机器学习

5.Python Pandas模块

本课目标

- 掌握Pandas模块中的常用数据结构
- 掌握如何使用Pandas模块对数据进行处理及操作
- 掌握如何使用Pandas读取不同数据源

课程目录

Course catalogue

Pandas介绍

NumPy：提供多维数组的高效存储和处理方，更适合处理同质型的数值类型数组数据。

- 科学计算任务中，干净整齐、组织良好的数据

Pandas：在NumPy基础上建立的新程序库，用来处理表格型或异质型数据。

-为“数据清理”任务提供了捷径。

- import pandas as pd

数据结构介绍

两个重要的数据结构：序列Series、 数据框DataFrame。

Series类似于numpy中的一维数组，Series对象是一个带索引数据构成的一维数组。

除了通吃一维数组可用的函数或方法，而且其可通过索引标签的方式获取数据，还具有索引的自动对齐功能；

DataFrame类似于numpy中的二维数组，同样可以通用numpy数组的函数和方法，而且还具有其他灵活应用。

Series数据类型

序列创建：

```
#1.通过一维数组创建序列
data = pd.Series([0.25,0.5,0.75,1])
print(data)
print(type(data))

#values和index属性
print("values:",data.values)
print("Indexes:",data.index)

#数据获取：
print(data[1])
print(data[:3])
```

显然，Series对象将一维数组和一组索引绑定在一起。

可以通过values属性和index属性获取数据。

Vaues属性返回和numpy数组类似。

Index属性返回的是一个pd.Index对象类型的数组对象。

Series数据类型

Series是通用的NumPy数组：

```
#索引类型：字符串
data = pd.Series([0.25,0.5,0.75,1.0],
                 index = ['a','b','c','d'])
print(data)

#获取数值方式不变：
print(data['b'])
```

```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
0.5
```

Series是通用的NumPy数组：本质差别在于索引。

NumPy数组通过隐式定义的整数索引获取数值。Pandas的Series对用用显示定义的索引与数值关联。

Series索引可以是任意类型。

Series数据类型

通过字典创建Series：

```
#通过字典创建Series
code_dict = {'Beijing':'010','Shanghai':'021',
             'Tianjin':'022','Chongqing':'023'}
code = pd.Series(code_dict)
print(code)

#获取数值：
print(code['Shanghai'])
print(code['Beijing':'Tianjin'])
```

```
Beijing      010
Shanghai     021
Tianjin      022
Chongqing    023
dtype: object
021
Beijing     010
Shanghai    021
Tianjin     022
dtype: object
```

Series是特殊的字典，在一些操作上比字典更加高效

Series对象可以支持诸如切片等数组形式的操作，而字典不行

Series数据类型

创建Series：

pd.Series(data,index=*index*)

data可以是列表或NumPy数组，此时index默认值是整数序列：

如： pd.Series([2,4,6])

data也可以是标量，创建Series对象时，会重复填充到每个索引上：

```
pd.Series(5, index=[100, 200, 300])
```

```
100    5
200    5
300    5
dtype: int64
```

data是字典时，index默认是字典键

DataFrame对象

创建数据框DataFrame：

通过包含等长列表或NumPy数组的字典形式创建DataFrame：

```
#通过包含等长列表或NumPy数组的字典形式来创建
data = {'state':['ohio','ohio','ohio','Nevada','Nevada','Nevada'],
        'year':[2000,2001,2002,2001,2002,2003],
        'pop':[1.5,1.7,3.6,2.4,2.9,3.2]}
frame = pd.DataFrame(data)
print(frame)
```

```
    state  year  pop
0    ohio  2000  1.5
1    ohio  2001  1.7
2    ohio  2002  3.6
3  Nevada  2001  2.4
4  Nevada  2002  2.9
5  Nevada  2003  3.2
```

DataFrame表示的是矩阵的数据表。

产生的DataFram会自动为Series分配索引。

DataFrame既有行索引，也有列索引。

和Series一样，DataFrame既可以作为一个通用型的NumPy数组，也可以看做特殊的Python字典。

DataFrame对象

创建数据框DataFrame：

若指定了列的顺序，则按照指定顺序排列：

```
#通过包含等长列表或NumPy数组的字典形式来创建
data = {'state':['ohio','ohio','ohio','Nevada','Nevada','Nevada'],
        'year':[2000,2001,2002,2001,2002,2003],
        'pop':[1.5,1.7,3.6,2.4,2.9,3.2]}
frame = pd.DataFrame(data,columns=['year','state','pop'])
print(frame)
```

```
   year   state  pop
0  2000    ohio  1.5
1  2001    ohio  1.7
2  2002    ohio  3.6
3  2001  Nevada  2.4
4  2002  Nevada  2.9
5  2003  Nevada  3.2
```

DataFrame对象

创建数据框DataFrame：

若指定的列不在字典中，则结果中会出现缺失值，NA

```
#通过包含等长列表或NumPy数组的字典形式来创建
data = {'state':['ohio','ohio','ohio','Nevada','Nevada','Nevada'],
        'year':[2000,2001,2002,2001,2002,2003],
         'pop':[1.5,1.7,3.6,2.4,2.9,3.2]}
frame = pd.DataFrame(data,columns=['year','state','pop','debt'],
                           index=['one','two','three','four','five','six'])
print(frame)
```

```
       year   state  pop debt
one    2000    ohio  1.5  NaN
two    2001    ohio  1.7  NaN
three  2002    ohio  3.6  NaN
four   2001  Nevada  2.4  NaN
five   2002  Nevada  2.9  NaN
six    2003  Nevada  3.2  NaN
```

课程目录

Course catalogue

DataFrame对象基本属性

DataFrame基本属性：

函数	返回值
values	元素
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）

在开始统计分析之前，需要使用DataFrame的属性和方法对数据基本状况有一定的了解，才能依据数据的状况，进行相应的分析。

DataFrame对象数据查改增删

1.数据基本查看方式：

对某一列进行查看：

```
frame2['state']
```

```
one        Ohio
two        Ohio
three      Ohio
four     Nevada
five     Nevada
six      Nevada
Name: state, dtype: object
```

```
frame2.state
```

```
one        Ohio
two        Ohio
three      Ohio
four     Nevada
five     Nevada
six      Nevada
Name: state, dtype: object
```

访问DataFrame中某一列的某几行时，单独一列的DataFrame可以视为一个Series，而访问一个Series基本和访问一个一维的ndarray相同。

可以按照字典或属性形式检索DataFrame的一列，结果是个Series。（frame2.column只有在列名有效时可用）

查改增删DataFrame对象数据

1.数据基本查看方式：

访问多列，多行：

```
frame2[['year','state']]['one':'five']
```

	year	state
one	2000	Ohio
two	2001	Ohio
three	2002	Ohio
four	2001	Nevada
five	2003	Nevada

```
frame2[:]['one':'three']
```

访问多列数据时，可以将多个列索引名称视为一个列表。

同时访问DataFrame多列数据中的多行数据，和访问单列多行基本相同。

访问多行时，选所有列用：代替。

head和tail方法也用于检索连续多行数据，在方法的（）后填入需要访问的行数。

查改增删DataFrame对象数据

1.数据基本查看方式：

loc，iloc访问方式： *DataFrame.**loc**[行索引名称或条件, 列索引名称]*

*DataFrame.**iloc**[行索引位置, 列索引位置]*

```
frame2.loc['three']
```

```
year      2002
state     Ohio
pop        3.6
debt       NaN
Name: three, dtype: object
```

```
frame2.loc['three',['year','pop']]
```

```
year     2002
pop       3.6
Name: three, dtype: object
```

```
frame2.iloc[2,[0,2]]
```

```
year     2002
pop       3.6
Name: three, dtype: object
```

```
print(frame2.loc['one':'two','year':'state'])
print(frame2.iloc[:2,0:2])
```

```
     year state
one  2000  Ohio
two  2001  Ohio
     year state
one  2000  Ohio
two  2001  Ohio
```

索引符号loc和iloc允许用轴标签（loc）或整数标签（iloc）以NumPy风格的语法从DataFrame中选出数组的任意行列子集。支持切片。 ---更加灵活

iloc和loc区别是iloc接收的必须是行索引和列索引的位置。

在loc使用的时候内部传入的行索引名称如果为一个区间，则前后均为闭区间；iloc方法使用时内部传入的行索引位置或列索引位置为区间时，则为前闭后开区间。

查改增删DataFrame对象数据

1.数据基本查看方式：

loc，iloc访问方式：loc方式可以传入表达式，返回满足表达式的所有值。

```
frame2.loc[frame2['pop']>2,['state','pop']]
```

	state	pop
three	Ohio	3.6
four	Nevada	2.4
five	Nevada	2.9
six	Nevada	3.2

查改增删DataFrame对象数据

2.更改DataFrame中的数据：

```
frame2['debt']=16.5
print(frame2)
frame2.loc[:,'debt']='NaN'
print(frame2)
frame2.iloc[:,3]=np.arange(6.0)
print(frame2)
```

```
val = pd.Series([-1.2,-1.5,-1.7],index=['two','four','five'])
frame2['debt']=val
frame2
```

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2003	Nevada	2.9	-1.7
six	2003	Nevada	3.2	NaN

列的引用可以修改，赋值为标量值或者值数组。

当将列表或数组赋给一个列时，值得长度必须和DataFrame的长度相匹配。

将Series赋给一列时，按对应索引赋值。

被赋值列不存在，则新建列

查改增删DataFrame对象数据

3.添加DataFrame中的数据：

```
frame2['NewCol'] = 666
frame2
```

	year	state	pop	debt	NewCol
one	2000	Ohio	1.5	NaN	666
two	2001	Ohio	1.7	-1.2	666
three	2002	Ohio	3.6	NaN	666
four	2001	Nevada	2.4	-1.5	666
five	2003	Nevada	2.9	-1.7	666
six	2003	Nevada	3.2	NaN	666

DataFrame添加一列：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。

新增的一列值是相同的则直接赋值一个常量即可。

查改增删DataFrame对象数据

4.删除DataFrame中的数据：

删除某列或某行数据：drop方法：

drop(labels, axis=0, level=None, inplace=False, errors='raise')

常用参数如下：

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（0删除行，1删除列)。默认为0。
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。

```
frame2.drop(labels='NewCol',axis=1,inplace=True)
frame2.columns
```

```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

若删除行，则只需将drop方法参数中的‘labels’参数换成对应的行索引，将‘axis’设为0即可。

索引重建

重建索引：

reindex()方法是Pandas的重要方法，创建一个符合新索引的新对象

```
obj = pd.Series([4.5,7.2,-5.3,3.6],index=['d','b','a','c'])
print(obj)
obj2 = obj.reindex(['a','b','c','d','e'])
print(obj2)
```

```
d    4.5
b    7.2
a   -5.3
c    3.6
dtype: float64
a   -5.3
b    7.2
c    3.6
d    4.5
e    NaN
dtype: float64
```

数据会按照新的索引进行排序，若某个索引之前不存在，则引入缺失值NaN.

索引重建

重建索引：

reindex()方法是Pandas的重要方法，创建一个符合新索引的新对象

```
obj3 = pd.Series(['blue','purple','yellow'],index=[0,2,4])
obj3.reindex(range(6),method='ffill')
```

```
0      blue
1      blue
2    purple
3    purple
4    yellow
5    yellow
dtype: object
```

若重建时，需要进行插值或填充，method方法可允许我们使用植入ffill等在重建索引时插值。

ffill方法会将值向前填充。

索引重建

重建索引：

reindex()方法是Pandas的重要方法，创建一个符合新索引的新对象

```
frame = pd.DataFrame(np.arange(9).reshape((3,3)),
                     index = ['a','c','d'],
                     columns=['Ohio','Texas','California'])

#重建行索引
frame2 = frame.reindex(['a','b','c','d'])
#用columns关键字重建列索引
states = ['Texas','Utah','California']
frame.reindex(['a','b','c','d'],columns=states)
```

DataFrame中，reindex可以改变行索引、列索引，也可以同时改变二者。

仅传入一个序列，改变行索引。

列可以用columns关键字重建索引

描述分析DataFrame数据

1.数值型特征的描述性统计—— pandas描述性统计方法

方法名称	说明	方法名称	说明
min	最小值	max	最大值
mean	均值	ptp	极差
median	中位数	std	标准差
var	方差	cov	协方差
sem	标准误差	mode	众数
skew	样本偏度	kurt	样本峰度
quantile	四分位数	count	非NA值数目
describe	描述统计	mad	平均绝对离差

统计性描述用来概括、表述事物整体状况，以及事物间关联、类属关系的统计方法。

NumPy中提供了不少统计函数，Pandas基于NumPy，内建了处理缺失值的功能。

pandas提供了更加便利的方法来计算均值，如frame2['pop'].mean()。

描述分析DataFrame数据

1.数值型特征的描述性统计—— pandas描述性统计方法

```
df = pd.DataFrame([[1.4,np.nan],[7.1,-4.5],
                   [np.nan,np.nan],[0.75,-1.3]],
                   index=['a','b','c','d'],
                   columns=['one','two'])
print(df.sum())  #sum()返回一个包含列上加和的Series
print(df.sum(axis='columns')) #or axis=1
```

```
one    9.25
two   -5.80
dtype: float64
a     1.40
b     2.60
c     0.00
d    -0.55
dtype: float64
```

NA值会自动被排除

可用skipna=False 来实现不排除NA值。

```
df.sum(axis='columns',skipna=False)
```

```
a      NaN
b     2.60
c      NaN
d    -0.55
dtype: float64
```

描述分析DataFrame数据

1.数值型特征的描述性统计—— pandas描述性统计方法

```
frame2.describe()
```

	year	pop	debt
count	6.000000	6.000000	3.000000
mean	2001.666667	2.550000	-1.466667
std	1.211060	0.836062	0.251661
min	2000.000000	1.500000	-1.700000
25%	2001.000000	1.875000	-1.600000
50%	2001.500000	2.650000	-1.500000
75%	2002.750000	3.125000	-1.350000
max	2003.000000	3.600000	-1.200000

pandas还提供了一个方法叫作describe，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。

描述分析DataFrame数据

2.类别型特征的描述性统计

```
frame2['state'].value_counts()
```

```
Nevada    3
Ohio      3
Name: state, dtype: int64
```

```
frame2.loc['four','debt'] = -1.2
frame2['debt'] = frame2['debt'].astype('category')
frame2['debt'].describe()
```

```
count      3.0
unique     2.0
top       -1.2
freq       2.0
Name: debt, dtype: float64
```

描述类别型特征的分布状况，可以使用频数统计表。pandas库中实现频数统计的方法为value_counts()。

pandas提供了categories类，可以使用astype方法将目标特征的数据类型转换为category类别。

describe方法能够支持对category类型的数据进行描述性统计，四个统计量分别为列非空元素的数目，类别的数目，数目最多的类别，数目最多类别的数目。

课程目录

Course catalogue

读写不同数据源

1.数据库数据读取

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

读写不同数据源

1.数据库数据读取

（1）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)

（2）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)

（3）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)

读写不同数据源

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。

读写不同数据源

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	接收fail，replace，append。fail表示如果表名存在则不执行写入操作；replace表示如果存在，将原数据库表删除，再重新创建；append则表示在原数据库表的基础上追加数据。默认为fail。
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。

读写不同数据源

2. 文本读取

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）。

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='\t', header='infer', names=None, index_col=None, dtype=None, engine=None, nrows=None)*

读写不同数据源

2. 文本读取: 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。
nrows	接收int。表示读取前n行。默认为None。

读写不同数据源

2. 文本读取

read_table和read_csv函数中的sep参数是指定文本的分隔符的，如果分隔符指定错误，在读取数据的时候，每一行数据将连成一片。

header参数是用来指定列名的，如果是None则会添加一个默认的列名。

encoding代表文件的编码格式，常用的编码有utf-8、utf-16、gbk、gb2312、gb18030等。如果编码指定错误数据将无法读取，IPython解释器会报解析错误。

读写不同数据源

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。		

读写不同数据源

3. 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, index_col=None, names=None, dtype=None)*

参数名称	说明
io	接收string。表示文件路径。无默认。
sheetname	接收string、int。代表excel表内数据的分表位置。默认为0。
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int、sequence或者False。表示索引列的位置，取值为sequence则代表多重索引。默认为None。
index_col	接收int、sequence或者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。

读写不同数据源

3. 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)*

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。