第3章

Pandas 数据处理和分析

本章学习目标

- 掌握 Pandas 的安装、导入和基本的数据结构。
- 掌握 Series 的创建、属性、数据提取、选择和修改方法。
- 掌握 DataFrame 的创建、属性、数据提取、选择和修改方法。
- 掌握 Pandas 对文件读写的一般方法。
- 掌握对缺失值、重复值和异常值的数据清洗方法。
- 掌握索引操作和数据操作的基本方法。
- 掌握合并、分组和变形操作的基本方法。

本章主要内容如图 3.1 所示。

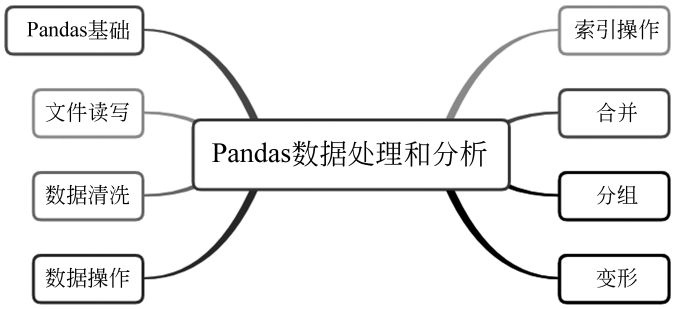

图 3.1　本章主要内容

3.1　Pandas 基础

3.1.1　Pandas 简介

Pandas 是构建于 Python 语言之上的一个快速、强大、灵活且易于使用的开源数据分析和操作工具集。它的使用基础是 NumPy，主要用于数据挖掘和数据分析。

Pandas 最早由 AQR 资金管理公司于 2008 年 4 月开发，并于 2009 年底开源。其初衷是为了便于进行金融数据分析，Pandas 的名称源于 panel data（面板数据，是经济学中

关于多维数据集的术语)和 data analysis(数据分析)。目前,Pandas 的最新版本是 2021 年 4 月 12 日发布的 1.2.4,本书所有函数原型、应用示例及结果都基于该版本。

本书推荐在 Anaconda 集成环境中学习 Pandas。一般情况下,Anaconda 在安装后已经将常用的库安装完毕,如 NumPy、Pandas、Matplotlib 等。若发现 Anaconda 自带的 Pandas 版本过低,可在 Anaconda 提示符后输入 conda install pandas=1.24 命令更新至 1.2.4 版本。

若未使用 Anaconda 编程环境且仅安装了 Python,则需首先检查 Python 版本,确认是否正确安装了 pip,然后下载与 Python 版本相符的 Pandas 安装包并进行安装。

Pandas 在使用之前需使用 import pandas as pd 的方法导入。

Pandas 有 3 种数据结构:Series、DataFrame 和 Panel。

(1) Series:一维数据结构,由一组数据及与之相关的数据标签组成。

(2) DataFrame:二维数据结构,包含一组或多组有序的列,每列的数据类型可以不同。DataFrame 既有行索引也有列索引,可视为 Series 的容器。

(3) Panel:三维数据结构,可视为 DataFrame 的容器。

本书仅介绍 Series 和 DataFrame。

3.1.2 Series

1. 创建 Series

创建 Series 的方法是 pandas.Series(),其参数说明如下:

- data:用于创建 Series 的数据源,可以是 Python 字典、多维数组和标量值。
- index:索引列表。
- dtype:待创建 Series 对象的数据类型。
- name:待创建 Series 对象的名称。

Series 具有隐式索引和显式索引。隐式索引是从 0 开始的整数,也称为位置索引;而显式索引需要通过 index 参数指定,一般称之为标签索引。

Series 可通过多种方法创建,如通过标量、Python 可迭代对象、Python 字典对象、ndarray 对象创建等(ndarray 是 NumPy 的典型数据结构,具体信息可参考 NumPy 文档),还可直接从文件中读取数据创建 Series。

(1) 使用标量创建 Series 的示例如下:

In[1]:	import numpy as np import pandas as pd	#导入 NumPy #导入 Pandas
In[2]:	s=pd.Series(9,index=["a","b","c"]) s	#根据标量 9 创建 Series
Out[2]:	a 9 b 9 c 9 dtype: int64	

(2) 使用可迭代对象创建 Series,如列表或元组等。例如:

In[3]:	s=pd.Series([21,39,42,56])　　#默认索引为 0,1,2,… s
Out[3]:	0　21 1　39 2　42 3　56 dtype: int64
In[4]:	s=pd.Series(range(4)) s
Out[4]:	0　0 1　1 2　2 3　3 dtype: int32

(3) 使用字典创建 Series,其中字典的键就是索引。例如:

In[5]:	s=pd.Series({"a":0,"b":1,"c":2,"d":3}) s
Out[5]:	a　0 b　1 c　2 d　3 dtype: int64

(4) 使用 ndarray 对象创建 Series。例如:

In[6]:	s=pd.Series(np.random.randn(5),index=["a","b","c","d"]) #np.random.randn(5)表示随机生成符合正态分布的 5 个随机数 s
Out[6]:	a　-2.093363 b　 0.735281 c　 0.762980 d　 1.652227 dtype: float64

2. Series 的属性

Series 的常用属性包括 index、array/values、hasnans、empty、dtype、shape、ndim、size、nbytes、name、T 等。

1) index

index返回Series的索引。默认情况下index属性返回显式索引(标签索引),若未指定显式索引则返回隐式索引(位置索引)。对于显式索引,可使用形如s.loc[]的方法访问数据;对于隐式索引,可使用形如s.iloc[]的方法访问数据。例如:

In[7]:	s1=pd.Series(np.random.randint(10,100,4),index=["a","b","c","d"]) #np.random.randint(10,100,4)表示随机生成4个10~100的整数 s1
Out[7]:	a 73 b 74 c 98 d 14 dtype: int32

如果要获取s1的索引,则应使用s1.index:

In[8]:	s1.index
Out[8]:	Index(['a', 'b', 'c', 'd'], dtype='object')

如果要修改s1的索引,则应采用以下形式:

In[9]:	s1.index=list("ABCD") s1
Out[9]:	A 73 B 74 C 98 D 14 dtype: int32

创建Series并查看其索引:

In[10]:	s2=pd.Series(range(5)) s2
Out[10]:	0 0 1 1 2 2 3 3 4 4 dtype: int32
In[11]:	s2.index
Out[11]:	RangeIndex(start=0, stop=5, step=1)

2）array

array和values属性均能返回数组形式的数据，但Pandas推荐使用array。示例如下：

In[12]:	s1.array
Out[12]:	<PandasArray> [73, 74, 98, 14] Length: 4, dtype: int32
In[13]:	s1.values
Out[13]:	array([73, 74, 98, 14])

3）hasnans

hasnans属性用于判断Series中是否有缺失值。例如：

In[14]:	s1.hasnans
Out[14]:	False

4）empty

empty属性用于判断Series是否为空。例如：

In[15]:	s1.empty
Out[15]:	False

dtype、shape、ndim、size、nbytes、name、T等属性与NumPy中的ndarray的属性类似，分别表示Series对象的数据类型、形状、维数、数据元素个数、在内存中占据的字节数、名称、转置等。

3. 基本操作

1）提取和修改数据

从Series中提取和修改数据的操作类似于列表和ndarray。例如：

In[16]:	s1["B"]　　#利用显式索引提取数据，相当于s1.loc["b"]
Out[16]:	74
In[17]:	s1["B"]=3.14　　#利用显式索引修改数据，注意，s1的数据元素均为int32 s1
Out[17]:	A　73 B　 3 C　98 D　14 dtype: int32

In[18]:	s1[1]=26　　#相当于 s1.iloc[1] s1[1]
Out[18]:	26
In[19]:	s1[[0,2]]　　#利用隐式索引提取多个数据
Out[19]:	A　73 C　98 dtype: int32
In[20]:	s1[[0,2]]=(24,58) s1
Out[20]:	A　24 B　26 C　58 D　14 dtype: int32
In[21]:	s1[:3]　　#利用隐式索引切片
Out[21]:	A　24 B　26 C　58 dtype: int32
In[22]:	s1[:2]=(12,64)　　#修改数据 s1
Out[22]:	A　12 B　64 C　58 D　14 dtype: int32

2）添加数据

可直接使用新的索引或 append()添加数据。例如：

In[23]:	s1["F"]=31　　#使用新的索引添加一行数据 s1
Out[23]:	A　12 B　64 C　58 D　14 F　31 dtype: int64
In[24]:	s2

Out[24]:	0 0 1 1 2 2 3 3 4 4 dtype: int32
In[25]:	s3=s1.append(s2[[0,2]]) #使用 append()追加数据,s1 本身不变 s3
Out[25]:	A 12 B 64 C 58 D 14 F 31 0 0 2 2 dtype: int64

3) 删除数据

使用 del()可原地删除索引和相应的数据。drop()默认返回删除索引和元素后的新 Series 对象。drop()使用参数 inplace=True 可原地删除数据,使用参数 labels=[]可一次性删除多个索引及数据。例如:

In[26]:	del s3[0] s3
Out[26]:	A 12 B 64 C 58 D 14 F 31 2 2 dtype: int64
In[27]:	s3.drop(labels=2)
Out[27]:	A 12 B 64 C 58 D 14 F 31 dtype: int64
In[28]:	s3 #s3 并未改变

Out[28]:	A 12 B 64 C 58 D 14 F 31 2 2 dtype: int64
In[29]:	s3.drop(labels=2,inplace=True) s3
Out[29]:	A 12 B 64 C 58 D 14 F 31 dtype: int64

4）删除 Series

使用 del 语句删除 Series。例如：

In[30]:	del s3 s3
Out[30]:	NameError: name 's3' is not defined

3.1.3 DataFrame

1. 创建 DataFrame

创建 DataFrame 的方法是 pandas.DataFrame()，其参数说明如下：

- data：用于创建 DataFrame 的数据源，可以是 Python 字典、多维数组和标量值。
- index：行索引列表。
- columns：列索引列表。
- dtype：待创建的 DataFrame 中数据元素的数据类型。

DataFrame 同样具有隐式索引和显式索引。行方向的显式索引通过 index 参数指定，列方向的显式索引通过 columns 参数指定。

DataFrame 可通过多种方法创建，如通过 Python 字典、Series、二维数组等创建。也可以直接从文件提取数据创建 DataFrame。

（1）通过 Python 字典创建 DataFrame。例如：

In[31]:	import numpy as np import pandas as pd

In[32]:	data={ "张怡然":{"数学":90, "英语":89, "语文":78}, "乔欣":{"数学":82, "英语":95, "语文":86}, "李华轩":{"数学":85, "英语":94,"语文":65}} df=pd.DataFrame(data) df
Out[32]:	张怡然 乔欣 李华轩 数学 90 82 85 英语 89 95 94 语文 78 86 65

(2) 通过 Series 创建 DataFrame。例如：

In[33]:	s1=pd.Series({"数学":90, "英语":89,"语文":78}) s1
Out[33]:	数学 90 英语 89 语文 78 dtype: int64
In[34]:	s2=pd.Series({"数学":82, "英语":95,"语文":86}) s2
Out[34]:	数学 90 英语 89 语文 78 dtype: int64
In[35]:	s3=pd.Series({"数学":85, "英语":94,"语文":65}) s3
Out[35]:	数学 85 英语 94 语文 65 dtype: int64
In[36]:	df=pd.DataFrame({"张怡然":s1,"乔欣":s2,"李华轩":s3}) #通过 Series 创建 DataFrame df
Out[36]:	张怡然 乔欣 李华轩 数学 90 82 85 英语 89 95 94 语文 78 86 65

(3) 通过二维数组创建 DataFrame。例如：

```
In[37]:
df=pd.DataFrame(range(12).reshape(3,4))  #未显式指定行、列索引
df
```

```
Out[37]:
    0   1   2   3
0   0   1   2   3
1   4   5   6   7
2   7   9   10  11
```

```
In[38]:
ind=["上午","下午"]
col=["星期一","星期二","星期三","星期四","星期五"]
df=pd.DataFrame(np.arange(10).reshape(2,5),index=ind,columns=col)
df
```

```
Out[38]:
      星期一  星期二  星期三  星期四  星期五
上午    0      1      2      3      4
下午    5      6      7      8      9
```

```
In[39]:
np.random.seed(100)
ind=["数学","物理","化学"]
col=["张怡然","乔欣","李华轩","云小萌"]
df= pd.DataFrame(np.random.randint(60,100,(3,4)),index=ind,columns=col)  #随机生成12个10~100的整数,组成3行4列的二维数组
df
```

```
Out[39]:
      张怡然  乔欣  李华轩  云小萌
数学    68     84    63     99
物理    83     75    70     90
化学    94     62    94     74
```

2. DataFrame 的属性

DataFrame 主要有 index、columns、dtypes、values、axes、ndim、size、shape、empty 等属性。其中，dtypes、values、ndim、size、shape、empty 等属性的用法和 Series 类似；index 属性表示 DataFrame 的行索引列表，columns 属性表示 DataFrame 的列索引列表，axes 属性表示 DataFrame 行、列两个索引的列表。

```
In[40]:
df
```

```
Out[40]:
      张怡然  乔欣  李华轩  云小萌
数学    68     84    63     99
物理    83     75    70     90
化学    94     62    94     74
```

```
In[41]:
df.dtypes
```

```
Out[41]: 张怡然    int32
         乔欣      int32
         李华轩    int32
         云小萌    int32
         dtype: object
```

```
In[42]: df.values
```

```
Out[42]: array([[68, 84, 63, 99],
                [83, 75, 70, 90],
                [94, 62, 94, 74]])
```

```
In[43]: df.T
```

```
Out[43]:      数学  物理  化学
         张怡然  68    83    94
         乔欣    84    75    62
         李华轩  63    70    94
         云小萌  99    90    74
```

```
In[44]: df.index
```

```
Out[44]: Index(['数学', '物理', '化学'], dtype='object')
```

```
In[45]: df.columns
```

```
Out[45]: Index(['张怡然', '乔欣', '李华轩', '云小萌'], dtype='object')
```

```
In[46]: df.axes
```

```
Out[46]: [Index(['数学', '物理', '化学'], dtype='object'),
          Index(['张怡然', '乔欣', '李华轩', '云小萌'], dtype='object')]
```

3. 基本操作

类似于 Series，DataFrame 在行和列的方向上也存在显式索引（标签索引）和隐式索引（位置索引），提取和修改数据的操作基本都可以通过上述两种索引实现。

1）提取和修改列数据

示例如下：

```
In[47]: df
```

```
Out[47]:      张怡然  乔欣  李华轩  云小萌
         数学  68      84    63      99
         物理  83      75    70      90
         化学  94      62    94      74
```

```
In[48]: df["张怡然"]
        #等价于隐式索引 df.iloc[:,0]和显式索引 df.loc[:,"张怡然"]
```

```
Out[48]: 数学    68
         物理    83
         化学    94
         Name:张怡然, dtype: int32
```

```
In[49]: df.loc[:,"张怡然"]=[99,99,99]  #等价于 df["张怡然"]=[99,99,99]
        df
```

```
Out[49]:      张怡然   乔欣   李华轩   云小萌
         数学    99     84     63      99
         物理    99     75     70      90
         化学    99     62     94      74
```

```
In[50]: df.loc[:,["张怡然","李华轩"]]  #同时提取和修改多列数据的方法同上
```

```
Out[50]:      张怡然   李华轩
         数学    99     63
         物理    99     70
         化学    99     94
```

```
In[51]: df.iloc[:,1:3]=[[60,70],[60,70],[60,70]]  #3 行 2 列数据
        df
```

```
Out[51]:      张怡然   乔欣   李华轩   云小萌
         数学    99     60     70      99
         物理    99     60     70      90
         化学    99     60     70      74
```

2）提取和修改行数据

示例如下：

```
In[52]: import numpy as np
        import pandas as pd
```

```
In[53]: np.random.seed(100)     #在创建随机数之前确定种子值
        ind=["数学","物理","化学"]
        col=["张怡然","乔欣","李华轩","云小萌"]
        df= pd. DataFrame ( np. random. randint ( 60, 100, ( 3, 4)), index = ind,
        columns=col)   #随机生成 12 个 60~100 的整数,组成 3 行 4 列的二维数组。
        df
```

```
Out[53]:      张怡然   乔欣   李华轩   云小萌
         数学    68     84     63      99
         物理    83     75     70      90
         化学    94     62     94      74
```

```
In[54]: df.loc["数学"]
```

```
Out[54]: 张怡然    68
         乔欣      84
         李华轩    63
         云小萌    99
         Name:数学, dtype: int32
```

```
In[55]: df.loc[["数学","化学"]]
```

```
Out[55]:      张怡然  乔欣  李华轩  云小萌
         数学  68      84    63      99
         化学  94      62    94      74
```

```
In[56]: df.loc[["数学","化学"]]=[[100,100,100,100],[60,60,60,60]]
        #2行4列数
        df
```

```
Out[56]:      张怡然  乔欣  李华轩  云小萌
         数学  100     100   100     100
         物理  83      75    70      90
         化学  60      60    60      60
```

```
In[57]: df.iloc[2]
```

```
Out[57]: 张怡然    60
         乔欣      60
         李华轩    60
         云小萌    60
         Name:化学, dtype: int32
```

```
In[58]: df.iloc[:2]
```

```
Out[58]:      张怡然  乔欣  李华轩  云小萌
         数学  100     100   100     100
         物理  99      75    70      90
```

```
In[59]: df.iloc[2]=[99,99,99,99]
        df
```

```
Out[59]:      张怡然  乔欣  李华轩  云小萌
         数学  100     100   100     100
         物理  83      75    70      90
         化学  99      99    99      99
```

```
In[60]: df.head(2)      #提取前两行数据
```

```
Out[60]:      张怡然  乔欣  李华轩  云小萌
         数学  100     100   100     100
         物理  83      75    70      90
```

```
In[61]: df.tail(2)      #提取后两行数据
```

```
Out[61]:      张怡然  乔欣  李华轩  云小萌
         物理  83      75    70      90
         化学  99      99    99      99
```

3）增加行列数据

增加一行数据，可使用如下方法：

```
In[62]:
import numpy as np
import pandas as pd
np.random.seed(100)
ind=["数学","物理","化学"]
col=["张怡然","乔欣","李华轩","云小萌"]
df=pd.DataFrame(np.random.randint(60,100,(3,4)),index=ind,
columns=col)
df
```

Out[62]:

	张怡然	乔欣	李华轩	云小萌
数学	68	84	63	99
物理	83	75	70	90
化学	94	62	94	74

```
In[63]:
df.loc["英语"]=np.random.randint(60,100,4)
#等价于 df.loc["英语",:]=np.random.randint(60,100,4)
df
```

Out[63]:

	张怡然	乔欣	李华轩	云小萌
数学	68	84	63	99
物理	83	75	70	90
化学	94	62	94	74
英语	94	84	75	96

增加一列数据可采用两种方法。一种方法是直接使用列索引和数据。例如：

```
In[64]:
np.random.seed(100)
df["张菲"]=np.random.randint(60,100,4)
#等价于 df[:,"张晓菲"]=np.random.randint(60,100,4)
df
```

Out[64]:

	张怡然	乔欣	李华轩	云小萌	张菲
数学	100	100	100	100	68
物理	83	75	70	90	84
化学	99	99	99	99	63
英语	68	84	63	99	99

另一种方法是利用 insert()在 Dataframe 的指定列中插入数据。其参数说明如下：

- loc：表示第几列。若在第一列插入数据，则 loc=0。
- column：表示待插入的列索引，如 column='newcol'。
- value：表示要插入的数据，数字、array、Series 等均可。
- allow_duplicates：表示是否允许列名重复，True 表示允许与已存在的列名重复。

示例如下：

```
In[65]: df.insert(2,"郝建",np.random.randint(60,100,4))
        df
```

Out[65]:

	张怡然	乔欣	郝建	李华轩	云小萌	张菲
数学	100	100	83	100	100	68
物理	83	75	75	70	90	84
化学	99	99	70	99	99	63
英语	68	84	90	63	99	99

4）删除行列数据

del 命令可用来原地删除列数据。

df.drop()可用来删除行和列。其参数说明如下：

- labels：表示待删除的行或列标签列表。
- axis：为 0 时表示删除行，为 1 时表示删除列。
- inplace：用于指示是否原地删除。

示例如下：

```
In[66]: import numpy as np
        import pandas as pd
```

```
In[67]: np.random.seed(100)
        ind=["数学","物理","化学"]
        col=["张怡然","乔欣","李华轩","云小萌"]
        df= pd. DataFrame (np. random. randint (60, 100, (3, 4)), index = ind,
        columns=col)
        df
```

Out[67]:

	张怡然	乔欣	李华轩	云小萌
数学	68	84	63	99
物理	83	75	70	90
化学	94	62	94	74

```
In[68]: del df["李华轩"]
        df
```

Out[68]:

	张怡然	乔欣	云小萌
数学	68	84	99
物理	83	75	90
化学	94	62	74

```
In[69]: df.drop(labels="物理",axis=0,inplace=False)
```

Out[69]:

	张怡然	乔欣	云小萌
数学	68	84	99
化学	94	62	74

```
In[70]: df          #并未改变
```

```
Out[70]:
        张怡然   乔欣   云小萌
数学    68       84     99
物理    83       75     90
化学    94       62     74

In[71]:
df.drop(labels="云小萌",axis=1,inplace=True)
df

Out[71]:
        张怡然   乔欣
数学    68       84
物理    83       75
化学    94       62

In[72]:
df.drop(df.columns[0],axis=1)
#等价于 df.drop(columns="张怡然",axis=1)

Out[72]:
        乔欣
数学    84
物理    75
化学    62

In[73]:
df.drop(index="数学",axis=0)
#等价于 df.drop(df.index[0],axis=0)

Out[73]:
        张怡然   乔欣
物理    83       75
化学    94       62
```

3.2 文件读写

Pandas 通过一系列读写函数对文件进行读写操作。读文件操作通过 pandas.read_excel()、pandas.read_csv()等函数读取相应的文件内容并返回 Pandas 对象，写文件操作通过 DataFrame.to_excel()、DataFrame.to_csv()等函数将数据写入相应类型的文件中。表 3.1 列举了 Pandas 支持的主要文件读写函数。

表 3.1 Pandas 支持的主要文件读写函数

文件格式类型	数据描述	读文件的函数	写文件的函数
文本文件	CSV	read_csv()	to_csv()
文本文件	JSON	read_json()	to_json()
文本文件	HTML	read_html()	to_html()
文本文件	本地剪贴板	read_clipboard()	to_clipboard()
二进制文件	Excel	read_excel()	to_excel()

续表

文件格式类型	数据描述	读文件的函数	写文件的函数
二进制文件	HDF5	read_hdf()	to_hdf()
二进制文件	Feather	read_feather()	to_feather()
二进制文件	Parquet	read_parquet()	to_parquet()
二进制文件	Msgpack	read_msgpack()	to_msgpack()
二进制文件	Stata	read_stata()	to_stata()
二进制文件	SAS	read_sas()	
二进制文件	Python Pickle	read_pickle()	to_pickle()
SQL文件	SQL	read_sql()	to_sql()
SQL文件	Google Big Query	read_gbq()	to_gbq()

本书介绍最常用的CSV、Excel文件的读写方法，包括pandas.read_csv()、panda.read_excel()、pandas.DataFrame.to_csv()和pandas.DataFrame.to_excel()等。

3.2.1 读写CSV文件

1. 读CSV文件

pandas.read_csv()的常用参数如下：

- filepath_or_buffer：可以是文件路径或URL，也可以是实现read方法的任意对象。
- sep：读取CSV文件时指定的分隔符，默认为逗号。CSV文件的分隔符和读取CSV文件时指定的分隔符必须一致。
- delim_whitespace：默认为False；设置为True时，表示分隔符为空白字符，可以是空格、制表符(\t)等。例如，CSV文件中的分隔符是制表符。
- header：设置导入DataFrame的列名称，默认为"infer"。
- index_col：读取文件时指定某个列为索引。
- usecols：如果列有很多，但不要全部数据，而只要指定的列时，可以使用这个参数。
- skiprows：表示跳过数据开头的行数。
- encoding：指定字符集类型，通常指定为'utf-8'。

直接在pandas.read_csv()的参数中指定源文件，即可读出全部数据。示例代码如下：

```
In[1]:  import pandas as pd
        df=pd.read_csv("bikes.csv")
        df.head()      #head()默认显示前5行数据
```

```
Out[1]:
    Date;Berri;Brebeuf;Sainte;Maisonneuve;Maisonneuve;Parc;…
0    01/01/2012;35;0;38;51;26;10;16;
1    02/01/2012;83;1;68;153;53;6;43;
2    03/01/2012;135;2;104;248;89;3;58;
3    04/01/2012;144;1;116;318;111;8;61;
4    05/01/2012;197;2;124;330;97;13;95;
```

可以看出，上例中 CSV 文件的每一个字段都是用分号分隔的，应该在读取文件时指定分隔符为";"，代码和输出结果如下：

```
In[2]:
df=pd.read_csv("bikes.csv",sep=";")
df.head()
```

```
Out[2]:
    Date         Berri   Brebeuf   Sainte    Maisonneuve  …
0   01/01/2012   35      0         38        51           …
1   02/01/2012   83      1         68        153          …
2   03/01/2012   135     2         104       248          …
3   04/01/2012   144     1         116       318          …
4   05/01/2012   197     2         124       330          …
```

上例的结果中包含了数据分析并不需要的列。若只需要读取 Date、Berri、Sainte 3 列数据，则应使用 usecols 参数指定列名：

```
In[3]:
df=pd.read_csv("bikes.csv",sep=";",usecols=["Date","Berri",
"Sainte"])
df.head()
```

```
Out[3]:
        Date      Berri    Sainte
0    01/01/2012    35       38
1    02/01/2012    83       68
2    03/01/2012    135      104
3    04/01/2012    144      116
4    05/01/2012    197      124
```

若希望指定第一列 Date 作为索引列，则应使用 index_col 参数：

```
In[4]:
df=pd.read_csv("bikes.csv",sep=";",usecols=["Date","Berri",
"Sainte"],index_col="Date")
df.head()
```

```
Out[4]:
              Berri    Sainte
    Date
01/01/2012    35         38
02/01/2012    83         68
03/01/2012    135        104
04/01/2012    144        116
05/01/2012    197        124
```

2. 写入CSV文件

Series和DataFrame都有对应的to_csv()函数，分别为pandas.Series.to_csv()和pandas.DataFrame.to_csv()，部分常用参数如下：

- path_or_buf：字符串形式的文件路径或文件对象。
- sep：输出文件的字段分隔符。默认为逗号。
- columns：可选择某些列写入文件。
- index：布尔值，默认为True，表示写入行(索引)名称。
- encoding：表示在输出文件中使用的编码，通常指定为'utf-8'。

示例代码如下：

```
In[5]: df.to_csv("dataset/newbikes.csv")
```

3.2.2 读写Excel文件

1. 读取Excel文件

pandas.read_excel()的常用参数如下：

- io：字符串型文件类对象，为Excel文件或xlrd工作簿。该字符串也可以是一个URL，例如本地文件可写成file://localhost/path/to/workbook.xlsx。
- sheet_name：指定读取的工作表。默认为0，表示读取第一个工作表的数据。还可以直接指定工作表的名称。
- header：指定某行作为DataFrame的列标签。如果传递的是一个整数列表，则这些行被组合成一个多层索引。
- index_col：指定某列作为DataFrame的行标签。如果传递的是一个列表，则这些列被组合成一个多层索引。
- usecols：None表示解析所有列；如果为整数，则表示要解析的列号。
- skiprows：指定开始时跳过的行数。
- skip_footer：指定从文件尾部开始跳过的行数。

示例如下：

```
In[6]: import pandas as pd
       df=pd.read_excel("data.xlsx")
       df.head()
```

```
Out[6]:
   月份   销售额
0   1    537
1   2    1425
2   3    1819
3   4    3546
4   5    3941
```

若要读取 data.xlsx 中的第 4 个工作表，则应使用 sheet_name 参数。例如：

```
In[7]:  df=pd.read_excel("data.xlsx",sheet_name=3)  #从 0 开始计数
        df.head()
```

```
Out[7]:
    城市    指标
0   北京    94
1   上海    96
2   广州    91
3   深圳    95
4   南京    88
```

若希望将“城市”列的数据作为列索引，则应使用 index_col 参数：

```
In[8]:  df=pd.read_excel("data.xlsx",sheet_name=3,index_col="城市")
        df.head()
```

```
Out[8]:
        指标
城市
北京    94
上海    96
广州    91
深圳    95
南京    88
```

2. 写入 Excel 文件

pandas.DataFrame.to_excel()函数原型较长，具体参数参考相关 API 文档。示例代码如下：

```
In[9]:  df.to_excel("dataset/newbikes.xls")
```

3.3 数据清洗

数据清洗一般包括缺失值、重复值和异常值的处理。

3.3.1 缺失值处理

在 Padans 中，空值和缺失值是不同的。空值是" "；DataFrame 的缺失值一般表示为 NaN 或者 NaT(缺失时间)，Series 的缺失值可表示为 NaN 或 None。若要生成缺失值可用 np.nan 或 pd.Na。涉及缺失值的函数有 4 个：df.dropna()、df.fillna()、df.isnull()和 df.isna()。