10000000

```
pd.get_dummies(df1) # 转换为哑变量
```

	rooms	price	neighborhood_嘉定	neighborhood_宝山	neighborhood_杨浦	neighborhood_静安
0	1	850000	0	0	1	0
1	2	3000000	0	1	0	0
2	2	2600000	1	0	0	0
3	3	10000000	0	0	0	1



课堂实例

例2.7 类别特征预处理



类别特征的预处理.ipynb

```
data_file = 'house_tiny.csv'
with open(data_file, 'w', encoding='utf-8-sig') as f:
    f.write('房间数,楼层,价格\n') # 标题行, 由列名构成
    f.write('NA,顶层,2750000\n') # 每行表示一个数据样本
    f.write('2,中间层,4060000\n')
    f.write('4,低层,10780000\n')
    f.write('NA,NA,14000000\n')
```

```
import pandas as pd
import numpy as np
data = pd.read_csv(data_file, header=0)
inputs, outputs = data.iloc[:, 0:2], data.iloc[:, 2]
```

```
inputs1 = inputs.fillna(inputs['房间数'].mean())
```

```
inputs2 = pd.get_dummies(inputs1, dummy_na=True)
```

```
X = np.array(inputs2.values)
y = np.array(outputs.values)
```

	房间数	楼层
0	NaN	顶层
1	2.0	中间层
2	4.0	低层
3	NaN	NaN

inputs

	房间数	楼层
0	3.0	顶层
1	2.0	中间层
2	4.0	低层
3	3.0	NaN

inputs1

	房间数	楼层_中间层	楼层_低层	楼层_顶层	楼层_nan
0	3.0	0	0	1	0
1	2.0	1	0	0	0
2	4.0	0	1	0	0
3	3.0	0	0	0	1

inputs2

场景2：有的数据集中，存在取值非数值的特征，如 蘑菇数据集



mushroom.data

- 机器学习的算法通常只能处理数值类型的数据，若特征数据是字符串，需先转换为数值。
- 可利用scikit-learn提供的LabelEncoder，将原特征的n个不同取值转换为0~n-1之间的数字。



```
mushroom.data - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
p,x,s,n,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,y,t,a,f,c,b,k,e,c,s,s,w,w,p,w,o,p,n,n,g
e,b,s,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,n,m
p,x,y,w,t,p,f,c,n,e,e,s,s,w,w,p,w,o,p,k,s,u
e,x,s,g,f,n,f,w,b,k,t,e,s,s,w,w,p,w,o,e,n,a,g
e,x,y,t,a,f,c,b,n,e,c,s,s,w,w,p,w,o,p,k,n,g
e,b,s,w,t,a,f,c,b,g,e,c,s,s,w,w,p,w,o,p,k,n,m
e,b,y,w,t,l,f,c,b,n,e,c,s,s,w,w,p,w,o,p,n,s,m
p,x,y,w,t,p,f,c,n,p,e,e,s,s,w,w,p,w,o,p,k,v,g
e,b,s,y,t,a,f,c,b,g,e,c,s,s,w,w,p,w,o,p,k,s,m
e,x,y,y,t,l,f,c,b,g,e,c,s,s,w,w,p,w,o,p,n,n,g
e,x,y,y,t,a,f,c,b,n,e,c,s,s,w,w,p,w,o,p,k,s,m
e,b,s,y,t,a,f,c,b,w,e,c,s,s,w,w,p,w,o,p,n,s,g
p,x,y,w,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,n,v,u
```

■ LabelEncoder 标签编码

sklearn.preprocessing中LabelEncoder 将一组非数值数据，进行编码转换。

preprocessing.LabelEncoder()

作用：若一个数组（一个特征的所有取值可视为一个数组）包含n个不同的值，只要值不变且可比较，LabelEncoder将原数组的值转换为0~n-1之间的数字。

方法：`fit_transform(a)`：用数组a训练编码器，并返回编码。

属性：`classes_`：查看每个编码对应的原始值。



课堂实例

一个特征取值为: ["Paris", "BeiJing", "Paris","BeiJing","Berlin"] , 编码如下

```
from sklearn.preprocessing import LabelEncoder  
le = LabelEncoder() # 创建一个编码器对象  
le.fit_transform(["Paris", "BeiJing", "Paris","BeiJing","Berlin"])
```

```
array([2, 0, 2, 0, 1], dtype=int64)
```

```
le.classes_ # 通过classes_属性可以查看转换值对应的原值
```

```
array(['BeiJing', 'Berlin', 'Paris'], dtype='<U7')
```

原值	编码
BeiJing	0
Berlin	1
Paris	2



课堂实例



agaricus-lepiota.data

例 蘑菇数据集各列编码转换

```

mushrooms = pd.read_csv('agaricus-lepiota.data', header=None)
mushrooms.columns = ['class', 'cap-shape', 'cap-surface', 'cap-color', 'bruises',
                    'odor', 'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',
                    'gill-color', 'stalk-shape', 'stalk-root',
                    'stalk-surface-above-ring', 'stalk-surface-below-ring',
                    'stalk-color-above-ring', 'stalk-color-below-ring',
                    'veil-type', 'veil-color', 'ring-number', 'ring-type',
                    'spore-print-color', 'population', 'habitat']

```

`mushrooms.head(3)`

	class	cap- shape	cap- surface	cap- color	bruises	odor	gill- attachment	gill- spacing	gill- size	gill- color	...	stalk- surface- below- ring	stalk- color- above- ring	stalk- color- below- ring	veil- type	veil- color	ring- number	ring- type	spore- print- color	population	habitat
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s	
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n	
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n	

3 rows × 23 columns



课堂实例

例 蘑菇数据集各列编码转换

利用标签编码转换器，进行转换

```
from sklearn.preprocessing import LabelEncoder
```

```
labelenc = LabelEncoder()
```

```
for col in mushrooms.columns:
```

```
    mushrooms[col] = labelenc.fit_transform(mushrooms[col])
```

```
mushrooms.head(3)
```

或应用 **df对象名.apply(函数名)**

```
mushrooms2 = bak_mushrooms.apply(LabelEncoder().fit_transform)
```

	class	cap- shape	cap- surface	cap- color	bruises	odor	gill- attachment	gill- spacing	gill- size	gill- color	...	stalk- surface- below- ring	stalk- color- above- ring	stalk- color- below- ring	veil- type	veil- color	ring- number	ring- type	spore- print- color	population	
0	1	5	2	4	1	6	1	0	1	4	...	2	7	7	0	2	1	4	2	3	
1	0	5	2	9	1	0	1	0	0	4	...	2	7	7	0	2	1	4	3	2	
2	0	0	2	8	1	3	1	0	0	5	...	2	7	7	0	2	1	4	3	2	

3 rows × 23 columns



场景3：有时我们需要把数据拆分到几个区间，每个区间视为一个离散的值。

如，将百分制的成绩分到 $[0,60)$, $[60,70)$, $[70,80)$, $[80,90)$, $[90,100]$ 五个区间中，五个区间分别对应1~5个级别。

- 这种将数据划分到各区间的操作是**数据离散化**。
- Pandas的cut()函数可实现连续数据的离散化操作。

■ **离散化数据** Pandas的cut()函数可实现连续数据的离散化操作。

格式: `pd.cut(data,bins,right=True,labels=None,...)`

返回一个类似数组的对象，表示data中每个值对应的箱子。

说明:

- ① data: 表示要分箱的数组，必须是一维的。
- ② bins: 接收int和序列类型的数据。若传入int类型的值，则表示在data范围内的等宽单元的数目（即划分为多少个等距区间）；若传入一个序列，则表示将data划分在指定的序列中，若不在此序列中，则为NaN。
- ③ right: 是否包含右端点，决定区间的开闭，默认为True。
- ④ labels: 用于生成区间的标签。



课堂实例

例 连续数据离散化

```
scores = pd.read_csv('score1.csv')
bins = [0, 60, 70, 80, 90, 100]
cuts = pd.cut(scores['总分'], bins=bins,
               right=False, labels=[1, 2, 3, 4, 5])
```

```
scores['五级制'] = cuts
```

scores

	学号	总分	五级制
0	10153450101	84.8	4
1	10153450102	74.5	3
2	10153450103	64.5	2
3	10153450104	73.8	3
4	10153450105	72.8	3
5	10153450106	79.0	3
6	10153450107	81.4	4
7	10153450108	65.0	2
8	10153450109	76.7	3
9	10153450110	81.8	4
10	10153450111	42.8	1
11	10153450112	62.1	2
12	10153450113	92.7	5
13	10153450114	74.6	3
14	10153450115	92.4	5
15	10153450116	79.4	3
16	10153450117	53.8	1
17	10153450118	73.3	3
18	10153450119	87.1	4
19	10153450120	67.3	2

(1) 读取文本文件存入DataFrame对象，并给出列名，如：

```
'season','size','speed','mxPH','mnO2','Cl','NO3','NH4','oPO4','PO4','Chla',  
'a1','a2','a3','a4','a5','a6','a7'
```

(2) 剔除'a7'为NaN的行

(3) 将XXXXXXX的值替换为np.nan

(4) 若一行中非NaN的项数小于13，则剔除

(5) 将剩下的数据中NaN，用该列均值替代

(6) 对类别特征进行one-hot编码

(7) 将预处理后的数据存入train1.csv文件中





课堂演示程序

- (1) 读入文本文件analysis.data
- (2) 查找'a7'值为NaN时, NO3'列的值
- (3) 剔除'a7'值为NaN的行
- (4) 查看data的'NO3'列中是否还有带两个小数点的数据
- (5) 将不确定值"XXXXXXX"替换为np.nan
- (6) 从第4列开始, 数据类型都转换为float
- (7) 删除一行中非nan项数小于13的行
- (8) 对于data中含NaN的列,用该列均值 (不含NaN) 来代替NaN
- (9) 查看data是否还有含有NaN的列
- (10) 对于season,size,speed 这些类别值,我们进行one-hot编码
- (11) 将处理后的数据data存入train.csv文件中

**思考：若要从清洗后的数据中
随机抽取样本如何实现？**

□ 数据抽取:

- **字段抽取**: 抽出某列上指定位置的数据构成新列。

如 抽取身份证号码的第7位到15位。

`df[新列名] = df[列名].astype(str)`

[start, end)

↓ `df[新列名] = df[新列名].str.slice(start, end)`

- **记录抽取**: 抽取满足给定条件的行。

`df[条件]` 如 `df[df.ID.isnull()]` 抽取df中ID为缺失值的行。

- **随机抽取行**: 随机从数据中按照给定行数或比例抽取行数据。

↓ `r = numpy.random.randint(start, end, n)` 从[start, end)中随机抽n个整数构成数组。

↓ `df.iloc[r,:]`



Pandas 向量化字符串操作

Pandas 为**包含字符串的Series对象**提供的**str属性**，它既可满足向量化字符串操作的需要，又能正确处理**缺失值**。如：

```
names = pd.Series(['bob','ann',NaN,'mary' ])
```

```
name1 = names.str.capitalize() # pandas字符串方法，大写name中元素的首字母
```

```
name1
```

```
0      Bob
1      Ann
2      NaN
3  Mary a bob
dtype: object
```

```
nameslice = name1.str.slice(0,2) # pandas字符串方法，对names元素进行切片取值
```

```
nameslice
```

```
0      Bo
1      An
2      NaN
3      Ma
dtype: object
```



课堂实例

数据抽取举例

```
from pandas import DataFrame
import numpy as np
```

```
df = DataFrame({'birthday':[19991027,20000119,20000312,19991014],\
                'name':['张莉','李峰',np.nan,'吴峰'],\
                'age':['20','19','19','???']})
```

df

	birthday	name	age
0	19991027	张莉	20
1	20000119	李峰	19
2	20000312	NaN	19
3	19991014	吴峰	???

(1) 抽取birthday中的年份构成新列“ year”。

```
df['year'] = df['birthday'].astype(str)
df['year'] = df['year'].str.slice(0,4)
```

df

	birthday	name	age	year
0	19991027	张莉	20	1999
1	20000119	李峰	19	2000
2	20000312	NaN	19	2000
3	19991014	吴峰	???	1999

(2) 抽取数据框中有缺失值的行

```
df[df.name.isnull()]
```

	birthday	name	age	year
2	20000312	NaN	19	2000

(3) 抽取数据框中age为???的行数据

```
df[df.age=='???']
```

	birthday	name	age	year
3	19991014	吴峰	???	1999

(2) 在清洗后的数据集中，随机抽取100个样本

```
import pandas as pd
import numpy as np

data=pd.read_csv('train.csv',sep=',',index_col=0)
r = np.random.randint(0,data.shape[0],100)
df = data.iloc[r,:]
print(df)
```



作业 2.2

基于Z-score法检测异常值

(1) 自定义一个基于Z-score法的函数，来检查一组数据中是否存在异常值。数据来自文件outlier.csv文件。

(2) 对检查到的异常值进行如下替换：

将english异常值190替换为90；

将computer异常值209,替换为其他值的均值。



作业 2.3

泰坦尼克号幸存者数据预处理

titanic数据共有两个文件：

- train.csv是训练集，样本类别（Survived列）已标注；
- test.csv是测试集，无标注信息。

train.csv是892行(含表头)、12列的数据表。特征如下：

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
乘客ID	1表示幸存, 0表示遇难	舱位等级	乘客姓名	乘客性别	乘客年龄	兄弟姐妹同在船上的数量	同船的父辈人数	乘客票号	乘客的体热指标	乘客所在的船舱号	乘客登船的港口

```
data = pd.read_csv('train.csv', index_col=0)
```

读取train.csv文件，以PassengerId列为行索引，并进行下面要求的预处理。

要求：1)提取Survived列的数据作为目标向量 y，其余为X。

2) 从X中丢弃无用的特征：‘Name’，‘Ticket’，‘Cabin’

3) 将X中缺失数据用0填充。

4) 处理X中的性别数据，将male用0替换，female用1替换。

5) 对X中的Embarked进行one-hot编码转换。