

项目三

分析财务大数据——数据清洗与处理

知识目标

- 了解数组与列表的区别
- 掌握 Pandas 数据类型及常用函数
- 理解 DataFrame 与 Series 相互转化原理
- 掌握 Pandas 对不同文件的读写方法
- 理解三种数据筛选方法的不同应用场景
- 掌握 map() 函数、apply() 函数和 applymap() 函数的应用
- 掌握 Pandas 对重复值、缺失值的处理

能力目标

- 学会使用 Pandas 对 Excel 数据进行计算、统计等操作
- 学会借助 Pandas 分析财务报表
- 能使用 Pandas 编写简单的数据清洗函数
- 能使用 dataClean() 函数熟练地对数据进行清洗
- 能综合运用爬虫、数据清洗技术对大数据进行捕捉、处理和分析

素养目标

- 具备一定的创新意识
- 具有学习新技术,不断挑战自我的能力
- 具有认真负责、严谨细致的工作态度和团队协作意识

任务一 认知 Pandas



3-1-1

Pandas 起源
微课视频

1.1

Pandas 的概念

Pandas 译为“熊猫”,但是在 Python 中,Pandas 是一个扩展程序库,可以把它理解为

数据工具箱,用于数据分析。Pandas的优点在于,它纳入了大量数据库和数据模型,提供了高效地操作大型数据集所需的工具。因此,使用它处理数据会更加简单、便捷。经过多年的发展与完善,Pandas目前已被广泛应用于大数据分析的各个领域。



图 3-1-1
NumPy 图标

Pandas 是基于 NumPy 的一种工具,最初是为金融数据分析而开发出来的。而 NumPy 是 Python 的第三方库,是一个很重要、使用很多的科学计算包。它支持大量维度数组与矩阵运算,此外也针对数组运算提供了大量的数学函数库,计算速度非常快,是 Pandas 等与科学计算相关模块的依赖包。

Pandas 与 NumPy 两者之间的区别在于其作用不同。简单来说,Pandas 是专门为处理表格和混杂数据而设计的,而 NumPy 更适合处理统一的数值数组数据。

1.1.1 数组

NumPy 的主要操作对象是多维数组,数组对象名是 ndarray(别名 array),它是一系列同类型数据的集合,集合中元素的索引从 0 开始。数组以中括号[]的形式输出,元素间由空格分开。ndarray 本质是数组,其不同于一般的数组,也可简单理解为数组里面嵌套数组。Numpy 为 ndarray 提供了便利的操作函数,而且性能非常好,因此,其在数值计算、机器学习、人工智能、神经网络等领域都有广泛的应用。

1) 数组的创建

在 NumPy 中,创建数组使用 array()函数,它可以通过常规的 Python 列表或元组创建数组,数组的数据类型根据序列中元素的类型推导得出,数组以嵌套形式来表达多维概念。

【例 3-1-1】 创建数组的代码如下:

```
import numpy as np # 引入 numpy 库,并设置别名为 np
a = np.array([[1,2,3,4],[7,8,9,10]])
print(a)
```

运行结果如下:

```
[[ 1  2  3  4]
 [ 7  8  9 10]]
```

该数组为二维数组,包含两个一维数组,每个一维数组中有 4 个元素。

此外,还可以使用 arange()函数创建数组,NumPy 中的 arange()函数类似于 Python 内置函数 range(),主要用于生成数组,在给定间隔内返回均匀间隔的数组,其语法如下:

```
[numpy.arange(start, stop, step, dtype = None)
```

【例 3-1-2】 创建二维数组。

```
import numpy as np
b = np.arange(12).reshape(3,4) # 创建一个 3 行 4 列的 0~12 之间的数组
print(b)
```

运行结果如下:

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

arange()函数只能生成一维数组,如果想要改变数组的尺寸可以使用 reshape()函数来重新指定尺寸。



请注意

重新指定的数组元素个数必须与一维数组保持一致。

2) 数组的属性

以上述的数组 a 为例,数组的基本属性如表 3-1-1 所示。

表 3-1-1 数组的基本属性

属性	说明	实例	结果
ndim	数组维度数量(轴数)	a.ndim	2
shape	数组的尺寸,整数元组,对于 n 行 m 列的矩阵表示为(n,m)	a.shape	(2, 4)
size	数组元素的总数,相当于 shape 属性中元素的乘积 n * m	a.size	8
dtype	数组中元素类型,可以使用标准 Python 类型创建或指定 dtype,此外,NumPy 提供了原生数据类型,如 int64, float64	a.dtype	dtype('int64')

★随堂练习

数组 a 如下:

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
```

执行语句 a.shape,输出的结果是()。

3) 数组和列表的区别

在 Python 中已经有列表类型,那为什么还需要数组呢?

【例 3-1-3】 分别创建列表 a,b 并计算 $a * 2 + b * 3$ 的结果,常规的计算方法如下:

```
a = [1,2,3,4]
b = [7,8,9,10]
c = []
for i in range(len(a)):
    c.append(a[i] * 2 + b[i] * 3)
print(c)
```

运行结果如下:



3-1-3
Pandas 学习
拓展视频


```
[23, 28, 33, 38]
```

当然,还可以使用高阶函数 `map()` 进行计算,也可以得到同样的结果:

```
c = list(map(lambda x,y: x+2+y*3,a,b))
print(c)
```

接着,再尝试使用数组运算:

```
a = np.array([1,2,3,4])
b = np.array([7,8,9,10])
c = a*2+b*3
print(c)
```

可以发现,不管是使用 `for` 循环还是高阶函数,都需要逐一取出列表中的元素进行运算,而数组则可以直接进行乘法运算,这就体现出了数组的优势:

- 数组对象可以去掉元素间运算所需的循环,使一维向量更像单个单据。
- 使用数组运算可以提升运算速度。
- 数组对象采用相同的数据类型,有助于节省运算和存储空间。

★ 随堂练习

(判断题) `np.array()` 函数可以将列表数据类型(list)转化为 `ndarray` 数组。

()

1.1.2 NumPy 数学与统计函数



图 3-1-1
NumPy 常用数学函数

NumPy 提供了许多数学和统计学相关的函数,主要包含以下几类:纯数学函数、随机数函数、统计函数、梯度函数、线性代数相关函数。在数据分析中使用较多的是随机数函数和统计函数:

- 随机数函数:在 NumPy 下的 `random` 模块中有很多可以产生随机数构成数组的函数,如生成随机整数、0~1 内的浮点数等。
- 统计函数:NumPy 有很多有用的统计函数,可以很方便地从数组中给定的元素中查找最小值、最大值,计算标准差和方差等。

1) NumPy 随机数函数

NumPy 中常用的随机数函数如表 3-1-2 所示。

表 3-1-2 NumPy 中常用的随机数函数

常用的随机数函数	说明
<code>np.random.rand(d0,d1,...,dn)</code>	创建给定形状的随机数数组,返回[0,1)均匀分布的浮点数,如未提供参数返回单个 Python 浮点数

(续表)

常用随机数函数	说明
<code>np.random.randn(d0, d1, ..., dn)</code>	创建给定形状的随机数数组, 标准正态分布, 如未提供参数返回单个 Python 浮点数
<code>np.random.random(size = None)</code>	返回随机浮点数, 范围 $[0, 1)$, <code>size</code> 表示输出数组形状, 可选整数或元组
<code>np.random.randint(low, high = None, size = None, dtype = int)</code>	返回随机整数, 位于半开区间 $[low, high)$
<code>np.random.choice(a, size = None, replace = True, p = None)</code>	从给定的一维数组(<code>a</code>)生成一个随机样本, <code>replace</code> 表示采样是否允许重复(默认为 <code>True</code>), <code>p</code> 表示一维数组中每个元素被采样的概率

【例 3-1-4】 按要求生成随机数。

生成一组范围在 $[0, 1)$ 内的 2 行 2 列的随机浮点数。

生成一组范围在 $[1, 10)$ 内的 2 行 2 列的随机整数。

```
import numpy as np
a = np.random.random((2,2))      # 生成[0,1)随机浮点数
b = np.random.randint(1,10,(2,2)) # 生成[1,10)随机整数
print(a)
print(b)
```

运行结果如下:

```
[[0.2282551  0.10882186]
 [0.69084553 0.12397459]]
[[5 2]
 [8 3]]
```



3-1-5
Numpy 统计函数

2) NumPy 统计函数

在 NumPy 中统计函数非常多, 这里主要介绍几个简单的统计函数, 如 `sum`(求和)、`min`(最小值)、`max`(最大值)、`median`(中位数)、`std`(标准差)、`var`(方差)等。

【例 3-1-5】

```
a = np.array([1,2,3,4])
print(np.sum(a))
print(np.min(a))
print(np.max(a))
```

运行结果如下:

```
10
1
4
```

Pandas 是以 NumPy 为基础,所以,Pandas 中的统计函数也是继承于 NumPy,在后续 Pandas 的学习中,将会接触到更多的 NumPy 统计函数。

★ 课堂练习

数组 a 如下:

```
import numpy as np
```

```
a = np.array([[10,20,30,40],[11,12,13,14]])
```

执行以 np.max(a) 语句,输出的值是()。

1.2 Pandas 数据结构

Pandas 引入规则:

```
import pandas as pd
```

Pandas 包含 Series 和 DataFrame 两种数据类型。

1.2.1 Series

Series 是一种一维数据结构,类似于 Python 列表,能够保存任何数据类型(整数、浮点数、字符串等),相当于 Excel 表中的一列。Series 由一组数据及与之相关的数据索引组成,每个元素带有一个自动索引(索引从 0 开始,也称为原始索引),除自动索引外,还可以自定义索引,自定义索引可以是数字或字符串(自定义索引可重复)。

Series 类型可由 Python 列表、字典、NumPy 数组等数据类型创建,Series 数据类型使用其同名函数创建。

【例 3-1-6】

```
import pandas as pd
```

```
a = pd.Series([11,12,13,14])
```

```
print(a)
```

运行结果如下:

```
0    11
```

```
1    12
```

```
2    13
```

```
3    14
```

```
dtype: int64
```

左边一列从 0 开始的数字是索引,在不指定索引的情况下,默认为原始索引,右边一



3-1-6
认知 Pandas
微课视频一

列是数据项, dtype 是数据项的数据类型。如果想要自定义索引,则可以传入 index 参数:


```
a = pd.Series([11,12,13,14],index=[1,2,3,4])
print(a)
```

运行结果如下:

```
1    11
2    12
3    13
4    14
dtype: int64
```

★随堂练习

执行以下语句,元素值 2 对应的索引值是()。

```
a = pd.Series([1,2,3,4])
print(a)
```

1.2.2 DataFrame

DataFrame 是二维数据结构,相当于 Excel 表格,每列值数据类型可以不同。DataFrame 既有行索引,又有列索引,常用于表达二维数据。

DataFrame 是最常用的 Pandas 数据类型,也是财务数据在 Python 中的最佳存储方式。

1) DataFrame 的创建

DataFrame 由其同名函数创建,语法如下:

```
pd.DataFrame(data,columns=[序列],index=[序列])
```

若不传入 columns 和 index 参数,则默认为自动索引(从 0 开始)。

通常以二维数据创建 DataFrame。例如,可以由以下数据类型创建:

- 二维 ndarray 对象、二维列表、二维元组等。
- 一维 ndarray 对象、列表、字典、元组或 Series 构成的字典。

【例 3-1-7】 通过二维列表创建 Dataframe,如表 3-1-3 所示。

表 3-1-3 二维列表创建 Dataframe

时间	资产总额	负债总额	所有者权益总额
7 月 31 日	18 400 000	7 360 000	11 040 000
8 月 31 日	23 500 000	10 575 000	12 925 000
9 月 30 日	24 320 000	10 214 400	14 105 600



3-1-7
Series 与
DataFrame

```

# 引入 pandas
import pandas as pd

# 创建二维列表
data = [['7月31日', 18400000, 7360000, 11040000],
        ['8月31日', 23500000, 10575000, 12925000],
        ['9月30日', 24320000, 10214400, 14105600]]

# 创建 DataFrame
df = pd.DataFrame(data, columns=['时间', '资产总额', '负债总额',
                                '所有者权益总额'], index=range(1, 4))

df # 在 Jupyter notebook 中显示表格
show_table(df) # 在科云 Python 编辑器中显示表格

```

运行结果如图 3-1-1 所示。

	时间	资产总额	负债总额	所有者权益总额
1	7月31日	18400000	7360000	11040000
2	8月31日	23500000	10575000	12925000
3	9月30日	24320000	10214400	14105600

图 3-1-1 二维列表创建 DataFrame 运行结果

表 3-1-3 还可以通过字典来创建,可以得到同样的结果:

```

# 创建字典
data = {'时间': ['7月31日', '8月31日', '9月30日'],
        '资产总额': [18400000, 23500000, 24320000],
        '负债总额': [7360000, 10575000, 10214400],
        '所有者权益总额': [11040000, 12925000, 14105600]}

# 创建 DataFrame
df = pd.DataFrame(data, index=range(1, 4))

show_table(df)

```

可以看到与列表创建方式不同,使用字典创建 DataFrame,将键作为列索引,但两者可以得到同样的结果。

2) DataFrame 的属性

DataFrame 的常见属性如表 3-1-4 所示。

表 3-1-4 DataFrame 的常见属性

属性	说明	实例	结果
index	DataFrame 的行索引属性	df.index	RangeIndex(start = 1, stop = 4, step = 1)
columns	DataFrame 的列索引属性	df.columns	Index(['时间', '资产总额', '负债总额', '所有者权益总额'], dtype = 'object')
values	DataFrame 的 numpy 原生二维数组数据	df.values	[['7月31日' 18400000 7360000 11040000] ['8月31日' 23500000 10575000 12925000] ['9月30日' 24320000 10214400 14105600]]
dtypes size shape ndim	DataFrame 具有 ndarray 数组的部分属性	df.dtypes	时间 object 资产总额 int64 负债总额 int64 所有者权益总额 int64 dtype: object

★ 随堂练习

执行以下语句,输出的数据类型是什么?

```
import pandas as pd
data = {'项目': ['库存现金', '银行存款', '其他货币资金'],
        '期初借方余额': [8125, 4375, 2750],
        '本期借方发生额': [6875, 3125, 2250],
        '本期贷方发生额': [1000, 2000, 3000]}
df = pd.DataFrame(data, index = range(1, 4))
df.values
```

3) DataFrame 与 Series 的相互转化

DataFrame 单独取一列或一行就是一个 Series,也可以将 Series 转化为 DataFrame。

【例 3-1-8】

取 DataFrame 的一列

```
a = df['资产总额']
```

打印该列并查看数据类型

```
print(a)
```

```
type(a)
```

运行结果如下：

```
1    18400000
2    23500000
3    24320000
Name: 资产总额, dtype: int64

pandas.core.series.Series
```

通过 `type()` 函数可以看到 `a` 的数据类型为 `Series`，接下来再将 `Series` 转化为 `DataFrame`。

【例 3-1-9】

```
df1 = pd.DataFrame(a)
show_table(df1)
```

运行结果如图 3-1-2 所示。

	资产总额
1	18400000
2	23500000
3	24320000

图 3-1-2 变量 `a` 的序列化运行结果

4) DataFrame 的简单运算

`DataFrame` 可以直接取出行列数据进行加减乘除等运算。

【例 3-1-10】 以[例 3-1-7]创建的 `DataFrame` 为例，计算各时点资产负债率。

```
df['资产负债率'] = df['负债总额']/df['资产总额']
show_table(df)
```

运行结果如图 3-1-3 所示。

	时间	资产总额	负债总额	所有者权益总额	资产负债率
1	7/31/1	18400000	7360000	11040000	0.40
2	8/31/1	23500000	10575000	12925000	0.45
3	9/30/1	24320000	10214400	14105600	0.42

图 3-1-3 `DataFrame` 计算资产负债率

`DataFrame` 具有自动对齐功能，计算结果与索引一一对应，计算后 `DataFrame` 可以直接新增列存储计算结果。

★课堂练习

有以下代码：

```
import pandas as pd

data = {'项目': ['库存现金', '银行存款', '其他货币资金'],
        '期初借方余额': [8125, 4375, 2750],
        '本期借方发生额': [6875, 3125, 2250],
        '本期贷方发生额': [1000, 2000, 3000]}

df = pd.DataFrame(data, index = range(1,4))
```

计算各科目期末借方余额，应输入的语句是什么？

5) DataFrame 基本函数

DataFrame 的操作函数有许多，本节简单介绍以下几个基本函数，如表 3-1-5 所示。

表 3-1-5 DataFrame 基本函数

基本函数	描述
rename()	修改行索引/列名
insert()	插入列
drop()	删除行列
head()	返回 DataFrame 前 n 行
tail()	返回 DataFrame 后 n 行

(1) rename()函数：对行索引及列重命名，修改列名传入 columns 参数、修改行索引传入 index 参数。rename()函数语法如下：

```
DataFrame.rename mapper=None, index=None, columns=None, axis=None,
copy=True, inplace=False, level=None, errors='ignore')
```

常用参数如表 3-1-6 所示。

表 3-1-6 rename()函数常用参数

常用参数	描述
mapper	映射器，类似于字典，应用于特定轴标签，与 axis 联合使用
index	更改行索引名称
columns	更改列名
axis	{0 或 'index', 1 或 'columns'}，默认 0，确定 mapper 所针对的轴
inplace	默认为 False，返回新的 DataFrame，True 表示直接在原数据上修改

【例 3-1-11】 修改[例 3-1-7]中 DataFrame 的行列索引。

```
df.rename(columns={'资产总额': '资产', '负债总额': '负债', '所有者权益总额':  
                '所有者权益'}, index={'x': 1, 2: 'y', 3: 'z'})
```

运行结果如图 3-1-4 所示。

	时间	资产	负债	所有者权益
x	7月31日	18400000	7360000	11040000
y	8月31日	23600000	10575000	12925000
z	9月30日	24320000	10214400	14105600

图 3-1-4 DataFrame 的行列索引

知识点拨

rename() 函数适合用于修改个别索引或者列名, 如需修改全部行列索引名, 可直接对 df.columns、df.index 属性重新赋值, 但需要注意的是, 使用该方法必须对所有行索引或列索引进行修改。

(2) insert() 函数: 将列插入 DataFrame 中的指定位置。insert() 函数语法如下:

```
DataFrame.insert(loc, column, value, allow_duplicates=False)
```

常用参数如表 3-1-7 所示。

表 3-1-7 insert() 函数常用参数

常用参数	描述
loc	int, 插入列的位置, 表示第几列, 插入第一列为 loc = 0
column	插入列的列名
value	插入的值, 可选(int, Series, 数组)
allow_duplicates	是否允许列名重复, 默认 False, 如果列名已经存在则报错, 设置为 True 表示允许列名重复

【例 3-1-12】 在[例 3-1-10]中 DataFrame 列索引为 0 的位置插入一列“序号”。

```
df.insert(0, '序号', range(1, 4))  
show_table(df)
```

运行结果如图 3-1-5 所示。

需要在指定位置插入一列时, 可使用 insert() 函数, 若对插入的位置没有要求, 则可直接新增一列并赋值, 新增的一列默认在 DataFrame 最后。

【例 3-1-13】 直接在 DataFrame 后新增一列“序号 2”。

```
df['序号 2'] = range(1, 4)  
show_table(df)
```

	序号	时间	资产总额	负债总额	所有者权益总额	资产负债率
1	1	7月31日	18400000	7360000	11040000	0.40
2	2	8月31日	23500000	10575000	12925000	0.45
3	3	9月30日	24320000	10214400	14105600	0.42

图 3-1-5 列索引为 0 的位置插入一列“序号”

运行结果如图 3-1-6 所示。

	序号	时间	资产总额	负债总额	所有者权益总额	资产负债率	序号2
1	1	7月31日	18400000	7360000	11040000	0.40	1
2	2	8月31日	23500000	10575000	12925000	0.45	2
3	3	9月30日	24320000	10214400	14105600	0.42	3

图 3-1-6 在 DataFrame 后新增一列“序号 2”

(3) drop()函数：删除指定行列。drop()函数语法如下：

```
DataFrame.drop(labels=None, axis=0, index=None, columns=None,
level=None, inplace=False, errors='raise')
```

常用参数如表 3-1-8 所示。

表 3-1-8 drop()函数常用参数

常用参数	描述
labels	单个标签或标签列表
axis	{0 或 'index', 1 或 'columns'}, 是从索引还是列中删除标签, 默认 0, 从行索引中删除
index	单个标签或标签列表: index = labels 等效于 labels, axis = 0
columns	单个标签或标签列表: columns = labels 等效于 labels, axis = 1
inplace	默认为 False, 返回新的 DataFrame, True 表示直接在原数据上删除

【例 3-1-14】 从原数据上删除[例 3-1-13]中增加的“序号 2”。

```
df.drop(columns='序号 2', inplace=True)
show_table(df)
```

运行结果如图 3-1-7 所示。

	序号	时间	资产总额	负债总额	所有者权益总额	资产负债率
1	1	7月31日	18400000	7360000	11040000	0.40
2	2	8月31日	23500000	10575000	12925000	0.45
3	3	9月30日	24320000	10214400	14105600	0.42

图 3-1-7 drop()函数删除“序号 2”列

(4) head()函数、tail()函数语法分别如下:

DataFrame.head(n = 5), 返回 DataFrame 前 n 行, n 默认为 5

DataFrame.tail(n = 5), 返回 DataFrame 后 n 行, n 默认为 5

当 DataFrame 中数据较多时, 可以选择查看表格的前 n 行或后 n 行。

【例 3-1-15】 查看[例 3-1-14]数据的前 2 行和后 2 行。

```
# 查看前 2 行
show_table(df.head(2))
```

运行结果如图 3-1-8 所示。

序号	时间	资产总额	负债总额	所有者权益总额	资产负债率
1	1 7月31日	16400000	7360000	11040000	0.40
2	2 8月31日	23500000	10575000	12925000	0.45

图 3-1-8 使用 head() 函数查看前两行

```
# 查看后 2 行
show_table(df.tail(2))
```

运行结果如图 3-1-9 所示。

序号	时间	资产总额	负债总额	所有者权益总额	资产负债率
2	2 8月31日	23500000	10575000	12925000	0.45
3	3 9月30日	24320000	10214400	14105600	0.42

图 3-1-9 使用 tail() 函数查看后 2 行

6) DataFrame 基本函数运用

【例 3-1-16】 根据二维数组 data 创建 DataFrame, 并依次完成以下任务:

- (1) 使用 insert() 函数在第一列的位置插入列, 列名为“营业收入”, 值为 50 000。
- (2) 使用 rename() 函数在原数据上修改列名, 从左到右列名为营业收入、销售费用、管理费用、财务费用。
- (3) 使用 drop() 函数在原数据上删除最后一行。

```
# 引入规则
import numpy as np
import pandas as pd

# 创建 DataFrame
data = np.arange(1000, 16000, 1000).reshape(5, 3)
```


(续上)

```
df = pd.DataFrame(data)

# 在第一列插入营业收入
df.insert(0, '营业收入', value=50000)

# 依次修改列名
df.rename(columns={0: '销售费用', 1: '管理费用', 2: '财务费用'},
          inplace=True)

# 删除最后一行
df.drop(4, inplace=True)
show_table(df)
```

运行结果如图 3-1-10 所示。

	营业收入	销售费用	管理费用	财务费用
0	50000	1000	2000	3000
1	50000	4000	5000	6000
2	50000	7000	8000	9000
3	50000	10000	11000	12000

图 3-1-10 DataFrame 的基本函数操作

1.3 文件的读取

1.3.1 Excel 文件的读取和写入

在实际工作中,人们经常使用 Excel 进行数据处理和保存。作为功能强大的 Python,它也可以从 Excel 中读取数据,并为其所用。那么,如何将 Excel 中的数据读取出来呢?

1) Excel 文件的读取

Pandas 提供了非常灵活的数据读取方式,可以接受各种来源的数据。

Pandas 读取 Excel 文件使用 `read_excel()` 函数,语法如下:

```
pandas.read_excel(io, sheet_name=0, header=0, names=None,
                  index_col=None, usecols=None, squeeze=False, dtype=None, engine=None,
                  converters=None, true_values=None, false_values=None, skiprows=None,
                  nrows=None, na_values=None, parse_dates=False, date_parser=None,
                  thousands=None, comment=None, skipfooter=0, convert_float=True,
                  **kwargs)
```

`read_excel()` 函数的参数非常多, 这里我们只需要了解一些常用参数即可, 如表 3-1-9 所示。

表 3-1-9 `read_excel()` 函数常用参数

参数名称	备注	说明
<code>io</code>	文件路径, 可接收有效字符串路径	<code>r'D:\Python\data.xlsx'</code> (防止字符转义)
<code>sheet_name</code>	导入的 sheet 页	1. <code>sheet_name = 0</code> : 默认为 0, 即导入第一页, sheet 序号从 0 开始 2. <code>sheet_name = '表名'</code> : 直接输入目标 sheet 的名称, 中英文皆可 3. <code>sheet_name = 'SheetN'</code> : 代表第 N 个 sheet, S 要大写
<code>header</code>	用哪一行作列名	1. <code>header = 0</code> : 默认为 0, 即默认表格第一行作为列名 2. <code>header = [0, 1]</code> : 表示将前两行作为列名 (多重索引)
<code>names</code>	自定义最终的列名	1. <code>names = ['资产总额', '负债总额', '所有者权益总额']</code> 2. 注意: 一般适用于 Excel 缺少列名, 或者需要重新定义列名的情况, <code>names</code> 的长度必须和 Excel 列长度一致, 否则会报错
<code>index_col</code>	用作索引的列	1. <code>index_col = None</code> : 默认数据不带行索引号, pandas 自动分配从 0 开始的索引号 2. <code>index_col = 0</code> : 以第一列作为行索引 3. <code>index_col = [0, 1]</code> : 表示将前两列作为多重索引
<code>usecols</code>	需要读取哪些列	1. <code>usecols = None</code> : 默认取所有列 2. <code>usecols = [0, 2, 3]</code> : 以列号代表要取的列 3. <code>usecols = ['年', '月']</code> : 以列名代表要取的列
<code>converters</code>	强制规定列数据类型	<code>converters = {'时间': str, '资产总额': float}</code> : 将“时间”列数据类型强制规定为字符串, “资产总额”列强制规定为浮点型

【例 3-1-17】 读取以下地址中 Excel 的第一个表格, 并将“年”“月”两列强制转换为字符串。

```
# 引入 pandas
import pandas as pd

# 读取 Excel 文件
df = pd.read_excel('https://cloud-cdn.acctedu.com/webpython/course/d2633ec9ca5c4a288b73f97b4616d541/data.xlsx', converters = {'年': str, '月': str})

# 使用 show_table() 展示 df 前 5 行
show_table(df.head())
```

运行结果如图 3-1-11 所示。

2) Excel 文件的写入

【例 3-1-18】 对[例 3-1-17]读取的表格进行简单运算, 分别计算出“平均资产合

	年	月	流动资产	非流动资产	流动负债	非流动负债	所有者权益
0	2021	1	644977.56	3780673.82	572266.12	2120000	1733385.26
1	2021	2	663209.90	3820905.96	584387.34	2120000	1784728.52
2	2021	3	675872.23	3872786.78	607200.23	2120000	1821458.78
3	2021	4	692674.56	4105445.53	610013.12	2324750	1863356.97
4	2021	5	707906.90	4176813.56	642826.01	2324750	1917144.45

图 3-1-11 读取表格后运行结果

计”与“平均负债合计”。

```
df['平均资产合计'] = df['平均流动资产'] + df['平均非流动资产']
```

```
df['平均负债合计'] = df['平均流动负债'] + df['平均非流动负债']
```

```
df.head()
```

运行结果如图 3-1-12 所示。

	年	月	流动资产	非流动资产	流动负债	非流动负债	所有者权益	资产合计	负债合计
0	2021	1	644977.56	3780673.82	572266.12	2120000	1733385.26	4425651.38	2692266.12
1	2021	2	663209.90	3820905.96	584387.34	2120000	1784728.52	4483115.86	2704387.34
2	2021	3	675872.23	3872786.78	607200.23	2120000	1821458.78	4548659.01	2727200.23
3	2021	4	692674.56	4105445.53	610013.12	2324750	1863356.97	4798120.09	2934763.12
4	2021	5	707906.90	4176813.56	642826.01	2324750	1917144.45	4884720.46	2967576.01

图 3-1-12 读取的表格简单运算

那么,如何将[例 3-1-18]加工过的表格写入一个新的 Excel 呢?

这时就需要使用 to_excel()函数,语法如下:

```
DataFrame.to_excel(excel_writer, sheet_name='Sheet1', na_rep='',
float_format=None, columns=None, header=True, index=True,
index_label=None, startrow=0, startcol=0, engine=None,
merge_cells=True, encoding=None, inf_rep='Inf', verbose=True,
freeze_panes=None)
```

to_excel()函数常用参数如表 3-1-10 所示。

表 3-1-10 to_excel()函数常用参数

常用参数	描述	说明
excel_writer	文件路径	r'D:\Python\data.xlsx'
sheet_name	导出的 Excel 表名	1. sheet_name='Sheet1': 默认表名为 'Sheet1' 2. sheet_name='表名': 自定义表名
index	是否输出行索引	index=True: 默认输出; index=None: 不输出

【例 3-1-19】 将[例 3-1-18]加工过的表格写入一个新的 Excel, 命名为“data2.xlsx”。

```
df.to_excel('data2.xlsx', sheet_name='资产负债表项目')
```

这样就将 DataFrame 写入了一个新的 Excel 中, 文件保存在当前目录中, 由于 Jupyter Notebook 无法直接查看 Excel 文件, 可以将文件下载到本地计算机上查看, 如图 3-1-13 所示。

资产	负债及所有者权益	资产	负债及所有者权益
流动资产	流动资产	流动资产	流动资产
货币资金	货币资金	货币资金	货币资金
应收账款	应收账款	应收账款	应收账款
预付账款	预付账款	预付账款	预付账款
存货	存货	存货	存货
其他流动资产	其他流动资产	其他流动资产	其他流动资产
非流动资产	非流动资产	非流动资产	非流动资产
长期股权投资	长期股权投资	长期股权投资	长期股权投资
固定资产	固定资产	固定资产	固定资产
无形资产	无形资产	无形资产	无形资产
其他非流动资产	其他非流动资产	其他非流动资产	其他非流动资产
流动资产合计	流动资产合计	流动资产合计	流动资产合计
非流动资产合计	非流动资产合计	非流动资产合计	非流动资产合计
资产总计	资产总计	资产总计	资产总计
流动负债	流动负债	流动负债	流动负债
短期借款	短期借款	短期借款	短期借款
应付账款	应付账款	应付账款	应付账款
预收账款	预收账款	预收账款	预收账款
其他流动负债	其他流动负债	其他流动负债	其他流动负债
非流动负债	非流动负债	非流动负债	非流动负债
长期借款	长期借款	长期借款	长期借款
应付债券	应付债券	应付债券	应付债券
其他非流动负债	其他非流动负债	其他非流动负债	其他非流动负债
流动负债合计	流动负债合计	流动负债合计	流动负债合计
非流动负债合计	非流动负债合计	非流动负债合计	非流动负债合计
负债合计	负债合计	负债合计	负债合计
所有者权益	所有者权益	所有者权益	所有者权益
实收资本	实收资本	实收资本	实收资本
资本公积	资本公积	资本公积	资本公积
盈余公积	盈余公积	盈余公积	盈余公积
未分配利润	未分配利润	未分配利润	未分配利润
所有者权益合计	所有者权益合计	所有者权益合计	所有者权益合计
负债及所有者权益总计	负债及所有者权益总计	负债及所有者权益总计	负债及所有者权益总计

图 3-1-13 to_excel()函数输出结果

如果使用 to_excel()函数写入一个已经存在的 Excel, 会将原有的 Excel 表格中的数据全部覆盖。要解决这个问题, 可以使用 ExcelWriter()函数, 该函数语法如下:

```
ExcelWriter(path, engine=None, date_format=None,
datetime_format=None, mode='w', **engine_kwargs)
```

这里通过参数 mode 确定数据写入的模式: mode='w' 表示重写, mode='a' 表示追加。

【例 3-1-20】 在 data2.xlsx 表格中追加新表“资产负债表项目 2”。

```
with pd.ExcelWriter('data2.xlsx', mode='a') as writer:
    df.to_excel(writer, sheet_name='资产负债表项目 2')
```

这里使用 with 进行上下文管理, 通过 ExcelWriter()函数创建一个 ExcelWriter 对象, 并传入文件路径和写入的模式, 将函数返回的结果保存到变量 writer 中, 然后将 df 写入文件, 并指定表名为“资产负债表项目 2”, 追加写入后的 data2.xlsx 中, 如图 3-1-14 所示。

1.3.2 Pandas 读写其他文件

Pandas 支持多种文件读写, 包括文本文件、二进制文件、数据库文件, 具体如表 3-1-11 所示。

图 3-1-14 to_excel() 函数追加新表

表 3-1-11 Pandas 读写各类文件函数

数据源	读 (Reader)	写 (Writer)
CSV	read_csv	to_csv
JSON	read_json	to_json
HTML	read_html	to_html
Local clipboard	read_clipboard	to_clipboard
Fixed-Width Text File	read_fwf	
MS Excel	read_excel	to_excel
OpenDocument	read_excel	
HDF5 Format	read_hdf	to_hdf
Feather Format	read_feather	to_feather
Parquet Format	read_parquet	to_parquet
ORC Format	read_orc	
Msgpack	read_msgpack	to_msgpack
Stata	read_stata	to_stata
SAS	read_sas	
SPSS	read_spss	
Python Pickle Format	read_pickle	to_pickle
SQL	read_sql	to_sql
Google BigQuery	read_gbq	to_gbq

从科云大数据中心爬取的文件格式除了 Excel 文件,还包括 csv 文件和 json 文件,这两种文件的读写与 Excel 文件的读写类似。

1) csv 文件读写

【例 3-1-21】 爬取科云大数据中心“江苏立华牧业股份有限公司(股票代码:300761)”的资产负债表,并读取 csv 文件。

```
# 引入库
import pandas as pd

# 读取 csv 文件
df_csv = pd.read_csv('https://keyun-oss.acctedu.com/app/bigdata/2021/company/year/zcfzb_300761.csv')

show_table(df_csv.head())
```

运行结果如图 3-1-15 所示。

	报告日期	2021-1-31	2020-1-31	2019-1-31	2018-1-31	2017-1-31	2016-1-31	2015-1-31	2014-1-31	2013-1-31	2012-1-31	Username
0	货币资金(元)	24282	31807	19360	25452	37630	10156	10668	9277	20136	14494	NaN
1	应收账款(元)	—	—	—	—	—	—	—	—	—	—	NaN
2	预付款项(元)	—	—	—	—	—	—	—	—	—	—	NaN
3	存货(元)	71334	151195	290026	—	—	—	—	—	—	—	NaN
4	流动资产合计	—	—	—	—	—	—	—	—	—	—	NaN

图 3-1-15 读取 csv 文件运行结果

【例 3-1-22】 将[例 3-1-21]爬取的资产负债表写入 csv 文件。

```
df_csv.to_csv('300761.csv')
```

to_csv()函数存在 mode 参数,默认为“w”即重新写入,设置为“a”表示追加写入。

2) json 文件读写

【例 3-1-23】 爬取科云大数据中心“采矿业”行业信息,并读取 json 文件。

```
# 引入库
import pandas as pd

# 读取 json 文件
df_json = pd.read_json('https://keyun-oss.acctedu.com/app/bigdata/2021/block_hy02000.json')

show_table(df_json.head())
```


运行结果如图 3-1-16 所示。

Index	code	name	fullname
0	'600968'	'海陆发展'	'海陆发展股份有限公司'
1	'603727'	'博万村'	'博万村股份有限公司'
2	'600311'	'ST 圣华'	'ST 圣华股份有限公司'
3	'601808'	'中海油田服务'	'中海油田服务股份有限公司'
4	'600028'	'中国石化上海石油化工股份有限公司'	'中国石化上海石油化工股份有限公司'

图 3-1-16 json 文件读取示例

【例 3-1-24】 将[例 3-1-23]爬取的采矿业行业信息写入 json 文件。

```
df_json.to_json('hy002000.json', force_ascii=False)
```

知识点拨

写入 json 文件时,默认强制使用 ASCII 编码,ASCII 中不包含中文,所以,只有将 force_ascii 参数设置为 False 才能正常显示中文。

★随堂练习

以下代码实现的功能是什么?

```
with pd.ExcelWriter('data2.xlsx', mode='a') as writer:
    df.to_excel(writer, sheet_name='资产负债表项目 2')
```

3) set_option() 函数

在[例 3-1-21]中,读取到的资产负债表一共包含 108 行,但是 Pandas 默认输出的行列都有省略,当行数较多时,默认只显示前后 5 行数据,如图 3-1-17 所示。

	报告日期	2021-12-31	2020-12-31	2019-12-31	2018-12-31	2017-12-31	2016-12-31	2015-12-31	2014-12-31	2013-12-31	2012-12-31	Unnamed: 11
0	货币资金(万元)	24282	31807	19360	25452	37630	10156	10668	9277	20136	11404	NaN
1	结算备付金(万元)	—	—	—	—	—	—	—	—	—	—	NaN
2	拆出资金(万元)	—	—	—	—	—	—	—	—	—	—	NaN
3	交易性金融资产(万元)	71334	151105	288626	—	—	—	—	—	—	—	NaN
4	衍生金融资产(万元)	—	—	—	—	—	—	—	—	—	—	NaN
...
103	货币资金(万元)	—	—	—	—	—	—	—	—	—	—	NaN
104	归属于上市公司股东权益合计(万元)	625904	667060	682447	414814	284844	209417	166261	105557	62028	38030	NaN
105	少数股东权益(万元)	—	—	—	—	—	—	—	—	37	36	NaN
106	所有者权益(或股东权益)合计(万元)	625904	667060	682447	414814	284844	209417	166261	105557	61991	38066	NaN
107	负债和所有者权益(或股东权益)总计(万元)	1089422	925052	850570	569704	413267	342833	272585	222114	201374	154874	NA

108 rows x 12 columns

图 3-1-17 Pandas 默认输出行列有省略

那么,如何才能显示 DataFrame 的全部行列呢?

set_option()函数是用来设置 Pandas 中一些指定参数的值,如显示的最大行数、列数等等,其语法如下:

```
pandas.set_option(pat, value)
```

pat 参数数据类型为 str,表示需要设置的项目。value 参数表示新设置的值,Pandas 默认设置有许多,表 3-1-12 列出了几个常用设置项。

表 3-1-12 set_option()函数常用设置

变量设置	说明
display.max_rows	设置 DataFrame 显示最大行数 pd.set_option('display.max_rows',None)显示所有行
display.max_columns	设置 DataFrame 显示最大列数
display.max_colwidth	设置 DataFrame 显示最大列宽
display.precision	设置显示小数点后的位数
display.float_format	设置浮点数的显示格式

★课堂练习

df 为 DataFrame 数据类型,共 100 行,现在要查看最后 10 行,应填入的代码是什么?

1.4 财务案例实践

(1) 中淮公司固定资产折旧表情况如表 3-1-13 所示。

表 3-1-13 中淮公司固定资产折旧表

资产名称	资产原值	残值率	残值额	期限(月)	累计折旧	本月折旧
华为电脑	50 000	0.05	250	60	395.85	79.17
打印机	3 400	0.05	170	60	269.15	53.83
空调	2 200	0.05	110	60	174.15	34.83
小轿车	400 000	0.05	20 000	120	6 333.34	3 166.67

要求:先根据以上表格资料创建 DataFrame,再根据创建的 DataFrame 添加“期末净值”列。(注:期末净值=资产原值-累计折旧)

```
#引入 pandas
import pandas as pd
def proc():

    ###修改代码开始###
    #创建 data 数据
```

(续上)

```

data = [['华为电脑', 50000, 0.05, 250, 60, 395.65, 79.17],
        ['打印机', 3400, 0.05, 170, 60, 269.15, 53.83],
        ['空调', 2200, 0.05, 110, 60, 174.15, 34.83],
        ['小轿车', 400000, 0.05, 20000, 120, 6333.34, 3166.67]]

# 设置列索引属性,“期限(月)”为中文状态下的括号
Columns =

# 设置行索引属性,从 1 开始的数字
Index =

# 创建 DataFrame
df = pd.DataFrame( # 在此填充 DataFrame 属性)

# 计算“期末净值”

return df

### 修改代码结束 ###
df = proc()
print(df)

```

(2) 中准公司 2021 年 12 月工资明细表 Excel 数据如下:

URL 地址: <https://cloud-cdn.acctedu.com/webpython/course/d2633ec9ca5ca288b73f97b4616d541/gzmx.xls>

要求: 读取 Excel 中的“工资结算明细表”, 并添加列“单位支付合计数”计算单位支付合计数。(注: 单位支付合计数 = 应付工资 + 单位缴纳社保 + 单位缴纳住房公积金)

```

# 引入 pandas
import pandas as pd

def proc():

    ### 修改代码开始 ###
    # 读取 Excel 文件

    # 计算单位支付合计数

return df

```


(续上)

```
###修改代码结束###
```

```
df = proc()
```

```
print(df)
```

参考答案:

(1) 创建的 DataFrame 添加“期末净值”列的代码。

```
#创建 data 数据
```

```
data = [['华为电脑', 50000, 0.05, 250, 60, 395.85, 79.17],
```

```
['打印机', 3400, 0.05, 170, 60, 269.15, 53.83],
```

```
['空调', 2200, 0.05, 110, 60, 174.15, 34.83],
```

```
['小轿车', 400000, 0.05, 20000, 120, 6333.34, 3166.67]]
```

```
#设置列索引属性,“期限(月)”为中文状态下的括号
```

```
Columns = ['资产名称', '资产原值', '残值率', '残值额', '期限(月)', '累计折旧', '本月  
折旧']
```

```
#设置行索引属性,从 1 开始的数字
```

```
Index = range(1,5)
```

```
#创建 DataFrame
```

```
df = pd.DataFrame(data, columns = Columns, index = Index)
```

```
#计算“期末净值”
```

```
df['期末净值'] = df['资产原值'] - df['累计折旧']
```

```
return df
```

(2) 读取并添加列“单位支付合计数”计算单位支付合计数的代码。

```
#读取 Excel 文件
```

```
df = pd.read_excel(r'https://keyun-oss.acctedu.com/app/bigdata/basics/gzmx.xls', sheet_name = '工资结算明细表')
```

```
#计算单位支付合计数
```

```
df['单位支付合计数'] = df['应付工资'] + df['单位缴纳社保'] + df['单位缴纳住房公积金']
```

```
return df
```

任务二 数据筛选与查询

从其他平台爬取的数据往往包含各种各样的信息,而数据分析需要的可能仅仅是其中的一部分。那么,如何才能获取精简、合适的数据呢?