pandas 笔记

1.了解 pandas?

Pandas 是一个用于数据操作和数据分析的 Python 库。它提供了一种灵活且高效的数据结构 DataFrame 和 Series，可以快速处理各种数据类型（如 CSV、Excel、SQL 数据库等），并能够进行数据清洗、数据筛选、数据统计、数据可视化等操作。Pandas 库为数据科学家和数据分析师提供了一个强大而易于使用的工具集，可以帮助他们更快地进行数据探索和分析。

Pandas 库是 Python 中用于数据分析的重要工具之一，主要用于处理和分析数据。以下是 Pandas 库在数据分析中的常见用途：

数据读取和写入：Pandas 可以读取和写入各种数据格式，如 CSV、Excel、SQL 等。

数据清洗和预处理：Pandas 可以处理数据中的缺失值、重复值、异常值等，并进行数据类型转换、数据重塑等预处理操作。

数据分组和聚合：Pandas 可以通过分组和聚合操作对数据进行统计分析。

数据可视化：Pandas 可以使用 Matplotlib 等可视化工具绘制各种图表，如折线图、散点图、柱状图等。

时间序列分析：Pandas 可以处理时间序列数据，并进行时间序列分析、滑动窗口分析等操作。

数据合并和拆分：Pandas 可以合并多个数据集，并进行数据拆分、数据透视等操作，以便进行更深入的数据分析。

总之，Pandas 库是 Python 中非常强大的数据分析工具，可以帮助数据分析师处理和分析各种数据，提高数据分析的效率和准确性。

2.安装 pandas、numpy、xlrd

3.数据类型与新建文件

数据类型	说明	新建方法
csv、tsv、txt	用逗号分隔、tab分割的纯文本文件	pd.to_csv
excel	xls或xlsx	pd.to_excel
mysql	关系数据库表	pd.to_sql

3.1 新建空白 Excel 文件

```
import pandas as pd
路径 = 'c:/pandas/新建空白文件.xlsx'
数据=pd.DataFrame( )
数据.to_excel(路径)
print('新建空白文件.xlsx 成功')
```

3.2 新建文件同时写入数据

```
import pandas as pd
路径 = 'c:/pandas/新建空白文件.xlsx'
数据=pd.DataFrame({'id':[1,2,3],'姓名':['叶问','李小龙','条条']})
数据.to_excel(路径)
print('新建空白文件.xlsx 成功')
```

3.3 将 id 设置为索引

```
import pandas as pd
路径 = 'c:/pandas/新建空白文件.xlsx'
数据=pd.DataFrame({'id':[1,2,3],'姓名':['叶问','李小龙','条条']}) # 写入的数据
数据=数据.set_index('id') # 将 id 设置为索引
数据.to_excel(路径) # 将数据写入 Excel 文件
print(数据)
```

4..读取数据

数据类型	说明	读取方法
csv、tsv、txt	默认逗号分隔	pd.read_csv
csv、tsv、txt	默认\t分隔	pd.read_table
excel	xls或xlsx	pd.read_excel
mysql	关系数据库表	pd.read_sql

4.1 读取 txt 与 csv 文件

```
import pandas as pd
路径 = 'c:/pandas/读取文件.csv'
读取数据 = pd.read_csv(路径)
print(读取数据)
print(读取数据.head()) # 默认是 5 行，指定行数写小括号里
print(读取数据.shape) # 查看数据的形状，返回（行数、列数）
print(读取数据.columns) # 查看列名列表
print(读取数据.index) # 查看索引列
print(读取数据.dtypes) # 查看每一列数据类型
```

指定分隔符和列名

```
import pandas as pd
路径 = 'c:/pandas/读取文件.csv'
读取数据 = pd.read_csv(路径,sep=',',header=None,names=['姓名','年龄','地址','电话','入职日期'],encoding='utf-8',index_col='入职日期')
print(读取数据)
```

注意：你的 txt 文档必需另存为 utf-8 编码，如果是 ASCII 报错

参数	描述
sep	分隔符或正则表达式 sep='\s+'
header	列名的行号，默认0（第一行），如果没有列名应该为None
names	列名，与header=None一起使用
index_col	索引的列号或列名，可以是一个单一的名称或数字，也可以是一个分层索引
skiprows	从文件开始处，需要跳过的行数或行号列表
encoding	文本编码，例如utf-8
nrows	从文件开头处读入的行数　nrows=3

4.2 读取 mysql 数据库

```
import pandas as pd
import pymysql
连接对象 = pymysql.connect(host = 'localhost',user = 'root',password = '1234',database = 'test',charset = 'utf8')
读取文件 = pd.read_sql("select * from 1 班",con=连接对象)
print(读取文件)
```

4.3 读取 Excel 数据

```
import pandas as pd
路径 = 'c:/pandas/读取文件.xlsx'
读取数据 = pd.read_excel(路径,header=None,names=['序号','姓名','年龄','手机','地址','入职日期'],index_col='序号')
print(读取数据)
读取数据.to_excel(路径)
```

5.pandas 数据结构

DataFrame：二维数据，整个表格，多行多列【简称 df】
df.index：索引列
df.columns：列名
Series：一维数据，一行或一列

id	姓名	性别	来自
1	杨过	男	神雕
2	小龙女	女	神雕
3	张无忌	男	倚天
4	周芷若	女	倚天
5	赵敏	女	倚天

5.1 Series 一维数据，一行或一列

Series 是一种类似于一维数组的对象，它由一组数据（不同数据类型）以及一组与之相关的数据标签（即索引）组成。

仅有数据列表即可产生最简单的 Series

```
import pandas as pd
df= pd.Series([520,'条条',1314,'2020-07-30']) #
左侧是索引，右侧是数据
print(df)
print(df.index) # 获取索引，返回索引的（起始值，结束值，步长）
print(df.values) # 获取数据，返回值序列，打印元素值的列表
```

使用 Python 字典创建 Series

```
import pandas as pd
字典={'姓名':'条条','性别':'男','年龄':'20','地址':'花果山水帘洞'}
df=pd.Series(字典)
print(df)
print(df.index) # 返回 key
```

根据标签索引查询数据

```
print(df) # 查询整个字典
print(df['姓名']) # 通过 key 可以查对应的值
type(df['年龄']) # 通过 key 可以查对应值的类型
print(df[['姓名','年龄']]) # 通过多个 key 查对应的值
type(df[['姓名','年龄']]) # 注意：他不返回值的类型，而返回 Series
```

Series 常用方法

```
df.index #查看索引
df.values #查看数值
df.isnull() #查看为空的，返回布尔型
```

```
df.notnull()
df.sort_index() #按索引排序
df.sort_values() #按数值排序
```

5.2 认识 DataFrame

DataFrame 是一个表格型的数据结构
• 每列可以是不同的值类型（数值、字符串、布尔值等）
• 既有行索引 index，也有列索引 columns
• 可以被看做由 Series 组成的字典
创建 DataFrame 最常用的方法，参考读取 CSV、TXT、Excel、MySQL 等

5.2.1 *多个字典序列创建 DataFrame*

```
import pandas as pd
字典 = {
'姓名':['条条','李小龙','叶问'],
'年龄':[20,80,127],
'功夫':['撸铁','截拳道','咏春']
}
数据 = pd.DataFrame(字典)
print(数据)
print(数据.dtypes) # 返回每一列的类型
print(数据.columns) # 返回列索引，以列表形式返回：[列名 1，列名 2，…]
print(数据.index) # 返回行索引，（起始，结束，步长）

• 如果只查询一列，返回的是 pd.Series
print(数据['姓名']) # 返回索引和这一列数据
type(数据['姓名']) # 类型返回 Series
• 如果只查询一行，返回的是 pd.Series
print(数据.loc[1]) # 这时，它的索引是列名
type(数据.loc[1]) # 类型返回 Series
• 如果查询多列，返回的是 pd.DataFrame
print(数据[['姓名','年龄']]) # 返回索引和这两列数据
type(数据[['姓名','年龄']]) # 类型返回 DataFrame
• 如果查询多行，返回的是 pd.DataFrame
print(数据.loc[1:3]) # 返回前 3 行，包括结束值
type(数据.loc[1:3]) # 类型返回 DataFrame
```

5.2.2 *将多个 Series 加入 DataFrame*

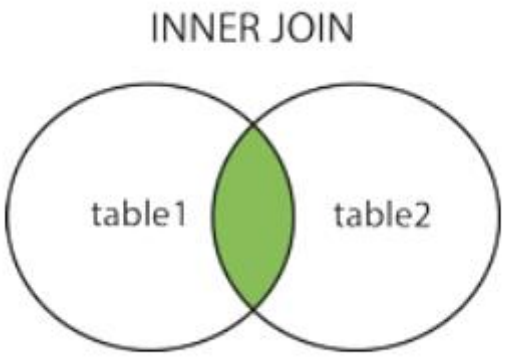
```
import pandas as pd
数据 1 = pd.Series(['叶问','李小龙','条条'],index=[1,2,3],name='姓名')
数据 2 = pd.Series(['男','男','男'],index=[1,2,3],name='性别')
```

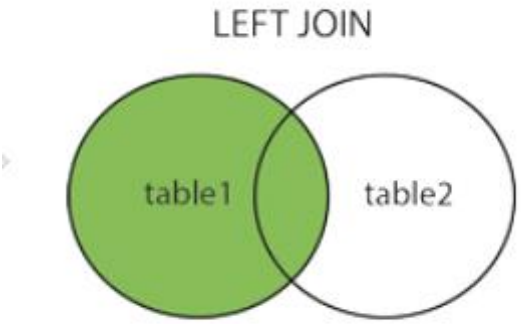
```
数据 3 = pd.Series([127,80,20],index=[1,2,3],name='年龄')
表 1 = pd.DataFrame({数据 1.name:数据 1,数据 2.name:数据 2,数据 3.name:数据 3})
print(表 1)
表 2 = pd.DataFrame([数据 1,数据 2,数据 3])
print(表 2)
```

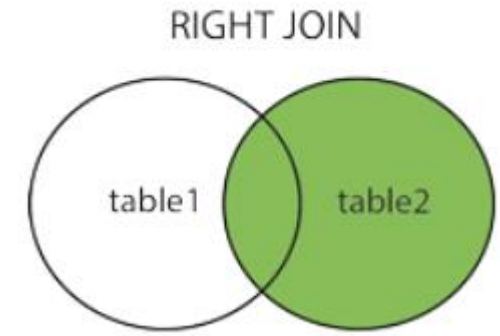
5.2.3 DataFrame 常用方法

```
数据.head( 5 ) #查看前 5 行
数据.tail( 3 ) #查看后 3 行
数据.values #查看数值
数据 shape #查看行数、列数
数据.fillna(0) #将空值填充 0
数据.replace( 1, -1) #将 1 替换成-1
数据.isnull() #查找数据中出现的空值
数据.notnull() #非空值
数据.dropna() #删除空值
数据.unique() #查看唯一值
数据.reset_index() #修改、删除，原有索引，详见例 1
数据.columns #查看数据的列名
数据.index #查看索引
数据.sort_index() #索引排序
数据.sort_values() #值排序
pd.merge(数据 1,数据 1) #合并
pd.concat([数据 1,数据 2]) #合并，与 merge 的区别，自查
pd.pivot_table( 数据 ) #用 df 做数据透视表（类似于 Excel 的数透）
```

6.连接查询







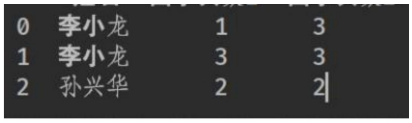
6.1 Merge

首先 **merge** 的操作非常类似 **sql** 里面的 **join**，实现将两个 **Dataframe** 根据一些共有的列连接起来，当然，在实际场景中，这些共有列一般是 **Id**，
连接方式也丰富多样，可以选择 **inner(默认)**，**left,right,outer** 这几种模式，分别对应的是内连接，左连接，右连接，全外连接

参数	描述
how	数据融合的方法，从在不重合的键，方式（inner、outer、left、right）
on	用来对齐的列名，一定要保证左表和右表存在相同的列名。
left_on	左表对齐的列，可以是列名。也可以是DataFrame同长度的arrays
right_on	右表对齐的列，可以是列名。
left_index	将左表的index用作连接键
right_index	将右表的index用作连接键
suffixes	左右对象中存在重名列，结果区分的方式，后缀名。
copy	默认：True。将数据复制到数据结构中，设置为False提高性能。

6.1.1 InnerMerge (内连接)

首先让我们简单的创建两个数据,分别为数据 1 和数据 2，它们的公有列是 key




```
import numpy as np
import pandas as pd
数据 1= pd.DataFrame({'姓名':['叶问','李小龙','条条','李小龙','叶问','叶问'],'出手次数 1':np.arange(6)})
数据 2 = pd.DataFrame({'姓名':['黄飞鸿','条条','李小龙'],'出手次数 2':[1,2,3]})
数据 3 = pd.merge(数据 1,数据 2)
print(数据 3)

数据 3 = pd.merge(数据 1,数据 2,on='姓名',how='inner')
```

6.1.2 LeftMerge (左连接)

```
数据 3 = pd.merge(数据 1,数据 2,on='姓名',how='left')
```

6.1.3 RightMerge (右连接)

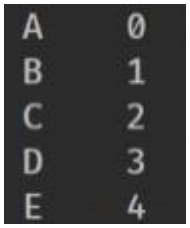
```
数据 3 = pd.merge(数据 1,数据 2,on='姓名',how='right')
```

6.1.4 OuterMerge (全连接)

```
数据 3 = pd.merge(数据 1,数据 2,on='姓名',how='outer')
```

6.2 Concat

```
import pandas as pd
数据 1 = pd.Series([0,1,2],index=['A','B','C'])
数据 2 = pd.Series([3,4],index=['D','E'])
数据 3 = pd.concat([数据 1,数据 2])
print(数据 3)
```

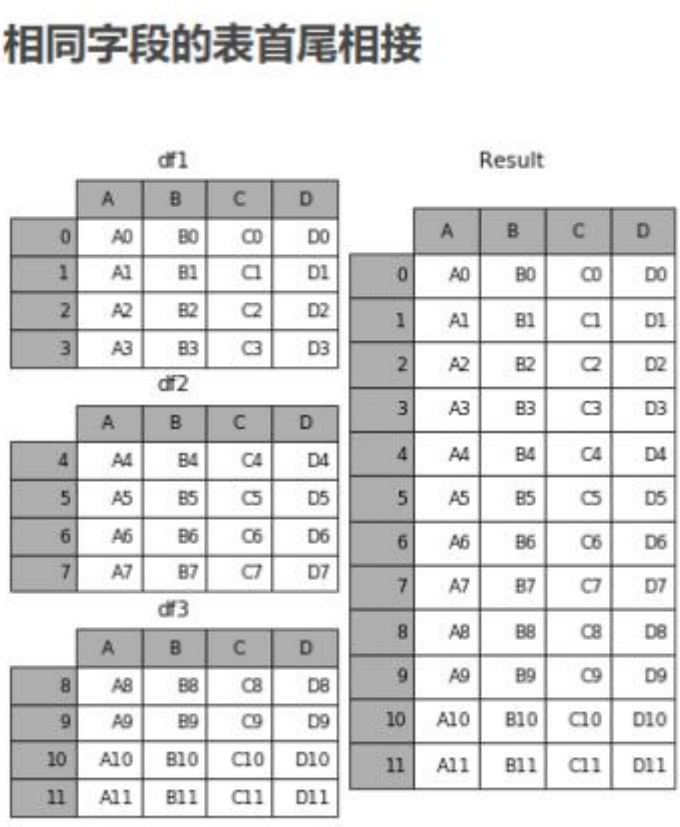



```
数据 4 = pd.concat([数据 1,数据 2],axis=1,sort =True) # sort=Ture 是默认的， pandas 总是默认
```

```
index 排序，默认 axis=0
print(数据 4)

axis=0 表示纵向合并（沿着 0 轴方向），
axis=1 表示横向合并（沿着 1 轴方向）
```

```
     0    1
A  0.0  NaN
B  1.0  NaN
C  2.0  NaN
D  NaN  3.0
E  NaN  4.0
```

6.2.1 首尾相接



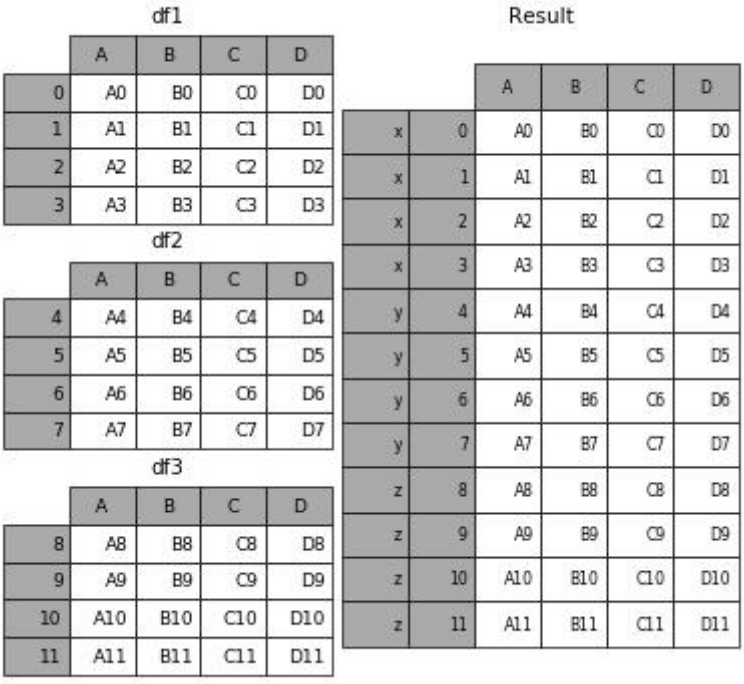
	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

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

# 要在相接的时候在加上一个层次的 key 来识别数据源自于哪张表，可以增加 key 参数
result = pd.concat(frames, keys=['x', 'y', 'z'])
```

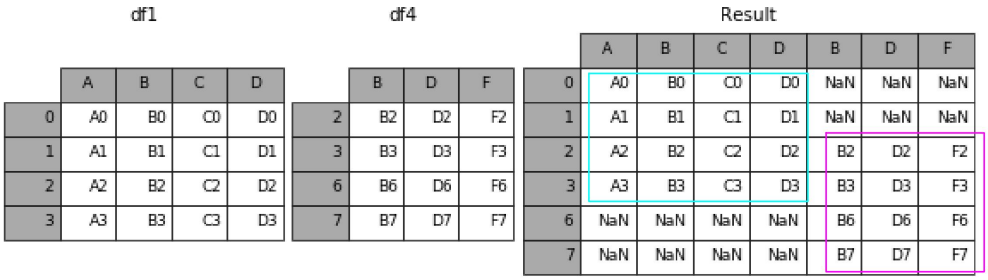


6.2.2 横向表拼接（行对齐）

（1）axis

当 axis = 1 的时候，concat 就是行对齐，然后将不同列名称的两张表合并

```
result = pd.concat([df1, df4], axis=1)
```



（2）join

加上 join 参数的属性，如果为'inner'得到的是两表的交集，如果是 outer，得到的是两表的并集。

```
result = pd.concat([df1, df4], axis=1, join='inner')
```

df1

	A	B	C	D
	A0	B0	C0	D0
	A1	B1	C1	D1
	A2	B2	C2	D2
	A3	B3	C3	D3

df4

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

Result

	A	B	C	D	B	D
2	A2	B2	C2	D2	B2	D2
3	A3	B3	C3	D3	B3	D3

6.2.3 append

append 是 series 和 dataframe 的方法，使用它就是默认沿着列进行凭借（axis = 0，列对齐）

result = df1.append(df2)

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

Result

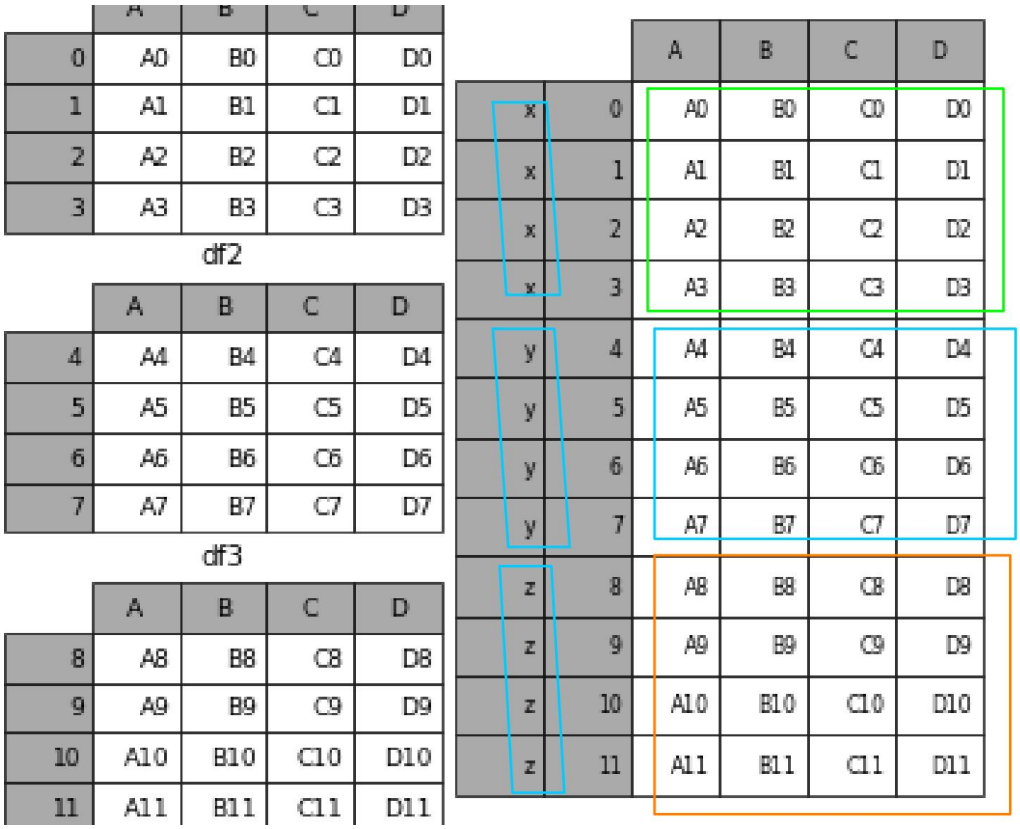
	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

6.2.4 合并的同时增加区分数据组的键

前面提到的 keys 参数可以用来给合并后的表增加 key 来区分不同的表数据来源

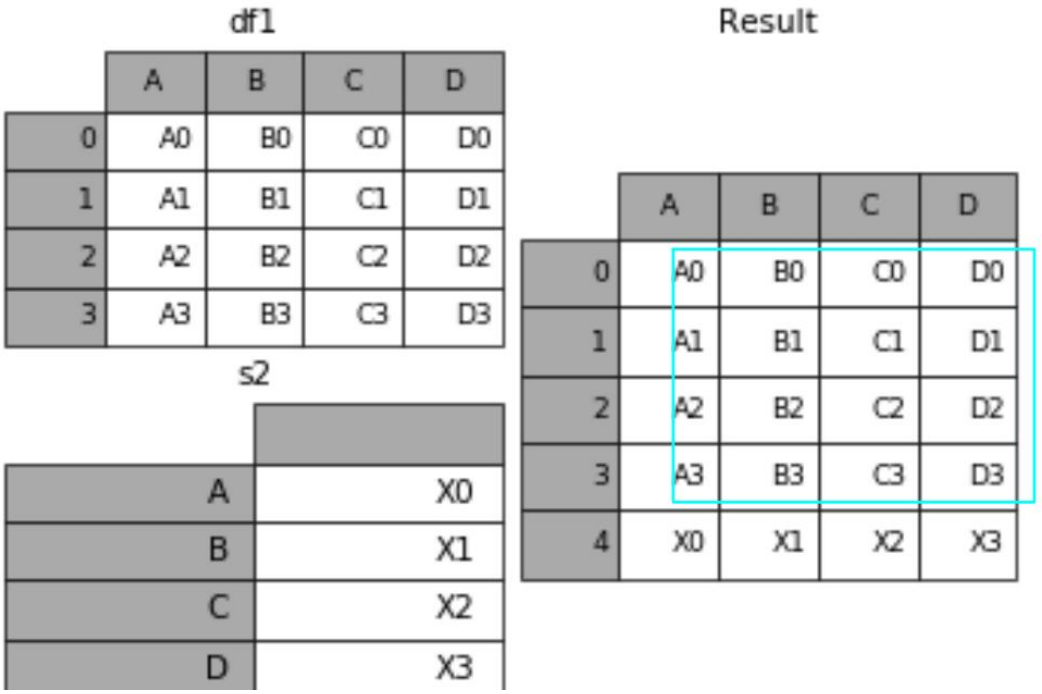
（1）可以直接用 key 参数实现

result = pd.concat(frames, keys=['x', 'y', 'z'])



6.2.5 在 dataframe 中加入新的行

append 方法可以将 series 和 字典就够的数据作为 dataframe 的新一行插入。



```
s2 = pd.Series(['X0', 'X1', 'X2', 'X3'], index=['A', 'B', 'C', 'D'])
result = df1.append(s2, ignore_index=True)
```

7.填充常用类型数据

pd.read_excel 参数

```
skiprows=行数 #跳过几行
usecols="区域" # 和 Excel 中一样，就是一个列的区域
index_col="字段名" # 将谁设置为索引
dtype={'序号':str,'性别':str,'日期':str} # 防止出错，把类型全指定为字符型
```

数据.at 的用法

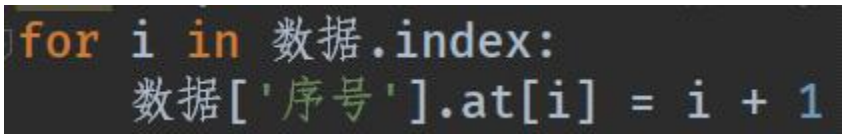
```
作用：获取某个位置的值，例如，获取第 0 行，第 a 列的值，即：index=0，columns='a'
变量名 = 数据.at[0, 'a']
```

日期模块 datetime

```
import pandas as pd
import datetime as 日期模块
路径 = 'c:/pandas/自动填充.xlsx'
起始日期 = 日期模块.date(2020,1,1)
读取数据 = pd.read_excel(路径,skiprows=2,usecols="B:E",index_col=None,dtype={'序号':str,'性别':str,'日期':str})
for i in 读取数据.index:
    读取数据['序号'].at[i] = i+1
    读取数据['性别'].at[i] = '男' if i%2 == 0 else '女'
    读取数据['日期'].at[i] = 起始日期 + 日期模块.timedelta(days=i) # timedelta 只能加天，小时，秒，毫秒
    读取数据['日期'].at[i] = 日期模块.date(起始日期.year+i,起始日期.month,起始日期.day) # 如果要在年上累加用 date
print(读取数据)
```

7.1 填充数字

```
import pandas as pd
路径 = 'c:/pandas/自动填充.xlsx'
数据 = pd.read_excel(路径,skiprows=8,usecols='F:I')
print(数据)
```



7.2 填充字符

```
import pandas as pd
路径 = 'c:/pandas/自动填充.xlsx'
数据 = pd.read_excel(路径,skiprows=8,usecols='F:I',dtype={'序号':str,'性别':str,'日期':str})
for i in 数据.index:
    数据['序号'].at[i] = i + 1
    数据['性别'].at[i] = '男' if i%2 == 0 else '女'
print(数据)
```

7.3 填充日期

```
import pandas as pd
import datetime as 日期模块
路径 = 'c:/pandas/自动填充.xlsx'
数据 = pd.read_excel(路径,skiprows=8,usecols='F:I',dtype={'序号':str,'性别':str,'日期':str})
起始日期 = 日期模块.date(2020,5,27)
for i in 数据.index:
    数据['序号'].at[i] = i + 1
    数据['性别'].at[i] = '男' if i%2 == 0 else '女'
    数据['日期'].at[i] = 起始日期+日期模块.timedelta(days=i)
print(数据)
```

如果只想累加年份，用 **Date**

```
import pandas as pd
import datetime as 日期模块
路径 = 'c:/pandas/自动填充.xlsx'
数据 = pd.read_excel(路径,skiprows=8,usecols='F:I',dtype={'序号':str,'性别':str,'日期':str})
起始日期 = 日期模块.date(2020,5,27)
for i in 数据.index:
    数据['序号'].at[i] = i + 1
    数据['性别'].at[i] = '男' if i%2 == 0 else '女'
    数据['日期'].at[i] = 起始日期+日期模块.timedelta(days=i)
    数据['日期'].at[i] = 日期模块.date(起始日期.year + i, 起始日期.month, 起始日期.day)
print(数据)
```

 # timedelta 只能加天，小时，秒，毫秒

7.4.计算列

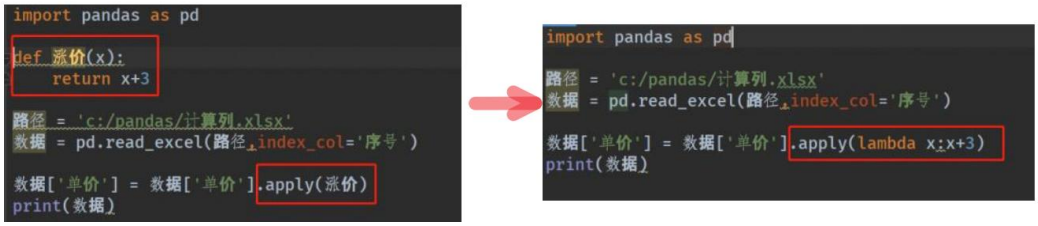
```
import pandas as pd
路径 = 'c:/pandas/计算列.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
数据['销售金额']=数据['单价']*数据['销售数量']
print(数据)
```

当你不想全部都计算，只想计算一部分行的时候，需要使用 For 循环

```
import pandas as pd
路径 = 'c:/pandas/计算列.csv'
数据 = pd.read_csv(路径,index_col='序号')
for i in range(1,3):
   数据['销售金额'].at[i] = 数据['单价'].at[i] * 数据['销售数量'].at[i]
print(数据)
```

8. pandas apply() 函数

理解 pandas 的函数，要对函数式编程有一定的概念和理解。函数式编程，包括函数式编程思维，当然是一个很复杂的话题，但对今天介绍的 apply() 函数，只需要理
解：函数作为一个对象，能作为参数传递给其它参数，并且能作为函数的返回值。
pandas 的 apply() 函数可以作用于 Series 或者整个 DataFrame，功能也是自动遍历整个 Series 或者 DataFrame, 对每一个元素运行指定的函数



Series.apply()：民族为少数民族的加 5 分

```
import pandas as pd
路径 = 'c:/pandas/apply 函数.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
数据['加分']=数据['民族'].apply(lambda x:5 if x != '汉' else 0)
数据['最终分数'] = 数据['总分']+数据['加分']
print(数据)
```

apply() 函数当然也可执行 python 内置的函数，比如我们想得到 Name 这一列字符的个数，如果用 apply() 的话：

```
数据['姓名字符个数']=数据['姓名'].apply(len)
```

DataFrame.apply() 函数则会遍历每一个元素，对元素运行指定的 function。比如下面的示例：计算数组的平方根

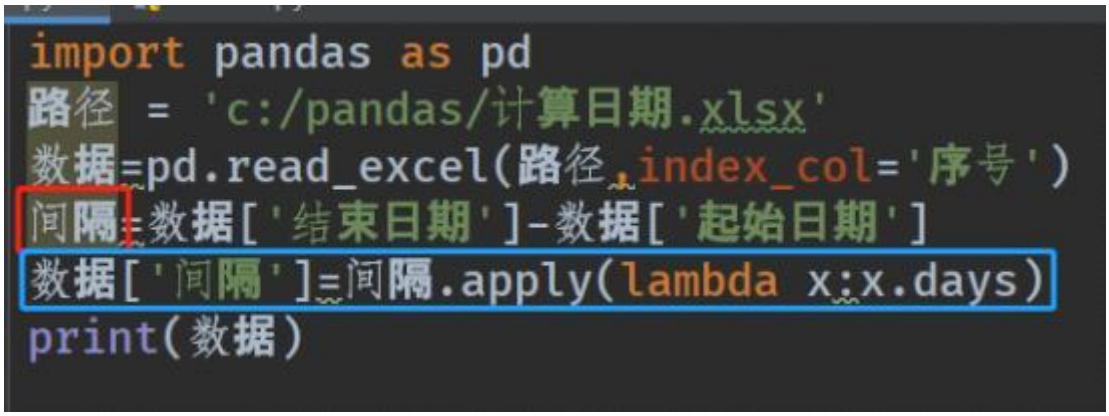
```
import pandas as pd
import numpy as np
arr = [
```

```
 [1,2,3],
 [4,5,6],
 [7,8,9]
]
数据=pd.DataFrame(arr,columns=['x','y','z'],index=['a','b','c'])
print(数据.apply(np.square)

如果只想 apply() 作用于指定的行和列，可以用行或者列的 name 属性进行限定。比如下面的示例将 x 列进行平方运算：
数据 2 = 数据.apply(lambda a : np.square(a) if a.name=='x' else a)
print(数据 2)

对 x 和 y 列进行平方运算：
数据 3 = 数据.apply(lambda a : np.square(a) if a.name in ['x','y'] else a)
print(数据 3)

第一行 （a 标签所在行）进行平方运算：
数据 4 = 数据.apply(lambda m : np.square(m) if m.name == 'a' else m, axis=1)
print(数据 4)

第一行和第三行（a 标签和 c 标签所在行）进行平方运算：
数据 4 = 数据.apply(lambda m : np.square(m) if m.name in ['a','c'] else m, axis=1)
print(数据 4)

默认情况下 axis=0 表示按列，axis=1 表示按行。
```

8.1 apply 函数计算日期相减

平时我们会经常用到日期的计算，比如要计算两个日期的间隔，比如下面的一组关于起止日期的数据：

序号	起始日期	结束日期
1	2020/1/1	2011/1/8
2	2020/3/1	2020/9/7
3	2020/5/3	2020/8/8
4	2020/4/8	2020/11/8
5	2020/7/30	2021/9/3



9.排序

```
## 参数
DataFrame.sort_values(by, axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
#### 参数说明
axis：如果 axis=0，那么 by="列名"；如果 axis=1，那么 by="行号"；
ascending:True 则升序，可以是[True,False]，即第一字段升序，第二个降序
inplace=True：不创建新的对象，直接对原始对象进行修改；
inplace=False：对数据进行修改，创建并返回新的对象承载其修改结果。
```

例 1：按语文分数降序排列

```
import pandas as pd
路径 = 'c:/pandas/排序.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
数据.sort_values(by='语文',inplace=True,ascending=False)
print(数据)
```

例 2：按语文分数排序降序，数学升序，英语降序

```
import pandas as pd
路径 = 'c:/pandas/排序.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
数据.sort_values(by=['语文','数学','英语'],inplace=True,ascending=[False,True,False])
print(数据)
```

例 3：按索引进行排序

```
import pandas as pd
路径 = 'c:/pandas/排序.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
数据.sort_index(inplace=True)
print(数据)
```

10.查询数据【loc】

10.1 单条件查询

```
import pandas as pd
路径 = 'c:/pandas/筛选.xlsx'
数据 = pd.read_excel(路径,index_col='出生日期')
print(数据.loc['1983-10-27','语文'])
```

10.2 多条件查询

```
import pandas as pd
路径 = 'c:/pandas/筛选.xlsx'
数据 = pd.read_excel(路径,index_col='出生日期')
print(数据.loc['1983-10-27',['语文','数学','英语']])
```

10.3 使用数据区域范围查询

```
import pandas as pd
路径 = 'c:/pandas/筛选.xlsx'
数据 = pd.read_excel(路径,index_col='出生日期')
print(数据.loc['1983-10-27':'1990-12-31',['语文','数学','英语']])
```

10.4 通过条件表达式查询

```
 import pandas as pd
路径 = 'c:/pandas/筛选.xlsx'
数据 = pd.read_excel(路径,index_col='出生日期')
print(数据.loc[(数据['语文'] > 60) & (数据['英语'] < 60),:])
这里的 ,: 指的是列取全部
```

11.筛选

```
一、加载数据
import pandas as pd
```

```
路径 = 'c:/pandas/筛选.xlsx'
数据 = pd.read_excel(路径,index_col='序号',sheet_name='Sheet1')
print(数据)

二、按位置筛选，筛选第 2 行至第 4 行数据
import pandas as pd
路径 = 'c:/pandas/筛选.xlsx'
数据 = pd.read_excel(路径,index_col='序号',sheet_name='Sheet1')
数据 2=数据.loc[2:4]
print(数据 2)

三、按值过滤，筛选所有男性
import pandas as pd
路径 = 'c:/pandas/筛选.xlsx'
数据 = pd.read_excel(路径,index_col='序号',sheet_name='Sheet1')
条件 = 数据['性别'] == '男'
print(数据[条件])

四、多条件筛选，男性和总分大于等于 150
import pandas as pd
路径 = 'c:/pandas/筛选.xlsx'
数据 = pd.read_excel(路径,index_col='序号',sheet_name='Sheet1')
条件 = "性别 == '男' and 总分 >= 150"
print(数据.query(条件))

query 方法，可以直接接受一个查询字符串，是不是很像 Sql 呢.指定多个值也很简单,使用 in 或 not in，如下：
import pandas as pd
路径='c:/pandas/筛选.xlsx'
数据=pd.read_excel(路径,index_col='序号',sheet_name='Sheet1')
条件="姓名 in ['王松','王刚']"
print(数据.query(条件))
```

11.1 文本筛选:开头和结尾

```
例：姓名列开头姓王的
import pandas as pd
路径 = 'c:/pandas/筛选.xlsx'
数据 = pd.read_excel(路径,index_col='序号',sheet_name='Sheet1')
条件 = 数据['姓名'].str.startswith('王') # 如果是结尾，就改成 endwith
print(数据[条件])
```

11.2 文本筛选：包含

```
例 1：筛选地址包含信阳市
```

```
import pandas as pd
路径 = 'c:/pandas/筛选.xlsx'
数据 = pd.read_excel(路径,index_col='序号',sheet_name='Sheet1')
条件 = 数据['地址'].str.contains('信阳市')
print(数据[条件])
```

11.3 筛选值的范围

```
例 1：语文分数在 60 至 100 之间的女性
import pandas as pd
路径 = 'c:/pandas/筛选.xlsx'
数据 = pd.read_excel(路径,index_col='序
号',sheet_name='Sheet1')
条件 = " 60 <= 语文 <= 100 and 性别 == '女'"
print(数据.query(条件))
```

11.4 筛选日期

11.4.1 获取某年某月数据

```
import pandas as pd
import datetime as dt
路径 = 'c:/pandas/筛选.xlsx'
# 将出生日期设置为索引 将出生日期转成日期格式
数据 = pd.read_excel(路径,index_col='出生日期',parse_dates=['出生日期'])
print(数据['1989'].head())
print(数据['1983-10'].head())
```

11.4.2 获取某个时期之前或者之后的数据

```
import pandas as pd
import datetime as dt
路径 = 'c:/pandas/筛选.xlsx'
数据 = pd.read_excel(路径,index_col='出生日期',parse_dates=['出生日期'])
数据 2 = 数据.sort_values('出生日期')
# 获取某个时期之前或之后的数据
# 获取 1980 年以后的数据
print(数据 2.truncate(before='1980').head())
# 获取 1990-12 之前的数据
print(数据 2.truncate(after='1990-12').head())
# 获取 1990-02 年以后的数据
print(数据 2.truncate(before='1990-02').head())
# 获取 1984-01-01 年以后的数据
print(数据 2.truncate(before='1984-01-1').head())
# 获取指定时间区间
print(数据 2['1983':'1990'])
print(数据 2['1983-01-1':'1990-12-31'])
```

12 处理缺失值

12.1 数据删除 drop 函数

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。 默认按行删除
levels	接收int或者索引名。代表标签所在级别。默认为None。
inplace	接收boolean。代表操作是否对原数据生效。默认为False。

12.1.1 drop 函数:删除行，删除列

```
删除行：
import pandas as pd
路径 = 'c:/pandas/删除.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
print(数据.drop(2)) # 删除单行，直接写行标签
print(数据.drop(labels=[1,3])) # 删除多行，使用 labels，标签写成列表

删除列：
import pandas as pd
路径 = 'c:/pandas/删除.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
print(数据.drop('语文',axis=1)) # 删除单列
print(数据.drop(labels=['语文','数学'],axis=1)) # 删除多列
```

注意：凡是会对原数组作出修改并返回一个新数组的，往往都有一个 inplace 可选参数。如果手动设定为 True（默认为 False），那么原数组直接就被替换。
而采用 inplace=False 之后，原数组名对应的内存值并不改变，需要将新的结果赋给一个新的数组或者覆盖原数组的内存位置。

12.2 查看缺失值

```
import pandas as pd
路径 = 'c:/pandas/删除.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
print(数据.isnull()) # 是缺失值就显示为 T
print(数据.notnull()) # 不是缺失值就显示为 T
```

12.2.1 去掉含有缺失值的行或者列

```
import pandas as pd
路径 = 'c:/pandas/删除.xlsx'
```

```
数据 = pd.read_excel(路径,index_col='序号')
print(数据.dropna()) # 删除有空值的行
print(数据.dropna(axis=1)) # 删除有空值的列
print(数据.dropna(how='all')) # 删除所有值为 Nan 的行
print(数据.dropna(thresh=2)) # 至少保留两个非缺失值
print(数据.dropna(subset=['语文','数学'])) # 在哪些列表中查看
```

12.2.2 填充常数

```
import pandas as pd
路径 = 'c:/pandas/删除.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
print(数据.fillna(0)) # 用常数填充
print(数据.fillna({'语文':0.1,'数学':0.2,'英语':0.3})) # 通过字典填充不同的常数
```

12.2.3 填充方式

```
import pandas as pd
路径 = 'c:/pandas/填充.xlsx'
数据 = pd.read_excel(路径)
print(数据.fillna(method='ffill'))
```

ffill	用前面的值填充
bfill	用后面的值填充
pad	向后填充
backfill	向前填充

13 数学统计函数

13.1 数学统计函数表

Function	Description	描述
count	Number of non-null observations	观测值的个数
sum	Sum of values	求和
mean	Mean of values	求平均值
mad	Mean absolute deviation	平均绝对方差
median	Arithmetic median of values	中位数
min	Minimum	最小值
max	Maximum	最大值
argmin	Calculate the index position (integer) that can get the minimum value	计算能够获取到最小值的索引位置（整数）
argmax	Calculate the index position where the maximum value can be obtained	计算能够获取到最大值的索引位置
idxmin	Row index of each column minimum	每列最小值的行索引
idxmax	Row index of the maximum value per column	每列最大值的行索引
mode	Mode	众数
abs	Absolute Value	绝对值
prod	Product of values	乘积
std	Bessel-corrected sample standard deviation	标准差
var	Unbiased variance	方差
sem	Standard error of the mean	标准误
skew	Sample skewness (3rd moment)	偏度系数
kurt	Sample kurtosis (4th moment)	峰度
quantile	Sample quantile (value at %)	分位数
cumsum	Cumulative sum	累加
cumprod	Cumulative product	累乘
cummax	Cumulative maximum	累最大值
cummin	Cumulative minimum	累最小值
cov()	covariance	协方差
corr()	correlation	相关系数
rank()	rank by values	排名
pct_change()	time change	时间序列变化

13.2 describle 数据值列汇总

所有数值列

```
import pandas as pd
路径 = 'c:/pandas/数据统计.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
print(数据.describe())
```

只看一列

```
import pandas as pd
```

```
路径 = 'c:/pandas/数据统计.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
print(数据['语文'].describe())
```

统计值变量说明：

count：数量统计，此列共有多少有效值
unipue：不同的值有多少个
std：标准差
min：最小值
25%：四分之一分位数
50%：二分之一分位数
75%：四分之三分位数
max：最大值
mean：均值

14 重复数据的处理

```
import pandas as pd
路径 = 'c:/pandas/去重.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
print(数据['姓名'].unique()) # 唯一值，以一个列表出现
print(数据['姓名'].value_counts()) # 姓名出现过几次
```

14.1 删除重复值

删除重复的方法：
DataFrame.drop_duplicates(subset=None, keep='first', inplace=False)
参数
keep：指定处理重复值的方法：
first：保留第一次出现的值
last：保留最后一次出现的值
False：删除所有重复值，留下没有出现过重复的
subset：用来指定特定的列，默认是所有列
inplace：是直接在原来数据上修改还是保留一个副本

```
import pandas as pd
路径 = 'c:/pandas/去重.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
print(数据.drop_duplicates(subset=['姓名'],keep='first'))
```

13.2 提取重复

DataFrame.duplicated(subset=None, keep='first')
参数
keep：指定处理重复值的方法：
first：保留第一次出现的值
last：保留最后一次出现的值
False：删除所有重复值，留下没有出现过重复的
subset：用来指定特定的列，默认是所有列
inplace：是直接在原来数据上修改还是保留一个副本

```
import pandas as pd
路径 = 'c:/pandas/去重.xlsx'
数据 = pd.read_excel(路径,index_col='序号')
# print(数据.duplicated()) # 判断重复行
# print(数据.duplicated(subset='姓名')) # 判断某列重复数据
重复 = 数据.duplicated(subset='姓名')
print(数据[重复]) # 提取重复
```

15 算术运算与数据对齐

算数运算无非就是加减乘除，但是需要注意 2 点：
1.空值与数字进行计算，结果是空值！
2.对除数为 0 的处理：
1/0 = inf 无穷大
-1/0 = -inf 负无穷大
0/0 = Nan

15.1 处理空值

```
import pandas as pd
路径 = 'c:/pandas/计算.xlsx'
数据 = pd.read_excel(路径)
结果 = 数据['1 店'] + 数据['2 店']
print(结果)
注意: 无论加减乘除，结果都是 空值与数字计算 等于 空值
```

方法一: 将空值填充为 0

```
import pandas as pd
路径 = 'c:/pandas/计算.xlsx'
数据 = pd.read_excel(路径)
结果 = 数据['1 店'].fillna(0) + 数据['2 店'].fillna(0)
print(结果)
```

15.2 处理 inf 无穷大

```
import pandas as pd
pd.options.mode.use_inf_as_na = True
```

```
路径 = 'c:/pandas/无穷大.xlsx'
数据 = pd.read_excel(路径)
结果 = 数据['1 店'].div(数据['2 店'],fill_value=0)
print(结果)
```

16. 数据替换

replace():替换

语法：字符串序列.replace(旧子串，新子串，替换次数)

注意：替换数次如果超过了子串的出现次数，就替换所有子串。省略替换次数就是全部替换。

```
变量名 = "你好，世界！^ 你好，中国！^ 我们一起学习中国话！^ 中国加油！"
print(变量名.replace('中国','祖国'))
print(变量名.replace('中国','祖国',2))
print(变量名.replace('中国','祖国',10))
Print(变量名)
```

观察一下，字符串被修改了吗？

16.1 替换全部或者某一行

```
一、整个表全部替换
import pandas as pd
路径 = 'c:/pandas/替换.xlsx'
数据 = pd.read_excel(路径)
数据.replace('城八区','海淀区',inplace=True)
print(数据)

二、某一行替换
import pandas as pd
路径 = 'c:/pandas/替换.xlsx'
数据 = pd.read_excel(路径)
数据['城市 2'].replace('城八区','海淀区',inplace=True)
print(数据)
```

16.2 替换指定的某个或者多个数值(用字典的形式)

```
import pandas as pd
路径 = 'c:/pandas/替换.xlsx'
数据 = pd.read_excel(路径)
字典 = {'A':20,'B':30}
```

```
数据.replace(字典,inplace=True)
print(数据)
```

这个很好理解，就是字典里的建作为原值，字典里的值作为替换的新值

```
也可以用列表的方式：
import pandas as pd
路径 = 'c:/pandas/替换.xlsx'
数据 = pd.read_excel(路径)
数据.replace(['A','B'],[20,30],inplace=True)
print(数据)
```

16.3 替换某个数据部分内容

```
import pandas as pd
路径 = 'c:/pandas/替换.xlsx'
数据 = pd.read_excel(路径)
数据['城市'] = 数据['城市'].str.replace('城八','市')
print(数据)
```

16.4 正则表达式替换

```
import pandas as pd
路径 = 'c:/pandas/替换.xlsx'
数据 = pd.read_excel(路径)
数据.replace('[A-Z]',88,regex=True,inplace=True)
print(数据)
```

17 离散化和分箱

在机械学习中，我们经常会对数据进行分箱处理的操作， 也就是 把一段连续的值切分成若干段，每一段的值看成一个分类。这个把连续值转换成离散值的过程，我们叫做分箱处理。

比如，把年龄按 15 岁划分成一组，0-15 岁叫做少年，16-30 岁叫做青年，31-45 岁叫做壮年。在这个过程中，我们把连续的年龄分成了三个类别，“少年”，“青年”和“壮年”。就是各个类别的名称，或者叫做标签。

cut 和 qcut 函数的基本介绍

在 pandas 中，cut 和 qcut 函数都可以进行分箱处理操作。其中 cut 函数是按照数据的值进行分割，而 qcut 函数则是根据数据本身的数量来对数据进行分割。

17.1 指定分界点分箱【cut】

Python 实现连续数据的离散化处理主要基于两个函数：

pandas.cut 和 pandas.qcut，

pandas.cut 根据指定分界点对连续数据进行分箱处理

pandas.qcut 可以指定箱子的数量对连续数据进行等宽分箱处理

（注意：所谓等宽指的是每个箱子中的数据量是相同的）

```
import pandas as pd
```

```
年份 = [1992, 1983, 1922, 1932, 1973] # 待分箱数据
箱子 = [1900, 1950, 2000] # 指定箱子的分界点
结果 = pd.cut(年份, 箱子)
print(结果)
# 结果说明：其中(1950, 2000]说明【年份】列表的第一个值 1992 位于(1950, 2000]区间
```

```
[(1950, 2000], (1950, 2000], (1900, 1950], (1900, 1950], (1950, 2000]]
Categories (2, interval[int64]): [(1900, 1950] < (1950, 2000]]
```

print(pd.value_counts(结果)) # 对不同箱子中的数进行计数

```
(1950, 2000]    3
(1900, 1950]    2
dtype: int64
```

结果 2 = pd.cut(年份, 箱子,labels=False)

```
[1 1 0 0 1]
```

 # 结果说明：其中 1 说明【年份】列表的第一个值 1992 位于(1950, 2000]区间
 # 其中 0 说明【年份】列表的第一个值 1922 位于(1900, 1950]区间

```
import pandas as pd
年份 = [1992, 1983, 1922, 1932, 1973] # 待分箱数据
箱子 = [1900, 1950, 2000] # 指定箱子的分界点
# 可以将想要指定给不同箱子的标签传递给 labels 参数
箱子名称 = [ '50 年代前', '50 年代后']
结果 3 = pd.cut(年份, 箱子, labels=箱子名称)
print(pd.value_counts(结果 3))
```

```
50年代后    3
50年代前    2
dtype: int64
```

17.2 等频分箱

```
import pandas as pd
年份 = [1992, 1983, 1922, 1932, 1973, 1999, 1993, 1995] # 待分箱数据
结果 = pd.qcut(年份,q=4) # 参数 q 指定所分箱子的数量
# 从输出结果可以看到每个箱子中的数据量时相同的
print(结果)
print(pd.value_counts(结果)) # 从输出结果可以看到每个箱子中的数据量时相同的
```

18 字符串操作

在使用 **pandas** 的时候，经常要对 **DataFrame** 的某一列进行操作，一般都会使用 **df["xx"].str** 下的方法，但是都有哪些方法呢？我们下面来罗列并演示一下。
既然是 **df["xx"].str**，那么 **xx** 这一列必须是字符串类型，当然在 **pandas** 里面是 **object**，不能是整型、时间类型等等。如果想对这些类型使用的话，必须先 **df["xx"].astype(str)**转化一下，才能使用此方法。
例如：**36℃**把 **36** 分割出来，并转成整形
数据['温度'].str.replace('℃','').astype('int64')

18.1 字符串对象方法

18.1.1 cat 和指定字符进行拼接

```
import pandas as pd
路径 = 'c:/pandas/字符串.xlsx'
数据 = pd.read_excel(路径)
print(数据['姓名'].str.cat()) # 不指定参数，所有姓名拼接
print(数据['姓名'].str.cat(sep='、'))
print(数据['姓名'].str.cat(['变身'] * len(数据)))
# ['变身'] * len(数据) 相当于 ['变身'] * 6 次
print(数据['姓名'].str.cat(['变身'] * len(数据),sep='^'))
print(数据['职业'].str.cat(['快跑'] * len(数据),sep='^',na_rep='@'))
# 如果一方为 NaN,结果也为 NaN,因此我们可以指定 na_rep,表示将 NaN 用 na_rep 替换

数据['合并 1']=数据['姓名']+数据['性别']+数据['身份']
# 等价于以下语句,可以加 sep 参数分隔符
#但是注意哦，只能应用于 series，不能使数据 Dateframe！
数据['合并 2']=数据['姓名'].str.cat(数据['性别'],sep=',').str.cat(数据['身份'],sep=',')
print(数据)
```

18.1.2 split 按照指定字符串分割

```
import pandas as pd
路径 = 'c:/pandas/字符串.xlsx'
数据 = pd.read_excel(路径)
print(数据['状态'].str.split()) # 不指定分隔符，就是一列表
print(数据['状态'].str.split('血')) # 和 python 内置 split 一样
print(数据['状态'].str.split('血',n=-1)) # 指定 n，表示分隔次数，默认是-1，全部分隔
print(数据['状态'].str.split('血',expand=True))
# 注意这个 expand，默认是 False，得到是一个列表
# 如果指定为 True，会将列表打开，变成多列，变成 DATAFrame
# 列名则是按照 0 1 2 3…的顺序，并且默认 Nan 值分隔后还是为 Nan
# 如果分隔符不存在，还是返回 DATAFrame
```

18.1.3 get 获取指定位置的字符，只能获取一个

```
import pandas as pd
路径 = 'c:/pandas/字符串.xlsx'
数据 = pd.read_excel(路径)
print(数据['状态'].str.get(2)) # 获取指定索引的字符，只能传入 int
```

18.1.4 slice 获取指定范围的字

```
import pandas as pd
路径 = 'c:/pandas/字符串.xlsx'
数据 = pd.read_excel(路径)
print(数据['状态'].str.slice(0)) # 指定一个值的话，相当于[m:]
print(数据['状态'].str.slice(0,3)) # 相当于[m:n],从 0 开始不包括 3
print(数据['状态'].str.slice(0,3,2)) # 相当于[m: n: step]
print(数据['状态'].str.slice(5,9,2)) # 索引越界，默认为空字符串，原来 Nan 还是 Nan
```

18.1.5 join 将每个字符之间使用指定字符串相连

```
import pandas as pd
路径 = 'c:/pandas/字符串.xlsx'
数据 = pd.read_excel(路径)
print(数据['状态'].str.join('a'))
```

19. Excel 文件的拆分与合并

19.1 一个文件夹下多个工作簿的合并【单独 Sheet】

```
1，把文件夹下面所有的文件都遍历出来
2、循环读取每个文件
（1）第一次读取的文件放入一个空的表中，起名叫合并表
（2）从第二次开始每次都与这个合并表进行合并
3、写入 Excel
4、所有表表头行数要一至，通过 header=1 进行设置
```

```
import pandas as pd
import os
路径 = 'c:/pandas/课件 025-2/'
合并表 = pd.DataFrame()
for 文件名 in os.listdir(路径):
   表格 = pd.read_excel(路径 + 文件名,header=1)
   合并表 =pd.concat([合并表,表格])
print(合并表)
```

```
保存：
路径 2 = 'c:/合并表.xlsx'
合并表.to_excel(路径 2)
```

19.2 同一工作簿中多个 Sheet 合并

```
import pandas as pd
路径 = 'C:/pandas/合并 2.xlsx'
数据 = pd.read_excel(路径,None)
合并表 = pd.DataFrame()
字段名 = list(数据.keys()) # 返回['序号', '姓名', '电话']
for 列标签 in 字段名:
    新数据 = 数据[列标签]
    合并表 = pd.concat([合并表,新数据])
print(数据)
```

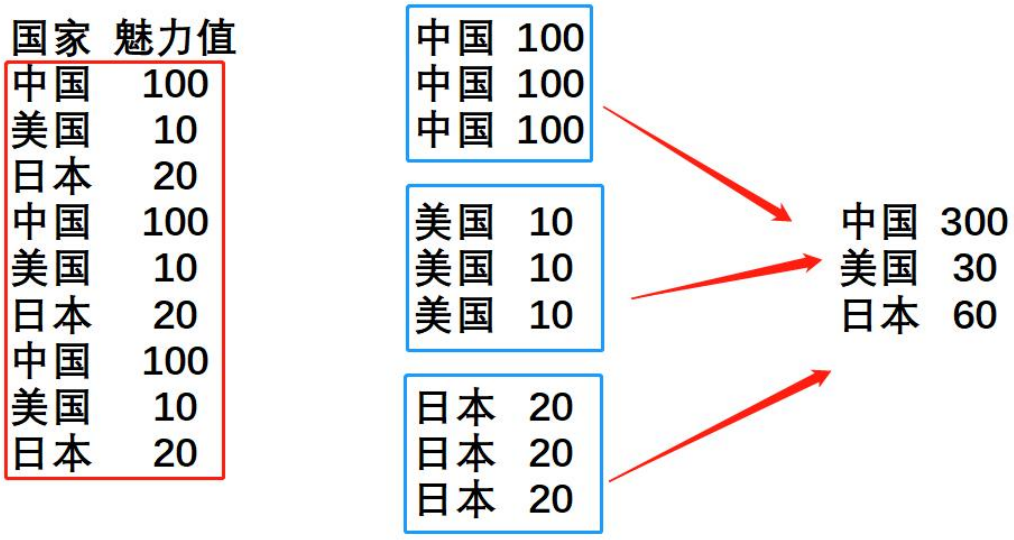
19.3 将一个工作表拆分成多个工作表

```
import pandas as pd
路径 = 'c:/pandas/拆分.xlsx'
数据 = pd.read_excel(路径)
分割列 = list(数据['部门'].drop_duplicates()) # 返回：['办公室', '销售部', '保洁部']
新数据 = pd.ExcelWriter('c:/多个 Sheet.xlsx')
for i in 分割列:
    数据 1 = 数据[数据['部门'] == i]
    数据 1.to_excel(新数据,sheet_name=i)
新数据.save()
新数据.close()
```

19.4 将一个工作表拆分成多个工作簿

```
import pandas as pd
路径 = 'c:/pandas/拆分.xlsx'
数据 = pd.read_excel(路径)
分割列 = list(数据['部门'].drop_duplicates()) # 返回：['办公室', '销售部', '保洁部']
for i in 分割列:
    数据 1 = 数据[数据['部门'] == i]
    数据 1.to_excel('c:/'+ i + '.xlsx')
```

20. 分组与聚合 groupby



groupby 分为三个步骤：拆分-应用-合并
DataFrame 可以在其行（axis=0）或列（axis=1）上进行分组。
然后，将一个函数应用到各个分组并产生新值。
最后，所有这些函数的执行结果会被合并到最终的结果对象中去。
GroupBy 的 size 方法可以返回一个含有分组大小的 Series。

20.1 对分组进行迭代【遍历各分组】

```
import pandas as pd
路径 = 'c:/pandas/分组聚合.xlsx'
数据 = pd.read_excel(路径)
for (i,j),group in 数据.groupby(['城市','区']):
    print(i) # 城市分组
    print(j) # 区分组
    print(group) # 城市和区的分组
```

20.2 分组聚合

```
import pandas as pd
路径 = 'c:/pandas/分组聚合.xlsx'
数据 = pd.read_excel(路径)
数据 2 = 数据.groupby(['城市','区'])[['人数']].sum()
print(数据 2)
数据 2.to_excel('c:/1.xlsx')

import pandas as pd
路径 = 'c:/pandas/分组聚合.xlsx'
数据 = pd.read_excel(路径)
数据 2 = 数据.groupby(['城市','区'])[['人数','金额']].sum()
```

```
print(数据 2)
数据 2.to_excel('c:/1.xlsx')

import pandas as pd
路径 = 'c:/pandas/分组聚合 2.xlsx'
数据 = pd.read_excel(路径,index_col='店号')
#不同列的不同计算方法
字典 = {'1 月':'count','2 月':sum,'3 月':max,'4 月':'mean'}
数据 2 = 数据.groupby('店号').agg(字典)
print(数据 2)
```

20.2.1 agg 函数

agg 函数一般与 groupby 配合使用，agg 是基于列的聚合操作，而 groupby 是基于行的

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

func : 函数，函数名称，函数列表，字典{‘行名/列名’，‘函数名’}

使用指定轴上的一个或多个操作进行聚合。

21.数据透视 pivot_table

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

pivot_table 有四个最重要的参数 index、values、columns、aggfunc。

在需要多个 group by 的时候,可以优先考虑此函数

index 需要聚合的列名，默认情况下聚合所有数据值的列

```
import pandas as pd
路径 = 'c:/pandas/透视.xlsx'
数据 = pd.read_excel(路径)
数据 2 = pd.pivot_table(数据,index=['部门','销售人员'])
print(数据 2)
```

values 在结果透视的行上进行分组的列名或其它分组键【就是透视表里显示的列】

```
import pandas as pd
路径 = 'c:/pandas/透视.xlsx'
数据 = pd.read_excel(路径)
数据 2 = pd.pivot_table(数据,index=['部门','销售人员'],values=['数量','金额'])
print(数据 2)
```

columns 在结果透视表的列上进行分组的列名或其它分组键

```
import pandas as pd
```

```
路径 = 'c:/pandas/透视.xlsx'
数据 = pd.read_excel(路径)
数据 2 = pd.pivot_table(数据,index=['部门','销售人员'],values=['数量','金额'],columns='所属区域')
print(数据 2)
```

Aggfunc 聚合函数或函数列表（默认情况下是 mean）可以是 groupby 里面的任意有效函数

```
import pandas as pd
import numpy as np
路径 = 'c:/pandas/透视.xlsx'
数据 = pd.read_excel(路径)
数据 2 = pd.pivot_table(数据,index=['部门','销售人员'],values=['数量','金额'],columns='所属区域',aggfunc=[sum,np.mean])
print(数据 2)
```