

pandas统计分析基础



概述

机器学习三剑客：

- ⌘ Numpy能够对数据进行很好的分析、操作、矩阵计算等，提高运算效率；
- ⌘ Pandas更侧重于数据的处理和分析，它的底层是使用Numpy实现的，在数据处理和分析方面提供了强大的功能；
- ⌘ Matplotlib 是Python 2D绘图领域的基础套件，它让使用者将数据图形化，并提供多样化的输出格式。

目录

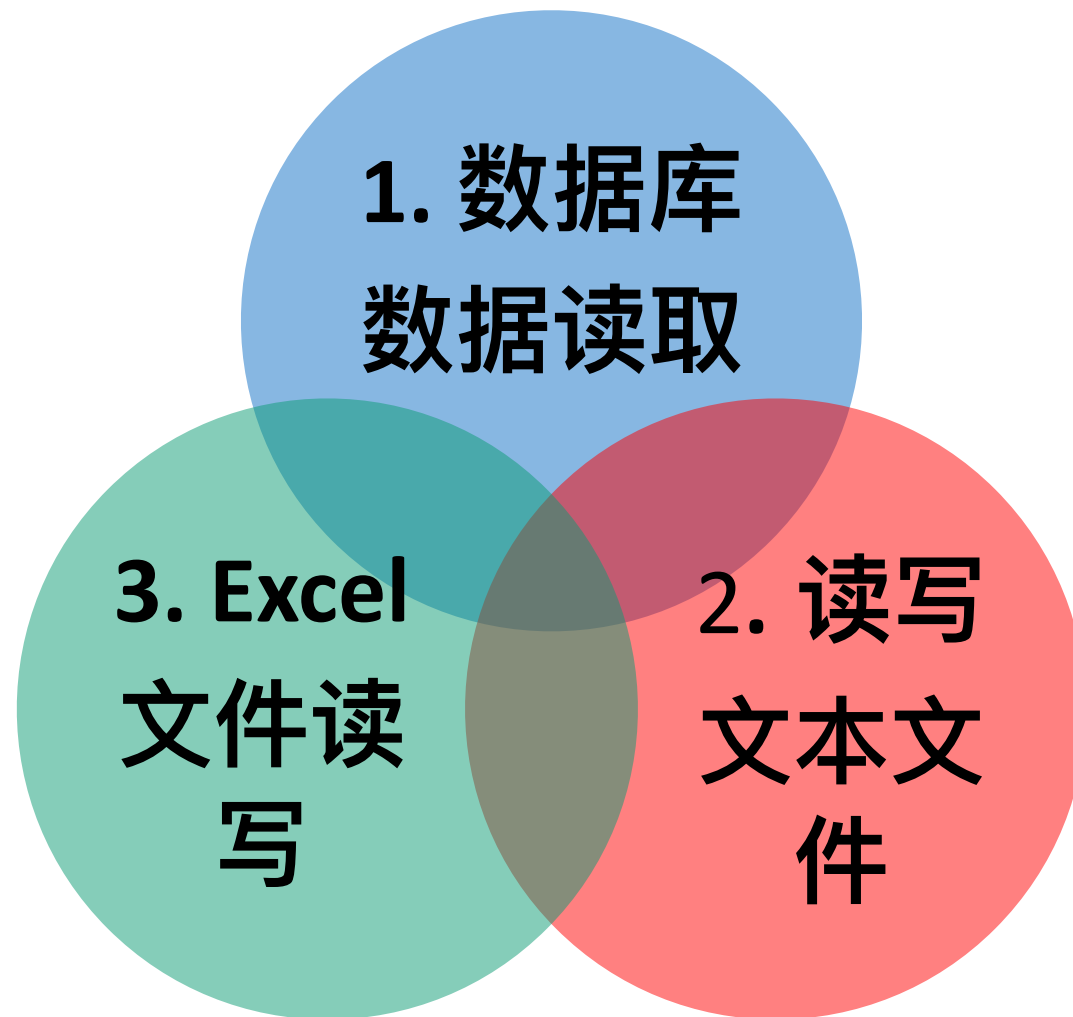
加油



- ① 读写不同数据源的数据
- ② 掌握DataFrame的常用操作
- ③ 转换与处理时间序列数据
- ④ 使用分组聚合进行组内计算
- ⑤ 创建透视表与交叉表
- ⑥ 使用pandas进行数据预处理（合并数据、清洗数据、标准化、转化数据）



一、读写不同数据源的数据



1. 数据库数据读取

需要安装好sqlalchemy和
pymysql库

pandas提供了读取与存储关系型数据库数据的函数与方法。除了pandas库外，还需要使用sqlalchemy库建立对应的数据库连接。sqlalchemy配合相应数据库的Python连接工具（例如MySQL数据库需要安装mysqlclient或者pymysql库），使用create_engine函数，建立一个数据库连接。

creat_engine中填入的是一个连接字符串。在使用Python的SQLAlchemy时，MySQL和Oracle数据库连接字符串的格式如下：

数据库产品名+连接工具名：//用户名:密码@数据库IP地址:数据库端口号/数据库名称? charset = 数据库数据编码

1) 读取数据库的三种方式

➤ `read_sql_table` 只能够读取数据库的某一个表格，不能实现查询的操作。

```
pandas.read_sql_table(table_name, con, schema=None, index_col=None,
coerce_float=True, columns=None)
```

➤ `read_sql_query` 则只能实现查询操作，不能直接读取数据库中的某个表。

```
pandas.read_sql_query(sql, con, index_col=None, coerce_float=True)
```

➤ `read_sql` 是两者的综合，既能够读取数据库中的某一个表，也能够实现查询操作。

```
pandas.read_sql(sql, con, index_col=None, coerce_float=True,
columns=None)
```

参数介绍

pandas三个数据库数据读取函数的参数几乎完全一致，唯一的区别在于传入的是语句还是表名。

参数名称	说明
sql or table_name	接收string。表示读取的数据的表名或者sql语句。无默认。
con	接收数据库连接。表示数据库连接信息。无默认
index_col	接收int，sequence或者False。表示设定的列作为行名，如果是一个数列则是多重索引。默认为None。
coerce_float	接收boolean。将数据库中的decimal类型的数据转换为pandas中的float64类型的数据。默认为True。
columns	接收list。表示读取数据的列名。默认为None。

2) 数据库数据存储

数据库数据读取有三个函数，但数据存储则只有一个to_sql方法。

DataFrame.to_sql(name, con, schema=None, if_exists='fail', index=True, index_label=None, dtype=None)

参数名称	说明
name	接收string。代表数据库表名。无默认。
con	接收数据库连接。无默认。
if_exists	表存在不替换就是fail，如果要替换值为replace
index	接收boolean。表示是否将行索引作为数据传入数据库。默认True。
index_label	接收string或者sequence。代表是否引用索引名称，如果index参数为True此参数为None则使用默认名称。如果为多重索引必须使用sequence形式。默认为None。
dtype	接收dict。代表写入的数据类型（列名为key，数据格式为values）。默认为None。

2. 读写文本文件

- 文本文件是一种由若干行字符构成的计算机文件，它是一种典型的顺序文件。
- csv是一种逗号分隔的文件格式，因为其分隔符不一定是逗号，又被称为字符分隔文件，文件以纯文本形式存储表格数据（数字和文本）。

1) 读取方法

- 使用read_table来读取文本文件。

```
pandas.read_table(filepath_or_buffer, sep='\t', header='infer', names=None, index_col=None, dtype=None, engine=None, nrows=None)
```



- 使用read_csv函数来读取csv文件。

```
pandas.read_csv(filepath_or_buffer, sep=',', encoding='gbk', header='infer', names=None, index_col=None, dtype=None, engine=None, nrows=None)
```

2. 读写文本文件

- 文本文件是一种由若干行字符构成的计算机文件，它是一种典型的顺序文件。
- csv是一种逗号分隔的文件格式，因为其分隔符不一定是逗号，又被称为字符分隔文件，文件以纯文本形式存储表格数据（数字和文本）。

1) 读取方法

- 使用read_table来读取文本文件。

```
pandas.read_table(filepath_or_buffer, sep='\t', header='infer', names=None, index_col=None, dtype=None, engine=None, nrows=None)
```



- 使用read_csv函数来读取csv文件。

```
pandas.read_csv(filepath_or_buffer, sep=',', encoding='gbk', header='infer', names=None, index_col=None, dtype=None, engine=None, nrows=None)
```

参数介绍

read_table和read_csv常用参数及其说明。

参数名称	说明
filepath	接收string。代表文件路径。无默认。
sep	接收string。代表分隔符。read_csv默认为“,”，read_table默认为制表符“[Tab]”。
header	接收int或sequence。表示将某行数据作为列名。默认为infer，表示自动识别。
names	接收array。表示列名。默认为None。
index_col	接收int、sequence或False。表示索引列的位置，取值为sequence则代表多重索引。默认为None。
dtype	接收dict。代表写入的数据类型（列名为key，数据格式为values）。默认为None。
engine	接收c或者python。代表数据解析引擎。默认为c。

2) 文本文件存储

文本文件的存储和读取类似，结构化数据可以通过pandas中的to_csv函数实现以csv文件格式存储文件。

DataFrame.to_csv(path_or_buf=None, sep=',', na_rep="", columns=None, header=True, **index=True**, index_label=None, mode='w', **encoding=None**)

参数名称	说明	参数名称	说明
path_or_buf	接收string。代表文件路径。无默认。	index	接收boolean，代表是否将行名（索引）写出。默认为True。
sep	接收string。代表分隔符。默认为“,”。	index_labels	接收sequence。表示索引名。默认为None。
na_rep	接收string。代表缺失值。默认为“”。	mode	接收特定string。代表数据写入模式。默认为w。
columns	接收list。代表写出的列名。默认为None。	encoding	接收特定string。代表存储文件的编码格式。默认为None。
header	接收boolean，代表是否将列名写出。默认为True。		

3. Excel文件读写

需要安装
xlrd和openpyxl才能读取

1) Excel文件读取

pandas提供了read_excel函数来读取“xls”“xlsx”两种Excel文件。

```
pandas.read_excel(io, sheetname=0, header=0, skiprows=None, skip_footer=0, index_col=None,
names=None, parse_cols=None, parse_dates=False, date_parser=None, na_values=None,
thousands=None, convert_float=True, has_index_names=None, converters=None,
true_values=None, false_values=None, engine=None, squeeze=False, **kwds)
```

参数说明：

- io：用来指定要读取的Excel文件，可以是字符串形式的文件路径、url或文件对象；
- Sheetname：用来指定要读取的worksheet，可以是表示worksheet序号的整数或表示worksheet名字的字符串，如果要同时读取多个worksheet可以使用形如[0, 1, 'sheet3']的列表，如果指定该参数为None则表示读取所有worksheet并返回包含多个DataFrame结构的字典，该参数默认为0（表示读取第一个worksheet中的数据）；
- Skiprows：用来指定要跳过的行索引从0开始组成的列表；

-
- `index_col`: 用来指定作为DataFrame索引的列下标, 可以是包含若干列下标的列表;
 - `headers`: 用来指定worksheet中表示表头或列名的行索引, 默认为0, 如果没有作为表头的行, 必须显式指定`headers=None`;
 - `names`: 用来指定读取数据后使用的列名;
 - `thousands`: 用来指定文本转换为数字时的千分符, 如果excel中有以文本形式存储的数字, 可以使用该参数;
 - `usecols`: 用来指定要读取的列的索引或名字, 如果确定字段类型可以使用`{'col':type}`的形式提高速度;
 - `na_values`: 用来指定哪些值被解释为缺失值。

2) Excel文件存储

将文件存储为Excel文件，可以使用to_excel方法。其语法格式如下：

```
DataFrame.to_excel(excel_writer=None, sheetname=None, na_rep="",  
header=True, index=True, index_label=None, mode='w', encoding=None)
```

与to_csv方法的常用参数基本一致，区别之处在于指定存储文件的文件路径参数名称为excel_writer，并且没有sep参数，增加了一个sheetnames参数用来指定存储的Excel sheet的名称，默认为sheet1。

二、DataFrame操作

1. Series数据结构
2. DataFrame数据结构
3. 操作DataFrame数据



二、DataFrame结构

DataFrame是pandas最常用的数据结构之一。DataFrame的**单列**数据为一个**Series**。多列的DataFrame是一个带有标签的**二维数组**，每个标签相当每一列的列名，可以看作一个二维表格。

注意首字母大写



1. Series数据结构

Series是pandas提供的一维数组，由索引和值两部分组成，是一个类似于字典的结构。其中值的类型可以不同，如果在创建时没有明确指定索引则会自动使用从0开始的非负整数作为索引。

`pd.Series(data, index)`

索引

0	1
1	6
2	11
3	16

→ 值

索引

语文	90
数学	92
python	98
物理	87
化学	92

— 值

例子

```
import pandas as pd
```

```
#不指定索引
```

```
s1 = pd.Series(range(1, 20, 5))
```

```
print(s1)
```

```
#指定索引
```

```
s2 = pd.Series([1,2,3], index=['A', 'B', 'C'])
```

```
print(s2)
```

```
#使用字典创建Series，字典的键作为索引
```

```
s3 = pd.Series({'语文':90, '数学':92, 'python':98, '物理':87, '化学':92})
```

```
print(s3)
```

1) Series基本属性（属性后面没有括号）

函数	返回值
values	元素
index	索引
dtypes	类型
size	元素个数
ndim	维度数
shape	数据形状（行列数目）

2) Series与numpy转换

```
s = pd.Series({'语文':90, '数学':92, 'python':98, '物理':87, '化学':92})
```

```
print(s.values)
```

```
print(np.array(s)) #注意array转换的是数值
```

3) Series中数据读取操作

```
s = pd.Series({'语文':90, '数学':92, 'python':98, '物理':87, '化学':92})
```

```
print(s.values)
```

```
print( s['语文'] ) #等价s[0]
```

```
print(s[0:2])
```

```
print(s[[0, 2]])
```

```
print(s[['数学', 'python']])
```

4) Series中函数操作

```
s = pd.Series({'语文':90, '数学':92, 'python':98, '物理':87, '化学':92})  
print(s.median())  
print(s.mean())  
print(s.std())  
print(s.sum())  
print(s.nsmallest(2))  
print(s.nlargest(2))  
print(s.max())  
print(s.idxmax())  
print(s.value_counts()) #实现频数统计，且结果呈现的是降序排序
```

5) Series中查询操作

```
s = pd.Series({'语文':90, '数学':92, 'python':98, '物理':87, '化学':92})
```

```
print(s[s>90])
```

```
print( s[s>=s.median()])
```

```
print((s>90).sum())
```

```
s = s + 3
```

```
print(s)
```

更新操作

```
arr = s.values
```

```
n_arr = np.where(arr>90, 'good', 'work hard')
```

```
print(n_arr)
```

```
print((arr>90).sum())
```

比较运算符<, <=, >, >=
=, ==, !=等可以用于数字

6) 多个条件的逻辑乘积（和），逻辑和（或）、否定（非）和isin

当组合多个条件时，将每个条件（由其生成的布尔值的对象）括在**括号**
() 中。

□ AND: &

□ 或: |

□ 否定（不）: ~

□ s.isin([87,92])

```
s[(s > 90) & (s < 93)]  
s[~(s==92)]
```

7) 通过指定字符串条件进行计数

pandas.Series的字符串方法str可用于字符串。请注意，字符串方法是pandas.Series方法，不在pandas.DataFrame中。

- ❑ str.contains () : 包含特定的字符串
- ❑ str.endswith () : 以特定字符串结尾
- ❑ str.startswith () : 以特定的字符串开头
- ❑ str.match () : 匹配正则表达式模式

```
s_index[s_index.str.contains('学')]
s_index[s_index.str.endswith('学')]
s_index[s_index.str.match('[a-z]
+')]

```

练习



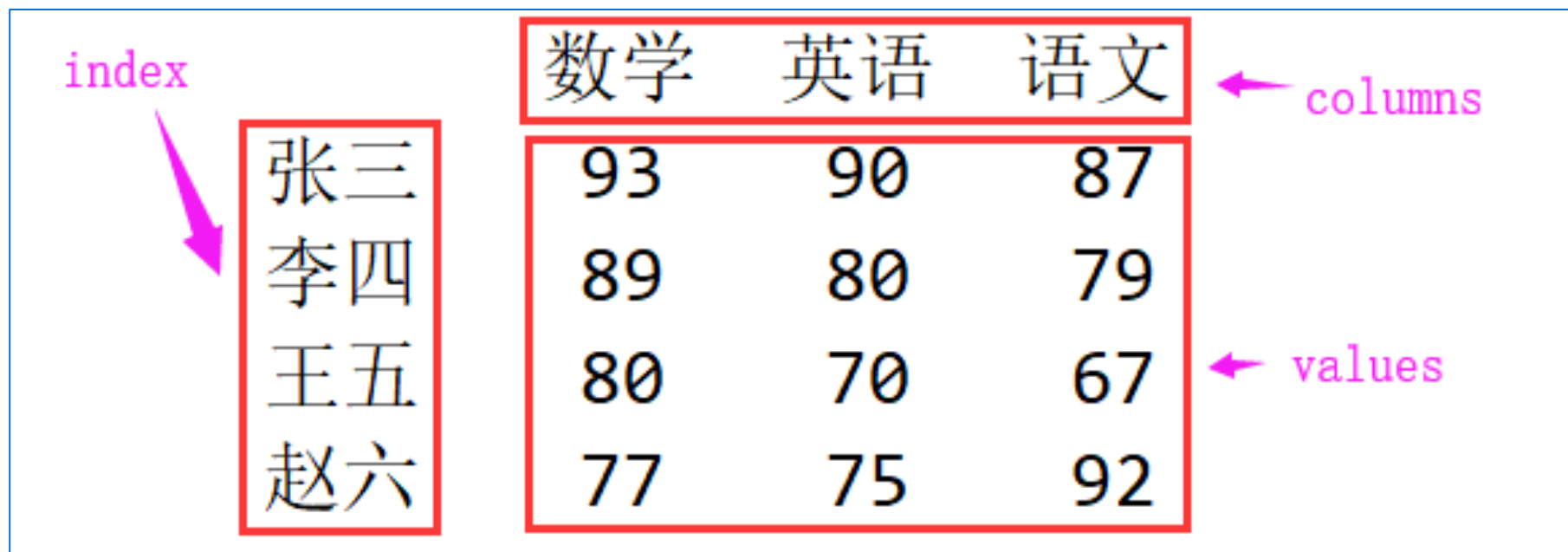
用pandas库分析政府文件中的词频，并过滤不需要的词语。



2. DataFrame数据结构

每个DataFrame对象可以看作一个二维表格，由索引（index）、列名（columns）和值（value）三部分组成。

pd.DataFrame(data, index, columns)



The diagram illustrates the structure of a DataFrame. It shows a table with four rows and three columns. The first column is labeled 'index' with a pink arrow pointing to it. The first row is labeled 'columns' with a pink arrow pointing to it. The remaining cells are labeled 'values' with a pink arrow pointing to them. The table is enclosed in a blue border, and the index, columns, and values are highlighted with red borders.

index	数学	英语	语文
张三	93	90	87
李四	89	80	79
王五	80	70	67
赵六	77	75	92

1) DataFrame的创建

#行和列都是从0开始的默认索引

```
pd.DataFrame(np.random.randint(0,10, (3,3)))
```

#设置列标题

```
pd.DataFrame(np.random.randint(0,10, (3,3)), columns=['A', 'B', 'C'])
```

#同时设置行、列标题

```
pd.DataFrame(np.random.randint(0,10, (3,3)), columns=['A', 'B', 'C'], index=['张三', '李四', '王五'])
```

#通过字典创建，字典的键就是列标题

```
pd.DataFrame({'语文': [87, 79, 76, 92],  
              '数学': [93, 89, 80, 77],  
              '英语': [90, 80, 70, 75]},  
              index=['张三', '李四', '王五', '赵六'])
```

练习

模拟2022年4月1日某个超市熟食、化妆品、日用品从早8：00-晚8：00
每小时的销量，销售从5~15中随机抽取。

2) DataFrame的基础属性（属性后面没有括号）

属性名	返回值
values	元素（值）
index	索引
columns	列名
dtypes	类型
size	元素个数
ndim	维度数
shape	数据形状（行列数目）

3) 查看整体数据和整体描述统计分析

1、info()函数

2、describe()函数：能够一次性得出数据框所有数值型特征的非空值数目、均值、四分位数、标准差。

参数：

- include：包含哪类数据。默认只包含连续值，不包含离散值；include = 'all' 设置全部类型
- Percentiles：设置输出的百分位数，默认为[.25,.50,.75]，返回第25，50，75百分位数

结果分析：

✓ 对连续值来说：

count:每一列非空值的数量

mean: 每一列的平均值

std:每一列的标准差

min: 最小值

25%: 25%分位数，排序之后排在25%位置的数

50%: 50%分位数

75%: 75%分位数

max:最大值

✓ 对离散值来说特有的：

count:每一列非空值的数量

unique: 不重复的离散值数目，去重之后的个数

top: 出现次数最多的离散值

freq: 上述的top出现的次数

4) 查看访问DataFrame中的数据

- ♣ 访问**每一列**：**df ['列名称']**。因为DataFrame的数据组成形式，是一列列的Series拼接的。
- ♣ 访问**多列**：将多个列索引名称**放在一个列表**
- ♣ 访问**行**：用切片。如**df[1:3]**
- ♣ DataFrame结构还提供了loc、iloc等访问器来访问指定的数据。其中，**iloc**使用整数来指定行、列的**索引**，而**loc**使用**标签**指定要访问的行和列。

`DataFrame.iloc[行索引位置, 列索引位置]`

`DataFrame.iloc[行索引位置, 列索引位置]`

例子

```
data = pd.read_csv('yingyee.csv',encoding='gbk')
```

```
data['交易额']
```

```
data[['姓名', '交易额']]
```

列访问

```
data.head(10)
```

```
data[1:3]
```

行访问

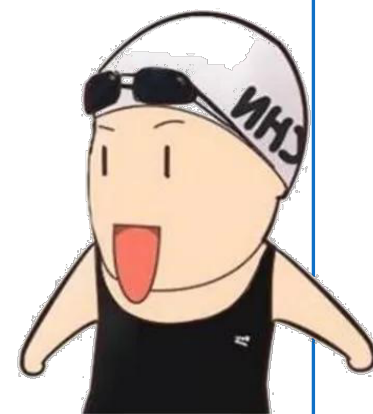
```
data[['姓名', '交易额']][1:3]
```

```
data.iloc[:4, 1:4]
```

```
data.loc[:4, '姓名':'交易额']
```

行列访问

```
data.loc[[1,3], ['姓名', '交易额']]
```



很轻松嘛！

5) 查询数据 (1)

```
df[df['交易额']>1700]
```

```
df[(df['交易额']>1700) & (df['交易额']<1800)]
```

```
df[df['交易额']>1700]['交易额'].mean()
```

```
df[df['时段'] == '14: 00-21: 00']['交易额'].sum()
```

#张三下午的交易情况

```
df[(df['姓名'] == '张三') & (df['时段'] == '14: 00-21: 00')][:10]
```

#姓名是张三或者时间是下午的交易情况

```
df[(df['姓名'] == '张三') | (df['时段'] == '14: 00-21: 00')][:10]
```

不支持这样写：
1800>df['交易额']>1700,
也不支持and

5) 查询数据 (2)

#筛选出多个值可以用isin(), 参数为可迭代对象, 往往是列表

```
df[df['姓名'].isin(['张三', '李四'])['交易额'].sum()
```

#字符串的模糊筛选可以用.str.contains()来实现。

```
data.loc[data['姓名'].str.contains('张|李'),'姓名']='陈九'
```

loc访问器除了之前的作用外, 还可以根据某个位置的True or False 来选定, 如果某个位置的布尔值是True, 则选定该row

6) 修改DataFrame中的数据

修改DataFrame中的数据，原理是将这部分数据提取出来，重新赋值为新的数据。

例如：

```
df['交易额'] = df['交易额'] + 1000
```

```
data.loc[data['姓名']=='张三', '交易额']=1000
```

7) 为DataFrame增添数据列

DataFrame添加一列的方法非常简单，只需要**新建一个列索引**。并对该索引下的数据进行**赋值**操作即可。新增列的值都相同则直接赋一个常量值即可。

例如：

```
df ['total'] =5000
```

```
df['total'] = df['交易额']*5
```

8) 删除某列或某行数据

删除某列或某行数据需要用到pandas提供的方法**drop**，drop方法的用法如下。如果是删除某列，也可以直接用**del**

`drop(labels, axis=0, inplace=False)`

参数名称	说明
labels	接收string或array。代表删除的 行或列的标签 。无默认。
axis	接收0或1。代表操作的轴向。 默认为0 。 axis为0时表示删除行，axis为1时表示删除列。
inplace	接收boolean。代表操作是否对原数据生效。默认为False。

删除列

```
df.drop(['total'], axis=1,  
inplace=True)
```

```
del df['total'] #与上一行等价  
df.columns
```

删除行

```
data.drop(data[data['交易额']<1000].index, inplace=True)
```

9) 数据类型查询及转换

可以通过下面两种方法查看数据类型：

- info函数
- dtype属性

例子：

```
data['交易额'].dtype
```

```
data['交易额'].astype(str)
```

```
data['日期'] = pd.to_datetime(data['日期'])
```

```
data.info()
```


案例———标准差std或者方差var

1、某校两名学生的成绩都很优秀，但参加物理竞赛的名额只有一个，选谁去获得名次的几率更大的？

数据如下：

	物理1	物理2	物理3	物理4	物理5
小黑	110	113	102	105	108
小白	118	98	119	85	118

案例———协方差

读取'xiefangcha.csv'数据：

- 1、计算不同柜台的交易额数据的标准差和协方差。根据运行结果进行分析哪个柜台的员工间的交易额相差最小，以及不同柜台之间的联系。
- 2、求每个员工的销售总额，添加到原数据的最后一列中
- 3、求每个员工的平均销售额，添加到销售总额这一列的后面
- 4、把销售总额低于30%的员工找出来Dataframe.quantile(**q=0.3**)
- 5、求化妆品列的平均值

2、3两题先后
顺序不能换

例子

#间隔5天

```
print(pd.date_range(start='20210401', end='20210430', freq='5D'))
```

#间隔2天，生成5个数据。注意这里就不要end值了

```
print(pd.date_range(start='20210401', periods=5, freq='2D'))
```

#在[5,15]区间上生成13行3列39个随机数

```
df = pd.DataFrame(np.random.randint(5, 15, (13, 3)),
```

```
index=pd.date_range(start='202204010900', end='202204012100', freq='H'),
```

```
columns=['熟食', '化妆品', '日用品'])
```

3、清洗数据



1) 检测与
处理重复值



2) 检测与
处理缺失值



3) 检测与
处理异常值

3、清洗数据



1) 检测与
处理重复值



2) 检测与
处理缺失值



3) 检测与
处理异常值

1) 检测与处理重复值

① 判断行与行数据是否重复

- `df.duplicated()`
- 返回值为True 或者False

② 去除指定列的重复数据

- `drop_duplicates`不仅支持单一特征的数据去重，还能够依据DataFrame的几个特征进行去重操作。
- `df.drop_duplicates(subset=['A','B'],keep='first',inplace=True)`
 - ✓ Subset: 对应的值是列名，这里表示只考虑A、B两列，将这两列对应值相同的行进行去重。默认值为subset=None表示考虑所有列。
 - ✓ Keep: 'first'表示保留第一次出现的重复行，是默认值。keep另外两个取值为"last"和False，分别表示保留最后一次出现的重复行和去除所有重复行。
 - ✓ Inplace: 接收boolean。表示是否在原表上进行操作。默认为False。

2) 检测与处理缺失值

① 查看缺失值

➤ 使用`info`函数

② 利用`isnull`或`notnull`找到缺失值

➤ 数据中的某个或某些特征的值是不完整的，这些值称为缺失值。

➤ pandas提供了识别缺失值的方法`isnull`以及识别非缺失值的方法`notnull`，这两种方法在使用时返回的都是布尔值`True`和`False`。`isnull`和`notnull`之间结果正好相反，因此使用其中任意一个都可以判断出数据中缺失值的位置。

➤ 结合`sum`函数和`isnull`、`notnull`函数，可以检测数据中缺失值的分布以及数据中一共含有多少缺失值。

③ 处理缺失值

第一种方法：删除缺失值

删除法分为删除观测记录和删除特征两种，它属于利用减少样本量来换取信息完整度的一种方法，是一种最简单的缺失值处理方法。

pandas中提供了简便的删除缺失值的方法dropna，该方法既可以删除观测记录，亦可以删除特征。

DataFrame.dropna(axis=0, how='any', subset=None, inplace=False)

参数名称	说明
axis	接收0或1。表示轴向，0为删除观测记录（行），1为删除特征（列）。默认为0。
how	接收特定string。表示删除的形式。any表示只要有缺失值存在就执行删除操作。all表示当且仅当全部为缺失值时执行删除操作。默认为any。
subset	接收类array数据。表示进行去重的列/行。默认为None，表示所有列/行。
inplace	接收boolean。表示是否在原表上进行操作。默认为False。

第二种方法：替换法

当缺失值比较多时，删除法就不太合适了。替换法是指用一个特定的值替换缺失值。特征可分为数值型和类别型，两者出现缺失值时的处理方法也是不同的。

pandas库中提供了缺失值替换的方法名为**fillna**，其基本语法如下。

```
pandas.DataFrame.fillna(value=None, method=None, axis=None, inplace=False)
```

- 缺失值所在特征为**数值型**时，通常利用其**均值、中位数和众数**等描述其集中趋势的统计量来代替缺失值。
- 缺失值所在特征为**类别型**时，则选择使用**众数**来替换缺失值。

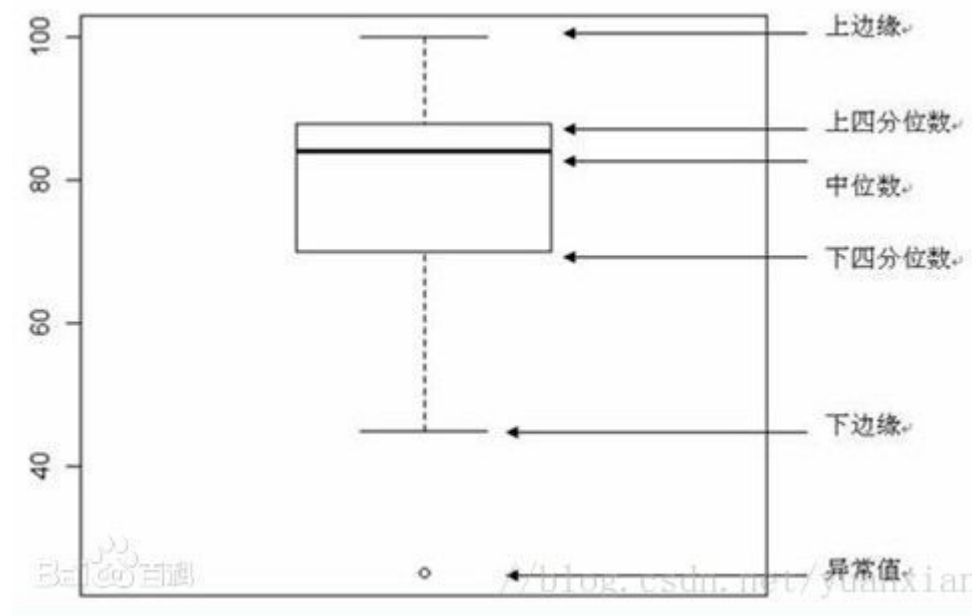
参数名称	说明
value	接收scalar, dict, Series或者DataFrame。表示用来替换缺失值的值。无默认。
method	接收特定string。backfill或bfill表示使用下一个非缺失值填补缺失值。pad或ffill表示使用上一个非缺失值填补缺失值。默认为None。
axis	接收0或1。表示轴向。默认为1。
inplace	接收boolean。表示是否在原表上进行操作。默认为False。

3) 检测与处理异常值

箱型图经常被用来检测异常值。任何高于上限或低于下限的数据都可以认为是异常值。

箱形图有5个参数：

- 下边缘（下限），表示最小值；
- 下四分位数（Q1），又称“第一四分位数”，等于该样本中所有数值由小到大排列后第25%的数字；
- 中位数（Q2），又称“第二四分位数”等于该样本中所有数值由小到大排列后第50%的数字；
- 上四分位数（Q3），又称“第三四分位数”等于该样本中所有数值由小到大排列后第75%的数字；
- 上边缘（上限），表示最大值。
- 第三四分位数与第一四分位数的差距又称四分位间距。



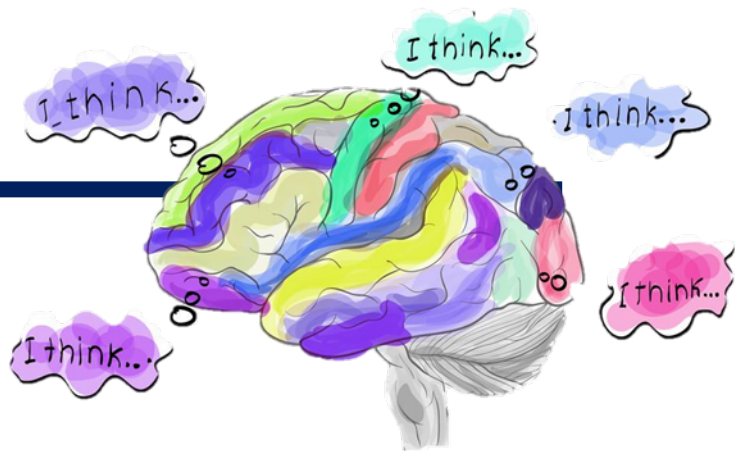
处理异常值

了解了异常值后，接下来介绍如何处理异常值，主要包括以下几方面：

- 1、最常用的方式是删除
- 2、将异常值当缺失值处理，以某个值填充
- 3、将异常值当特殊情况处理，研究异常值出现的原因

4、使用分组聚合进行组内计算

- 1) 按不同标准对数据排序
- 2) 使用分组groupby()分组计算
- 3) 使用agg方法聚合数据
- 4) apply方法聚合数据



1) 按不同标准对数据排序sort_values

```
sort_values(by, axis=0, ascending=True, inplace=False, na_position='last')
```

- 参数by用来指定依据哪个或哪些名字的列进行排序，如果只有一列则直接写出列名，**多列**的话需要放到**列表**中；
- 参数ascending=True表示升序排序，ascending=False表示降序排序，如果ascending设置为包含若干True/False的列表（必须与by指定的列表长度相等），可以为不同的列指定不同的顺序；
- 参数na_position用来指定把缺失值放在最前面（na_position='first'）还是最后面（na_position='last'）。

按不同标准对数据排序sort_index

sort_index()沿某个方向按标签进行排序并返回一个新的DataFrame。

```
sort_index(axis=0, ascending=True,  
inplace=False)
```

axis=0时表示根据行索引标签进行排序，axis=1时表示根据列名进行排序

2) 使用groupby()分组计算

对数据进行分组统计，主要使用DataFrame对象的groupby()函数，将数据按照一系列或多列进行分组，其功能如下：

1. 根据给定的条件将数据拆分分组
2. 每个组都可以独立应用函数（如sum()、mean()等）
3. 将结果合并到一个数据结构中

```
groupby(by=None, axis=0, level=None, as_index=True, sort=True,  
group_keys=True, squeeze=False, **kwargs)
```

- ✓ 参数by用来指定列名作为分组依据，也可以是index的函数、列表、字符串、字典等。
- ✓ axis表示操作的轴向，默认0表示对列进行操作。
- ✓ as_index=False时用来分组的列中的数据不作为结果DataFrame对象的index

GroupBy对象常用的描述性统计方法

用groupby方法分组后的结果并不能直接查看，而是被存在内存中，输出的是内存地址。实际上分组后的数据对象GroupBy类似Series与DataFrame，是pandas提供的一种对象。GroupBy对象常用的描述性统计方法如下。

方法名称	说明	方法名称	说明
count	计算分组的数目，包括缺失值。	cumcount	对每个分组中组员的进行标记，0至n-1。
head	返回每组的前n个值。	size	返回每组的大小。
max	返回每组最大值。	min	返回每组最小值。
mean	返回每组的均值。	std	返回每组的标准差。
median	返回每组的中位数。	sum	返回每组的和。

例子

```
print(df.groupby(by='时段'))  
print(df.groupby(by='时段')['交易额'].sum())  
print(df.groupby(by='时段').sum()['交易额'])
```

不能直接查看到结果

时段

14: 00-21: 00 151228.0

9: 00-14: 00 176029.0

Name: 交易额, dtype: float64

#统计员工上班次数

```
dfn = df.groupby(by='姓名')['日期'].count()  
dfn.name = '上班次数'  
print(dfn)
```

这里的dfn.name是针对输出结果中的
Name变成上班次数

#根据交易总额给工作人员排名

```
dff = df.groupby(by='姓名')['交易额'].sum().rank(ascending=True)
```

3) 使用agg方法聚合数据

agg支持对每个分组应用某函数，包括Python内置函数或自定义函数，也能够直接对DataFrame进行函数应用操作。

DataFrame.agg(func, axis=0, **kwargs)

参数名称	说明
func	接收list、dict、function。表示应用于每行 / 每列的函数。无默认。
axis	接收0或1。代表操作的轴向。默认为0。

例子

- 可以使用agg方法一次求某数值列的中值与均值，如

```
df['交易额'].agg([np.median, np.mean])
```

```
df[['交易额', '工号']].agg(['median', 'mean'])
```

- 对于某个字段希望只做求均值操作，而对另一个字段则希望只做求和操作，可以使用字典的方式，将两个字段名分别作为key，求和与求均值的函数分别作为value，如

```
df.agg({'交易额': 'median', '工号': 'mean'})
```

```
df.agg({'交易额': ['median', 'sum'], '工号': 'mean'})
```

- 对分组结果进行聚合

```
df.groupby(by='姓名').agg({'交易额': ['max', 'min', 'mean', 'median'], '日期': ['max', 'min']})
```

4) apply方法聚合数据

apply方法类似agg方法能够将函数应用于每一列。不同之处在于apply方法只能够作用于整个DataFrame或者Series，而无法像agg一样能够对不同字段，应用不同函数获取不同结果。

`DataFrame.apply(func, axis=0, **kwargs)`

例如：

```
df['交易额'].apply('sum')
```

```
df[['交易额', '工号']].apply('sum')
```

```
df['新的交易额'] = df['姓名'].apply(lambda x: 5 if x=='张三' else 1) * df['交易额']
```

5、使用pivot_table函数创建透视表

利用pivot_table函数可以实现透视表， pivot_table()函数的常用参数及其使用格式如下：

```
DataFrame.pivot_table(values=None, index=None, columns=None, aggfunc='mean', fill_value=None, margins=False, dropna=True, margins_name='All')
```

参数名称	说明
values	接收string或list。用于指定想要聚合的数据字段名，默认为None。
index	接收string或list。表示行分组键。默认为None。
columns	接收string或list。表示列分组键。默认为None。
aggfunc	接收functions。表示聚合函数。默认为mean。多个聚合结果放在list中
margins	接收boolean。表示汇总（Total）功能的开关，设为True后结果集中会出现名为“ALL”的行和列。默认为False。
fill_value	当某些数据不存在时，会自动填充NaN，因此可以指定fill_value参数，表示当存在缺失值时，以指定数值进行填充。
dropna	接收boolean。表示是否删掉全为NaN的列。默认为True。

6、使用crosstab函数创建交叉表

交叉表是一种特殊的透视表，由pandas提供。crosstab函数格式如下：

```
pandas.crosstab(index, columns, values=None, rownames=None,
colnames=None, aggfunc=None, margins=False, dropna=True,
normalize=False)
```

其参数与透视表基本保持一致，不同之处在于crosstab函数中的index，columns，values填入的都是对应的从Dataframe中取出的某一行。

参数名称	说明
index	接收string或list。表示行索引键。无默认。
columns	接收string或list。表示列索引键。无默认。
values	接收array。表示聚合数据。默认为None。
aggfunc	接收function。表示聚合函数。默认为None。
rownames	表示行分组键名。无默认。
colnames	表示列分组键名。无默认。
dropna	接收boolearn。表示是否删掉全为NaN的。默认为False。
margins	接收boolearn。默认为False。汇总（Total）功能的开关，设为True后结果集中会出现名为“ALL”的行和列。
normalize	接收boolearn。表示是否对值进行标准化。默认为False。