大连理工大学
未来技术学院/人工智能学院
SCHOOL OF FUTURE TECHNOLOGY, SCHOOL OF ARTIFICIAL INTELLIGENCE, DUT

第11章 Python机器学习常用工具2

AI程序设计课程组

Pandas介绍

- Series数据结构
- DataFrame数据结构

■ Pandas

Pandas 是一个强大的数据分析工具，它是基于 Python 编程语言开发的，专为处理和分析数据而设计。

Pandas 提供了两个主要的数据结构：Series 和 DataFrame。

padas.Series

Series 是一维数据结构，类似于数组或列表，但它具有标签（label），这些标签可以帮助你更容易地访问和操作数据。

Series 由两个主要部分组成：数据和索引。数据是一维的数组，可以包含不同数据类型，如整数、浮点数、字符串等。索引是标签，用于唯一标识数据点。

可以通过标签访问 Series 中的元素，这使得数据检索非常方便。

padas.Series

pandas.Series(data, index, dtype, name, copy)

data: 输入的数据，可以是列表、常量、ndarray 数组等。
index: 索引与data的长度相同，默认为np.arange(n)
dtype: 数据类型
name: 设置名称
copy: 是否复制数据，默认为false

```
import pandas as pd

data = [1, 2, 3, 4, 5]
index = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(data, index=index)
print(series)
```

```
a    1
b    2
c    3
d    4
e    5
```

不指定索引（index）创建Series

使用默认索引，索引默认从 0 开始分配

```
In [13]: import pandas as pd
    ...: a = pd.Series([2, 4, 5, 6])
In [14]: print(a)
0    2
1    4
2    5
3    6
dtype: int64
```

```
In [15]: data = ['xiaoming', 18]
In [16]: x = pd.Series(data)
In [17]: print(x)
0    xiaoming
1          18
dtype: object
```

输出的第一列为index，第二列为数据value。

指定索引（index）创建Series

用户可自定义索引标签，为新建的Series数据指定标识的标签

```
In [20]: data = ['xiaoming', 18, 'python'] # 数据
    ...: index = ['name', 'age', 'class']  #自定义索引标签
    ...: x = pd.Series(data, index)
    ...: print(x)
name     xiaoming
age            18
class      python
dtype: object
```

```
In [18]: s = pd.Series([2, 4, 5, 6], index=list("abcd"))
In [19]: print(s)
a    2
b    4
c    5
d    6
dtype: int64
```

从字典创建Series

通过字典构造，在Series构造函数里传入字典数据，即提供了数据又提供了索引index。

```
import pandas as pd

data = {'a': 0., 'b': 1., 'c': 2.}
s = pd.Series(data)
print(s)
```

```
a    0.0
b    1.0
c    2.0
dtype: float64
```

字典的key作为Series的index，而字典的value作为Series的value值。

从字典创建Series

通过字典构造Series时，如果同时也由额外传入了索引，则需要将索引标签与字典中的值一一对应。如果字典中没有对应的键，使用NaN填充

```
import pandas as pd

data = {'a': 0., 'b': 1., 'c': 2.}
s = pd.Series(data, index=['b', 'c', 'd', 'a'])
print(s)
```

```
b    1.0
c    2.0
d    NaN
a    0.0
dtype: float64
```

创建Series

注意：字典不能有重复的key，key是访问字典value的唯一标识。但是，Series中是允许有重复的index。

```
In [27]: s = pd.Series(np.array([1,2,3]), index = ['a', 'b', 'a'])
In [28]: print(s)
a    1
b    2
a    3
dtype: int32
```

访问Series数据

可以利用 Series数据中的 index 和 values属性获取 具体的 索引与数值。

```
In [37]: s = pd.Series(np.array([1,2,3]), index = ['a', 'b', 'c'])
In [38]: print(s)
a    1
b    2
c    3
dtype: int32
In [39]: print(s.index)
Index(['a', 'b', 'c'], dtype='object')
In [40]: print(s.values)
[1 2 3]
```

访问Series数据

通过索引/标签访问Series数据

Series 类似于字典dict，把 index 中的索引当做 key，而把序列中的元素值当做 value，通过 index 索引来访问或修改元素值

```
import pandas as pd

# 1. 使用索引访问单个元素值
s = pd.Series([6, 7, 8, 9, 10], index=['a', 'b', 'c', 'd', 'e'])
print(s['a'])

# 2. 使用索引访问多个元素值，如果使用了index中不包含的标签，会报异常
print(s[['a', 'c', 'd']])
```

```
6
a    6
c    8
d    9
dtype: int64
```

通过位置访问Series数据

通过位置访问Series数据与 ndarray 和 list 相同，使用元素自身的下标进行访问单个数据或者切片。

```python
import pandas as pd

s = pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e'])
# 1. 位置索引访问
print(s[0])  # pandas的高版本会删除这种访问方式，需要用s.iloc[0]
# 2. 通过切片的方式访问 Series 序列中的数据
print(s[:3])
# 3. 获取后三个元素
print(s[-3:])
```

```
1
a    1
b    2
c    3
dtype: int64
c    3
d    4
e    5
dtype: int64
```

访问Series数据

当Series中有重复的index，用索引 和 位置 访问数据的区别：

```
In [27]: s = pd.Series(np.array([1,2,3]), index = ['a', 'b', 'a'])
In [28]: print(s)
a    1
b    2
a    3
dtype: int32
In [29]: s['a']
Out[29]:
a    1
a    3
dtype: int32
In [30]: s[0]
Out[30]: 1
```

修改Series数据

通过索引/标签修改元素，但如果含有相同索引，会同时被修改

```
import pandas as pd

s = pd.Series([6, 7, 8, 9, 10],
              index=['a', 'b', 'c', 'd', 'e'])
s['a'] = 0
print(s)
```

```
a     0
b     7
c     8
d     9
e    10
dtype: int64
```

```
In [41]: s = pd.Series(np.array([1,2,3]), index = ['a', 'b', 'a'])
In [42]: s['a'] = 0
In [43]: s
Out[43]:
a    0
b    2
a    0
dtype: int32
```

修改Series数据

通过位置修改数据

```
In [44]: s = pd.Series(np.array([1,2,3]), index = ['a', 'b', 'a'])
In [45]: print(s)
a    1
b    2
a    3
dtype: int32
In [46]: s[0] = 5
In [47]: print(s)
a    5
b    2
a    3
dtype: int32
```

添加Series数据

通过索引/标签添加元素

```
import pandas as pd

s = pd.Series([6, 7, 8, 9, 10],
              index=['a', 'b', 'c', 'd', 'e'])
s['f'] = 11
print(s)
```

```
a     6
b     7
c     8
d     9
e    10
f    11
dtype: int64
```

删除Series数据

Series.drop(index, inplace=True)

根据标签 index， 删除Series中的对应元素，inplace=True表示原地操作

```
In [65]: s = pd.Series([1,2, 3, 4, 5], index = ['a', 'b', 'c', 'd', 'e'])
In [66]: s.drop('a')
Out[66]:
b    2
c    3
d    4
e    5
dtype: int64
In [67]: s.drop(['d', 'e'])
Out[67]:
a    1
b    2
c    3
dtype: int64
```

Series对象常用的属性

- axes：以列表的形式返回所有行索引标签
- dtype：返回对象的数据类型。
- empty：返回一个布尔值，判断数据对象是否为空
- ndim：返回输入数据的维数
- size：返回输入数据的元素数量
- values：以 ndarray 的形式返回 Series 对象
- index：返回一个索引的取值

Series对象常用的属性

```python
import pandas as pd

s = pd.Series([6, 7, 8, 9, 10], index=['a', 'b', 'c', 'd', 'e'])

# 1. axes：所有行索引
print(s.axes)

# 2. dtype:返回对象的数据类型
print(s.dtype)

# 3. empty：判断数据对象是否为空
print(s.empty)
```

```
[Index(['a', 'b', 'c', 'd', 'e'], dtype='object')]
int64
False
```

Series对象常用的属性

```python
import pandas as pd

s = pd.Series([6, 7, 8, 9, 10], index=['a', 'b', 'c', 'd', 'e'])
# 4. ndim：查看序列的维度，根据定义，Series 是一维数据结构，因此它始终返回 1。
print(s.ndim)
# 5. size：返回Series对象的大小（长度）
print(s.size)
# 6. values:以数组的形式返回Series对象中的数据
print(s.values)
print(type(s.values))
# 7. index：查看 Series 中索引的取值
print(s.index)
```

```
1
5
[ 6  7  8  9 10]
<class 'numpy.ndarray'>
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

s.head(n)函数

查看前n个数据

```
import pandas as pd
import numpy as np

s = pd.Series(np.random.randn(5))
print(s)
print("============================")
# 返回前三行数据
print(s.head(3))
```

```
0   -0.904897
1   -1.319032
2    0.226215
3    0.044990
4   -0.117182
dtype: float64
============================
0   -0.904897
1   -1.319032
2    0.226215
dtype: float64
```

add 函数: 相加操作

可以将另一个Series对象加到某Series对象里。当两个Series具有相同的index或者label的对应值相加。add函数等价于算术运算符加号。

```
In [83]: x = pd.Series([1,2], index=['a', 'b'])
In [84]: y = pd.Series([3,3], index=['a', 'b'])
In [85]: x+y
Out[85]:
a    4
b    5
dtype: int64
In [86]: x.add(y)
Out[86]:
a    4
b    5
dtype: int64
```

```
In [89]: x = pd.Series([1,2], index=['a', 'b'])
In [90]: y = pd.Series([3], index=['a'])
In [91]: x.add(y)
Out[91]:
a    4.0
b    NaN
dtype: float64
```

_append 函数: 连接操作

_append函数和列表的append函数类似，将另外一个series连接在某series后边。

```
In [94]: x = pd.Series([1,2], index=['a', 'b'])
In [95]: y = pd.Series([3], index=['c'])
In [96]: x._append(y)
Out[96]:
a    1
b    2
c    3
dtype: int64
```

Series常用函数

此外，利用 pandas.concat函数对多个Series数据进行直接合并。

```
In [58]: import pandas as pd
In [59]: s = pd.Series([1,2], index = ['a', 'b'])
In [60]: x = pd.Series([3, 4], index = ['c', 'd'])
In [61]: y = pd.concat([s,x])
In [62]: print(y)
a    1
b    2
c    3
d    4
dtype: int64
```

Series常用函数

count()函数

count函数可以统计series里非NaN数据个数。

```
In [100]: x = pd.Series([1,2], index=['a', 'b'])
In [101]: y = pd.Series([3, 4], index=['a', 'c'])
In [102]: a = x+y
In [103]: x = pd.Series([1,2], index=['a', 'b'])
In [104]: y = pd.Series([3, 4], index=['a', 'c'])
In [105]: s = a+y
In [106]: print(s)
a    7.0
b    NaN
c    NaN
dtype: float64
In [107]: s.count()
Out[107]: 1
```

sort_index() 函数

Series.sort_index(axis=0, level=None, ascending=True, inplace=False, kind='quicksort', na_position='last', sort_remaining=True)

根据index对series数据排序。 ascending控制升序或降序，默认True为升序

```
In [113]: x = pd.Series([1,2,3, 4], index=['b', 'a', 'd', 'c'])
In [114]: x.sort_index()
Out[114]:
a    2
b    1
c    4
d    3
dtype: int64
```

sort_values() 函数

Series.sort_values(axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')

根据value对series数据排序。 ascending控制升序或降序，默认True为升序

```
In [115]: x = pd.Series([2, 3, 1, 4], index=['b', 'a', 'd', 'c'])
In [116]: x.sort_values()
Out[116]:
d    1
b    2
a    3
c    4
dtype: int64
```

Pandas介绍

- Series数据结构
- DataFrame数据结构

pandas.DataFrame

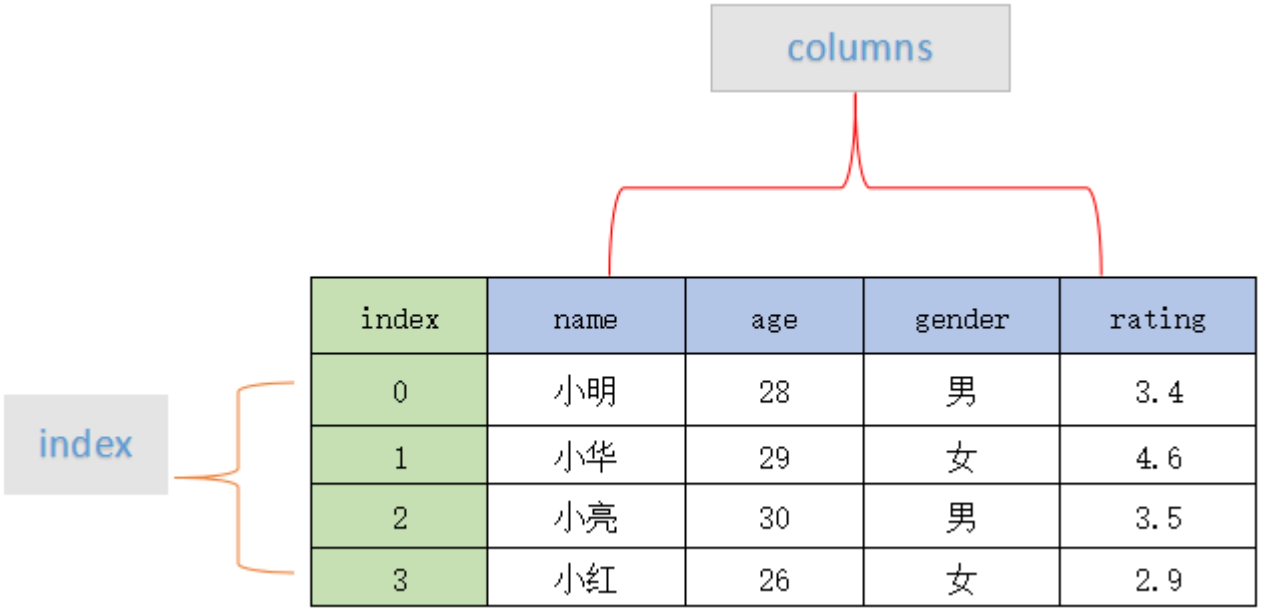
DataFrame 是一个二维的数据结构，类似于电子表格或 SQL 数据库表格。

它由多个 Series 对象构成，每一列都是一个Series结构。每个 Series 可以拥有不同的数据类型。这使得 DataFrame 非常适合处理复杂的数据集，包括结构化数据。

DataFrame 也包括行索引和列标签，使得你可以通过标签或索引轻松访问数据

DataFrame(data, index, columns)

data：数据
index：行标签
columns：列标签



index	name	age	gender	rating
0	小明	28	男	3.4
1	小华	29	女	4.6
2	小亮	30	男	3.5
3	小红	26	女	2.9

pandas.DataFrame

DataFrame(data, index, columns)

data：数据
index：行标签
columns：列标签

```
In [122]: pd.DataFrame([[1,2,3], [3, 4, 5]], index=['a', 'b'], columns=['c1', 'c2', 'c3'])
Out[122]:
   c1  c2  c3
a   1   2   3
b   3   4   5
```

创建DataFrame

通过list创建DataFrame

```
import pandas as pd
import numpy as np

df1 = pd.DataFrame(data=[[1, 2, 3], [11, 12, 13]],
                   index=['r_1', 'r_2'],
                   columns=['A', 'B', 'C'])
print(df1)
print('====================')
df2 = pd.DataFrame(data=[[1], [11]],
                   index=['r_1', 'r_2'],
                   columns=['A'])
print(df2)
print('====================')
df3 = pd.DataFrame(data=np.arange(12).reshape(3, 4),
                   index=list("abc"),
                   columns=list("ABCD"))
print(df3)
```

```
      A   B   C
r_1   1   2   3
r_2  11  12  13
====================
      A
r_1   1
r_2  11
====================
    A  B   C   D
a   0  1   2   3
b   4  5   6   7
c   8  9  10  11
```

创建DataFrame

通过字典创建DataFrame

```python
import pandas as pd
# 传入单个字典
dict1 = {"name": ["jack", "HanMeimei"], "age": ["100", "100"]}
df1 = pd.DataFrame(dict1, index=list("ab"))
print(df1)
print('===============')
# 传入字典列表
dict2 = [{"name": "MaYun1", "age": 100},
         {"name": "MaYun2", "age": 100},
         {"name": "MaYun3", "age1": 100}]
df2 = pd.DataFrame(dict2, index=list("abc"))
print(df2)
```

```
        name  age
a       jack  100
b  HanMeimei  100
===============
     name    age   age1
a  MaYun1  100.0    NaN
b  MaYun2  100.0    NaN
c  MaYun3    NaN  100.0
```

访问DataFrame元素

在Dataframe中选取数据有3种常用方式：

（1）**行（列）选取（单维度选取）：dfdata[]。**

这种情况一次只能选取行或者列，即一次选取中，只能为行或者列设置筛选条件（只能为一个维度设置筛选条件）。

（2）**区域选取（多维选取）： dfdata.loc[]， dfdata.iloc[] 。**

这种方式可以同时为多个维度设置筛选条件。

（3）**单元格选取（点选取）： dfdata.at[]， dfdata.iat[]。**

准确定位一个单元格。

其中，dfdata 表示 DataFrame格式数据

访问DataFrame元素

对 x 按照行选取数据：

```
In [161]: x
Out[161]:
    c1   c2   c3   c4
a    1    2    3    4
b    5    6    7    8
c    9   10   11   12
```

(a) 利用整数位置索引切片（前闭后开）

```
In [162]: x[0:1]  # 取x的第一行
Out[162]:
   c1  c2  c3  c4
a   1   2   3   4
In [163]: x[0:2]  # 取x的前两行
Out[163]:
   c1  c2  c3  c4
a   1   2   3   4
b   5   6   7   8
```

(b) 利用标签索引切片（前闭后闭）

```
In [166]: x[:'a']  # 取 x 的第一行
Out[166]:
   c1  c2  c3  c4
a   1   2   3   4
In [167]: x['a':'b']  #取 x 的前两行
Out[167]:
   c1  c2  c3  c4
a   1   2   3   4
b   5   6   7   8
```

(c) 利用布尔数组

•选取所有‘c1’大于2的行

```
In [170]: x[x['c1']>2]
Out[170]:
    c1   c2   c3   c4
b    5    6    7    8
c    9   10   11   12
```

访问DataFrame元素

对 x 按照列选取数据：

```
In [161]: x
Out[161]:
   c1  c2  c3  c4
a   1   2   3   4
b   5   6   7   8
c   9  10  11  12
```

(a) 利用标签索引（选取单列）

•选取 ‘c1’ 列所有数据

```
In [172]: x['c1']
Out[172]:
a    1
b    5
c    9
Name: c1, dtype: int64
```

(b) 利用标签列表（选取多个列）

•选取 ‘c1’ 、 ‘c3’ 列所有数据

```
In [173]: x[['c1','c3']]
Out[173]:
   c1  c3
a   1   3
b   5   7
c   9  11
```

(c) 取 ‘c1’ 列 ‘a’ 行的元素

```
In [174]: x['c1']['a']
Out[174]: 1
```

访问DataFrame元素

- 区域选取可以从多个维度（行和列）对数据进行筛选，可以通过dfdata.loc[]， dfdata.iloc[] 方法实现。
- 方括号内有两个参数，第一个参数是对行的筛选条件，第二个参数是对列的筛选条件，两个参数用逗号隔开。

dfdata.loc[]：只能使用标签索引，通过标签索引切片时，前闭后闭

dfdata.iloc[]：只能使用整数位置索引，通过整数索引切片时，前闭后开

其中，dfdata 表示 DataFrame格式数据

访问DataFrame元素

对 x 选取区域数据：

```
In [161]: x
Out[161]:
    c1   c2   c3   c4
a    1    2    3    4
b    5    6    7    8
c    9   10   11   12
```

使用 dfdata.loc[]

```
In [184]: x.loc['a':'b', 'c1':'c3']
Out[184]:
    c1   c2   c3
a    1    2    3
b    5    6    7
```

使用 dfdata.iloc[]

```
In [185]: x.iloc[0:2, 0:3]
Out[185]:
    c1   c2   c3
a    1    2    3
b    5    6    7
```

访问DataFrame元素

- 单元格选取包括 dfdata.at[]、dfdata.iat[]两种方法。
- dfdata.at[]和 dfdata.iat[]使用时，需要输入行索引和列索引
- dfdata.at[]只能使用标签索引， dfdata.iat[]只能使用整数索引。

```
In [161]: x
Out[161]:
   c1  c2  c3  c4
a   1   2   3   4
b   5   6   7   8
c   9  10  11  12
```

使用 dfdata.at[]

```
In [188]: x.at['a','c1']
Out[188]: 1
```

使用 dfdata.iat[]

```
In [189]: x.iat[0,0]
Out[189]: 1
```

增加新列


```
In [194]: x = pd.DataFrame([[1,2, 3, 4], [                    :=['a', 'b', 'c'], columns=['c1', 'c2', 'c3', 'c4'])
In [195]: x
Out[195]:
   c1  c2  c3  c4
a   1   2   3   4
b   5   6   7   8
c   9  10  11  12
```

```
In [161]: x
Out[161]:
   c1  c2  c3  c4
a   1   2   3   4
b   5   6   7   8
c   9  10  11  12
```

通过列标签增加列数据

```
In [191]: x['c5'] = [0, 0, 0]
In [192]: x
Out[192]:
   c1  c2  c3  c4  c5
a   1   2   3   4   0
b   5   6   7   8   0
c   9  10  11  12   0
```

通过loc[index]增加行数据

```
In [198]: x.loc['d'] = [0, 0, 0, 0]
In [199]: x
Out[199]:
   c1  c2  c3  c4
a   1   2   3   4
b   5   6   7   8
c   9  10  11  12
d   0   0   0   0
```

删除数据

df.drop(name, axis=0, inplace=True)

删除对应name的行/列，axis=0表示要删除行，axis=1表示要删除列，inplace=True表示原地操作

```
In [161]: x
Out[161]:
   c1  c2  c3  c4
a   1   2   3   4
b   5   6   7   8
c   9  10  11  12
```

```
In [205]: z = x.drop('a', axis=0)
In [206]: z
Out[206]:
   c1  c2  c3  c4
b   5   6   7   8
c   9  10  11  12
d   0   0   0   0
```

```
In [202]: y = x.drop('c1', axis=1)
In [203]: print(y)
   c2  c3  c4
a   2   3   4
b   6   7   8
c  10  11  12
d   0   0   0
```

修改数据


```
In [161]: x
Out[161]:
    c1  c2  c3  c4
a    1   2   3   4
b    5   6   7   8
c    9  10  11  12
```

通过列标签修改列的值

```
In [49]: x['c1'] = [0, 0, 0]
In [50]: x
Out[50]:
    c1  c2  c3  c4
a    0   2   3   4
b    0   6   7   8
c    0  10  11  12
```

通过loc[index]修改行数据

```
In [52]: x.loc['a'] = [0, 0, 0, 0]
In [53]: x
Out[53]:
    c1  c2  c3  c4
a    0   0   0   0
b    5   6   7   8
c    9  10  11  12
```

DataFrame常用属性

DataFrame对象常用的属性

和Series类似

- shape：行数列数
- dtypes：列数据类型
- ndim：数据维度
- index：行索引
- columns：列索引
- values：对象值，二维ndarray数组

DataFrame常用属性

- DataFrame对象常用的属性

```
import pandas as pd

dict1 = {"name": ["jack", "HanMeimei", "Lucy"],
         "age": ["100", "90", "98"],
         "salary": [30000, 50000, 999000]}
df1 = pd.DataFrame(dict1)
print(df1)
print('===============')
print(df1.index)
print('===============')
print(df1.columns)
print('===============')
print(df1.values)
print('===============')
```

```
        name  age  salary
0       jack  100   30000
1  HanMeimei   90   50000
2       Lucy   98  999000
===============
RangeIndex(start=0, stop=3, step=1)
===============
Index(['name', 'age', 'salary'], dtype='object')
===============
[['jack' '100' 30000]
 ['HanMeimei' '90' 50000]
 ['Lucy' '98' 999000]]
===============
```

DataFrame常用函数

DataFrame对象常用的函数

- df.head(n)：显示前n行
- df.tail(n)：显示后n行

```
In [161]: x
Out[161]:
   c1  c2  c3  c4
a   1   2   3   4
b   5   6   7   8
c   9  10  11  12
```

```
In [55]: x.head(2)
Out[55]:
   c1  c2  c3  c4
a   1   2   3   4
b   5   6   7   8
```

```
In [56]: x.tail(2)
Out[56]:
   c1  c2  c3  c4
b   5   6   7   8
c   9  10  11  12
```

DataFrame常用函数

DataFrame对象常用的函数

- df.info()：相关信息概览，行数、列数、列索引等等
- df.describe()：计数、均值、标准差、最大最小值等等

```
In [161]: x
Out[161]:
   c1  c2  c3  c4
a   1   2   3   4
b   5   6   7   8
c   9  10  11  12
```

```
In [57]: x.info()
<class 'pandas.core.frame.DataFrame'>
Index: 3 entries, a to c
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   c1      3 non-null      int64
 1   c2      3 non-null      int64
 2   c3      3 non-null      int64
 3   c4      3 non-null      int64
dtypes: int64(4)
memory usage: 120.0+ bytes
```

```
In [58]: x.describe()
Out[58]:
        c1    c2    c3    c4
count  3.0   3.0   3.0   3.0
mean   5.0   6.0   7.0   8.0
std    4.0   4.0   4.0   4.0
min    1.0   2.0   3.0   4.0
25%    3.0   4.0   5.0   6.0
50%    5.0   6.0   7.0   8.0
75%    7.0   8.0   9.0  10.0
max    9.0  10.0  11.0  12.0
```

利用pd.read_csv()读取csv文件

`pd.read_csv()` 是 Pandas 库中的一个函数，用于读取CSV文件并将其加载到 Pandas 数据结构中，通常是一个DataFrame。

data = pd.read_csv('文件路径', 参数列表)

```
womenDegrees = pd.read_csv('./data/womendegrees.csv')
print(womenDegrees)
```

	Year	Agriculture	Architecture	...
0	1970	4.229798	11.921005	...
1	1971	5.452797	12.003106	...
2	1972	7.420710	13.214594	...
3	1973	9.653602	14.791613	...

数据详情

给定的csv文件中，每一行代表不同出生年份平均每百人中获得各个领域学位的女性占比（百分比）情况。

Year	Agriculture	Architecture	nd Perform	Biology	Business	ations and	nputer Scie	Education	Engineering	English
1990	32.70344	40.82405	62.6	50.81809	47.20085	60.8	29.4	78.86686	14.1	66.9219
1991	34.71184	33.67988	62.1	51.46881	47.22432	60.8	28.7	78.99125	14	66.24147
1992	33.93166	35.20236	61	51.34974	47.2194	59.7	28.2	78.43518	14.5	65.62246
1993	34.94683	35.77716	60.2	51.12484	47.63933	58.7	28.5	77.26731	14.9	65.73095
1994	36.03267	34.43353	59.4	52.24622	47.98392	58.1	28.5	75.81493	15.7	65.64198
1995	36.84481	36.06322	59.2	52.5994	48.57318	58.8	27.5	75.12526	16.2	65.93695
1996	38.96977	35.92649	58.6	53.78988	48.64739	58.7	27.1	75.0352	16.7	66.43778
1997	40.68568	35.10193	58.7	54.99947	48.56105	60	26.8	75.1637	17	66.78636
1998	41.9124	37.59854	59.1	56.35125	49.25852	60	27	75.48616	17.8	67.25545
1999	42.8872	38.63153	59.2	58.22882	49.81021	61.2	28.1	75.83816	18.6	67.82022
2000	45.05777	40.02358	59.2	59.38986	49.80362	61.9	27.7	76.69214	18.4	68.36599
2001	45.86602	40.69028	59.4	60.71233	50.27514	63	27.6	77.37523	19	68.57852
2002	47.13466	41.13295	60.9	61.89513	50.55233	63.7	27	78.64424	18.7	68.82996
2003	47.93519	42.75854	61.1	62.16946	50.3456	64.6	25.1	78.54495	18.8	68.89449
2004	47.88714	43.46649	61.3	61.91459	49.95089	64.2	22.2	78.65075	18.2	68.45473
2005	47.67275	43.10037	61.4	61.50098	49.79185	63.4	20.6	79.06712	17.9	68.57122
2006	46.7903	44.49933	61.6	60.17284	49.21091	63	18.6	78.68631	16.8	68.29759
2007	47.60503	43.10046	61.4	59.41199	49.00046	62.5	17.6	78.72141	16.8	67.87492
2008	47.57083	42.71173	60.7	59.30577	48.88803	62.4	17.8	79.19633	16.5	67.59403
2009	48.66722	43.34892	61	58.48958	48.84047	62.8	18.1	79.53291	16.8	67.96979
2010	48.73004	42.06672	61.3	59.01026	48.75799	62.5	17.6	79.61862	17.2	67.92811
2011	50.03718	42.77344	61.2	58.7424	48.18042	62.2	18.2	79.43281	17.5	68.42673

Matplotib可视化实例

任务一:

以“year”这一列的数据作为横坐标；“Business”这一学位作为纵坐标，绘制折线图。

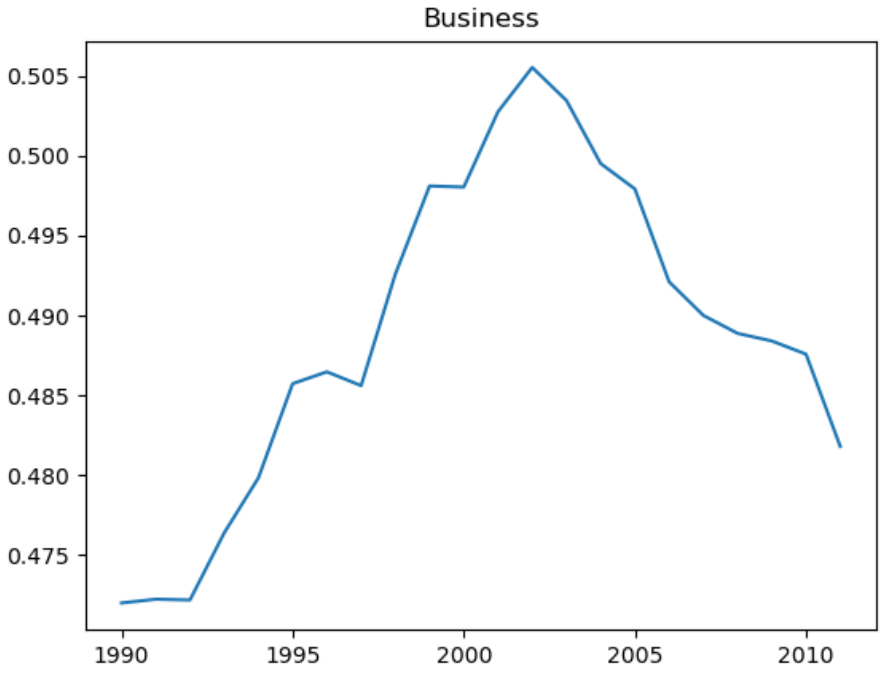
```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('womendegrees.csv')
plt.plot(data['Year'], data['Business']/100.)
plt.title('Business')
plt.show()
```

Matplotib可视化实例

任务一:

以“year”这一列的数据作为横坐标；“Business”这一学位作为纵坐标，绘制折线图。



Matplotib可视化实例

任务二:

以“year”这一列的数据作为横坐标；在一张图上同时绘制男女每百人获得“Business”这一学位的比例。

```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('womendegrees.csv')
plt.plot(data['Year'], data['Business']/100., label='woman')
plt.plot(data['Year'], 1-data['Business']/100., label='men')
plt.title('Business')
plt.show()
```

表中是女性的百分占比，因此男性的百分占比为 1 - 女性百分占比/100

Matplotib可视化实例

任务二:

以“year”这一列的数据作为横坐标；在一张图上同时绘制男女每百人获得“Business”这一学位的比例。