Python 程序设计教案

<table>
<tr><td>本次授课内容</td><td colspan="2">14.1 pandas 基本操作
14.2 pandas 结合 matplotlib 进行数据可视化</td></tr>
<tr><td>本次课的
教学目的</td><td colspan="2">掌握 pandas 的基本操作
掌握缺失值处理方法
掌握重复值处理方法
掌握异常值处理方法
了解如何使用 pandas 结合 matplotlib 进行数据可视化</td></tr>
<tr><td>本次课教学
重点与难点</td><td colspan="2">缺失值、重复值、异常值的处理</td></tr>
<tr><td>教学方法
教学手段</td><td colspan="2">PPT、边讲边练</td></tr>
<tr><td rowspan="3">课堂教学
时间分配</td><td>教学内容</td><td>时间分配（分）</td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td>课堂教学设计</td><td colspan="2">首先介绍 pandas 基本操作和数据处理与分析基本思路，然后重点介绍 DataFrame 结构的操作以及缺失值、重复值、异常值的处理，以及分组计算、数据差分和数据可视化的基础内容。</td></tr>
<tr><td>实　验</td><td colspan="2">教材中所有例题。</td></tr>
<tr><td>思考题及作业题</td><td colspan="2">本章所有课后习题。</td></tr>
<tr><td>备　注</td><td colspan="2"></td></tr>
<tr><td>教学后记</td><td colspan="2"></td></tr>
<tr><td></td><td colspan="2">第　一　节　课</td></tr>
</table>

课堂重点内容详解	可以根据 numpy 的二维数组生成 pandas 的二维数组。以下操作生成的 12 行 4 列的二维数组 DataFrame，索引为上一小节生成的 dates（间隔为月），列名分别为 A、B、C、D。
	
	`>>> pd.DataFrame(np.random.randn(12,4),  #数据`
	`index=dates,  #索引`
	`columns=list('ABCD'))  #列名`
	
	可以根据 Python 字典生成 pandas 的二维数组。以下生成的 4 行 6 列的二维数组中，A 列是取值 1 到 100 的随机正数；B 列为通过 date_range（）函数生成的时间序列，C 列为 pandas 的 Series 一维数组并指定了索引，D 列为 numpy 一维数组，E 列为 pandas 的 Categorical 类型的一维数组，F 列为 4 个字符串。
	`>>> df = pd.DataFrame({'A':np.random.randint(1, 100, 4),`
	`'B':pd.date_range(start='20180301', periods=4, freq='D'),`
	`'C':pd.Series([1, 2, 3, 4],`
	`index=['zhang', 'li', 'zhou', 'wang'],`
	`dtype='float32'),`
	`'D':np.array([3] * 4,dtype='int32'),`
	`'E':pd.Categorical(["test","train","test","train"]),`
	`'F':'foo'})`
	`>>> df`
	查看二维数组数据的统计信息
	
	`>>> df.describe()  #平均值、标准差、最小值、最大值等信息`
	对二维数组进行排序操作
	
	`>>> df.sort_index(axis=0, ascending=False) #对索引进行降序排序`
	`>>> df.sort_index(axis=0, ascending=True)  #对索引升序排序`
	`>>> df.sort_index(axis=1, ascending=False) #对列进行降序排序`
	`>>> df.sort_values(by='A')  #按 A 列对数据进行升序排序`
	`>>> df.sort_values(by=['E', 'C'])  #先按 E 列升序排序`
	`#如果 E 列相同，再按 C 列升序排序`
	二维数组数据的选择与访问
	
	`>>> df['A']  #选择某一列数据`
	`>>> 60 in df['A']  #df['A']是一个类似于字典的结构`
	`#索引类似于字典的键`
	`#默认是访问字典的键，而不是值`
	`>>> 60 in df['A'].values  #测试 60 这个数值是否在 A 列的值中`
	`>>> df[0:2]  #使用切片选择多行`
	`>>> df.loc[:, ['A', 'C']]  #选择多列`
	`>>> df.loc[['zhang', 'zhou'], ['A', 'D', 'E']]#同时指定多行和多列`
	`>>> df.loc['zhang', ['A', 'D', 'E']]  #查看'zhang'的三列数据`
	`>>> df.at['zhang', 'A']  #查询指定行、列位置的数据值`
	`>>> df.at['zhang', 'D']`
	`>>> df.iloc[3]  #查询二维数组第 3 行数据`
	`>>> df.iloc[0:3, 0:4]  #查询二维数组前 3 行、前 4 列数据`
	`>>> df.iloc[[0, 2, 3], [0, 4]]  #查询二维数组指定的多行、多列数据`
	`>>> df.iloc[0,1]  #查询二维数组第 0 行第 1 列位置的数据值`

```
>>> df.iloc[2,2]                        #查询二维数组第 2 行第 2 列位置的数据值
>>> df[df.A>50]                         #查询 A 列大于 50 的所有行
>>> df[df['E']=='test']                 #查询 E 列为'test'的所有行
>>> df[df['A'].isin([20,69])]           #查询 A 列值为 20 或 69 的所有行
>>> df.nlargest(3, ['C'])              #返回 C 列值最大的前 3 行
>>> df.nlargest(3, ['A'])              #返回 A 列值最大的前 3 行
```

二维数组的数据修改

```
>>> df.iat[0, 2] = 3                    #修改指定行、列位置的数据值
>>> df.loc[:, 'D'] = np.random.randint(50, 60, 4)
                                       #修改某列的值
>>> df['C'] = -df['C']                  #对指定列数据取反
>>> df                                    #查看上面三个修改操作的最终结果
>>> dff = df[:]                          #切片
>>> dff['C'] = dff['C'] ** 2            #替换列数据
>>> dff
>>> dff = df[:]
>>> dff.loc[dff['C']==9.0, 'D'] = 100
                                #把 C 列值为 9 的数据行中的 D 列改为 100
>>> dff
>>> data = pd.DataFrame({'k1':['one'] * 3 + ['two'] * 4,
                         'k2':[1, 1, 2, 3, 3, 4, 4]})
>>> data.replace(1, 5)                    #把所有 1 替换为 5
>>> data.replace({1:5, 'one':'ONE'})  #使用字典指定替换关系
```

第 二 节 课

二维数组缺失值的处理。

```
>>> df1 = df.reindex(columns=list(df.columns) + ['G'])
                                        #增加一列，列名为G
>>> df1.iat[0, 6] = 3    #修改指定位置元素值，该列其他元素仍为缺失值
>>> df1.dropna()                          #返回不包含缺失值的行
>>> df1['G'].fillna(5, inplace=True)  #使用指定值原地填充缺失值
```

二维数组重复值的处理。首先生成一个包含重复值的、7 行 2 列的二维数组 data，然后操作围绕其进行。

```
>>> data = pd.DataFrame({'k1':['one'] * 3 + ['two'] * 4,
                         'k2':[1, 1, 2, 3, 3, 4, 4]})
>>> data.drop_duplicates()        #返回新数组，删除重复行
>>> data.drop_duplicates(['k1'])  #删除 k1 列的重复数据
>>> data.drop_duplicates(['k1'], keep='last')
                                  #对于重复的数据，只保留最后一个
```

二维数组异常值的处理。首先生成一个 500 行 4 列的二维数组（这里生成的二维数组在此并未列出），以下操作围绕该二位数组进行。所谓异常值，一般指超出了正常范围的数据。对于异常值，常见的处理方式是使用正常范围的极限值进行替换，将其拉低或拉高。

```
>>> import numpy as np
>>> import pandas as pd
>>> data = pd.DataFrame(np.random.randn(500, 4))
>>> data.describe()                #查看数据的统计信息
>>> col2 = data[2]             #获取第 2 列的数据
>>> col2[col2>3.5]             #查询该列中大于 3.5 的数值
                               #12 表示行号，3.986027 是该行的数据值
>>> col2[col2>3.0]             #查看该列中大于 3.0 的数值
>>> col2[col2>2.5]             #查看该列中大于 2.5 的数值
                               #第一列为行号
>>> data[np.abs(data)>2.5] = np.sign(data) * 2.5
                          #把所有数据都限定到[-2.5, 2.5]之间
>>> data.describe()
```

二维数组的映射

映射是指使用数据处理的结果替换原始数据值，可以实现对原始数据的修改，其中 map() 方法的工作原理类似于 Python 内置函数 map()，但支持更多用法。既可以把一个函数或 lambda 表达式作用到一个序列上，还支持使用字典来指定映射关系。

```
>>> data = pd.DataFrame({'k1':['one'] * 3 + ['two'] * 4,
                         'k2':[1, 1, 2, 3, 3, 4, 4]})
>>> data['k1'] = data['k1'].map(str.upper)  #使用函数进行映射
>>> data['k1'] = data['k1'].map({'ONE':'one', 'TWO':'two'})
                                  #使用字典表示映射关系
```

二维数组数据离散化

数据离散化一般用来把采集到的数据点分散到设定好的多个区间中，然后可以统计不同区间内数据点的频次，或者也可以在不同的区间内选择特定数据值代表该区间的数据，实现

降维的效果。

```
>>> from random import randrange
>>> data = [randrange(100) for _ in range(10)]  #生成随机数
>>> category = [0, 30, 70, 100]                 #指定数据切分的区间边界
>>> pd.cut(data, category)
>>> pd.cut(data, category, right=False)          #左闭右开区间
>>> labels = ['low', 'middle', 'high']
>>> pd.cut(data, category, right=False, labels=labels) #指定标签
>>> pd.cut(data, 4)                             #四分数
```

移位与频次统计

移位用来把数据整体进行移动，而频次统计则很容易实现指定列的数据分布统计。

```
>>> df1.shift(1)                          #数据下移一行
>>> df1.shift(-1)                         #数据上移一行
>>> df1['D'].value_counts()               #直方图统计
>>> df1['G'].value_counts()               #统计 G 列数据分布情况
```

拆分与合并/连接

通过切片操作可以实现数据拆分，可用来计算特定范围内数据的分布情况，连接则是相反的操作，可以把多个 DataFrame 对象合并为一个 DataFrame 对象。

```
>>> df2 = pd.DataFrame(np.random.randn(10, 4))
>>> p1 = df2[:3]                     #拆分，得到前 3 行数据
>>> p2 = df2[3:7]                    #获取第 3 到 6 行数据
                                       #注意，切片表示左闭右开区间
>>> p3 = df2[7:]                     #获取下标为 7 之后所有行的数据
>>> df3 = pd.concat([p1, p2, p3])  #数据行合并
```

分组计算

在进行数据处理和分析时，经常需要按照某一列对原始数据进行分组，而该列数值相同的行中其他列则进行求和、求平均等操作，这可以通过 groupby()方法和 sum()、mean()等方法来实现。

```
>>> df4 = pd.DataFrame({'A':np.random.randint(1,5,8),
                        'B':np.random.randint(10,15,8),
                        'C':np.random.randint(20,30,8),
                        'D':np.random.randint(80,100,8)})
>>> df4.groupby('A').sum()          #数据分组计算
>>> df4.groupby(['A','B']).mean()
        C     D
>>> df4.groupby(['A','B'], as_index=False).mean()
                        #加 as_index=False 参数可防止分组名变为索引
```

数据差分

```
>>> df = pd.DataFrame({'a':np.random.randint(1, 100, 10),
                       'b':np.random.randint(1, 100, 10)},
```

```
                          index=map(str, range(10)))
>>> df
>>> df.diff()             #纵向一阶差分，每行数据变为该行与上一行数据的差
>>> df.diff(axis=1)       #横向一阶差分
>>> df.diff(periods=2)      #纵向二阶差分
```

读写文件

在处理实际数据，经常需要从不同类型的文件中读取数据，或者自己编写网络爬虫从网络上读取数据。从文本文件、Excel 或 Word 文件中读取数据的相关知识请参考本书第 9 章，网络爬虫有关的知识请参考本书第 13 章。这里简单介绍使用 pandas 直接从 Excel 和 CSV 文件中读取数据以及把 DataFrame 对象中的数据保存至 Excel 和 CSV 文件中的方法。

```
>>> df.to_excel('d:\\test.xlsx', sheet_name='dfg')
                                      #将数据保存为 Excel 文件
>>> df = pd.read_excel('d:\\test.xlsx', 'dfg',
                       index_col=None, na_values=['NA'])
>>> df.to_csv('d:\\test.csv')          #将数据保存为 csv 文件
>>> df = pd.read_csv('d:\\test.csv')  #读取 csv 文件中的数据
>>> import pandas as pd
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> df = pd.DataFrame(np.random.randn(1000, 2),     #1000 行 2 列随机数
                      columns=['B', 'C']).cumsum()  #创建 DataFrame
>>> df['A'] = pd.Series(list(range(len(df))))       #创建索引
>>> plt.figure()
>>> df.plot(x='A')                                     #绘制折线图
>>> plt.show()                                         #显示图形
>>> df = pd.DataFrame(np.random.rand(10, 4),   #生成 4 列随机数
                      columns=['a', 'b', 'c', 'd'])
>>> df.plot(kind='bar')                           #绘制垂直柱状图
>>> plt.show()
>>> df = pd.DataFrame(np.random.rand(10, 4), #生成 4 列随机数
                      columns=['a', 'b', 'c', 'd'])
>>> df.plot(kind='barh', stacked=True)        #绘制水平柱状图
>>> plt.show()
```

Python 程序设计教案

本次授课内容	14.3 pandas 应用案例	
本次课的 教学目的	了解 pandas 在数据分析中的应用	
本次课教学 重点与难点	pandas 在数据分析中的应用	
教学方法 教学手段	PPT、边讲边练	
课堂教学 时间分配	**教学内容**	**时间分配（分）**
课堂教学设计	通过几个案例演示 pandas 在数据分析中的应用。	
实　验	14.3 节所有例题。	
思考题及作业题	本章所有课后习题。	
备　注		
教学后记		

	第　一　节　课
	例 14-1 模拟转盘抽奖游戏，统计不同奖项的获奖概率。

课堂重点内容详解

```
import numpy as np
import pandas as pd

#模拟转盘100000次
data = np.random.ranf(100000)
#奖项等级划分
category = (0.0, 0.08, 0.3, 1.0)
labels = ('一等奖', '二等奖', '三等奖')
#对模拟数据进行划分
result = pd.cut(data, category, labels=labels)
#统计每个奖项的获奖次数
result = pd.value_counts(result)
#查看结果
print(result)
```

例 14-2 假设有个 Excel 2007 文件“电影导演演员.xlsx”，其中有三列分别为电影名称、导演和演员列表（同一个电影可能会有多个演员，每个演员姓名之间使用逗号分隔，如图 14-4 所示），要求统计每个演员的参演电影数量，并统计最受欢迎的前 3 个演员。

	A	B	C
1	电影名称	导演	演员
2	电影1	导演1	演员1，演员2，演员3，演员4
3	电影2	导演2	演员3，演员2，演员4，演员5
4	电影3	导演3	演员1，演员5，演员3，演员6
5	电影4	导演1	演员1，演员4，演员3，演员7
6	电影5	导演2	演员1，演员2，演员3，演员8
7	电影6	导演3	演员5，演员7，演员3，演员9
8	电影7	导演4	演员1，演员4，演员6，演员7
9	电影8	导演1	演员1，演员4，演员3，演员8
10	电影9	导演2	演员5，演员4，演员3，演员9
11	电影10	导演3	演员1，演员4，演员5，演员10
12	电影11	导演1	演员1，演员4，演员3，演员11
13	电影12	导演2	演员7，演员4，演员9，演员12
14	电影13	导演3	演员1，演员7，演员3，演员13
15	电影14	导演4	演员10，演员4，演员9，演员14
16	电影15	导演5	演员1，演员8，演员11，演员15
17	电影16	导演6	演员14，演员4，演员13，演员16
18	电影17	导演7	演员3，演员4，演员9
19	电影18	导演8	演员3，演员4，演员10

图 14-4 Excel 文件内容

```
>>> import pandas as pd
>>> df = pd.read_excel('电影导演演员.xlsx')  #从 Excel 文件中读取数据
>>> df
>>> pairs = []
>>> for i in range(len(df)):                    #遍历每一行数据
    actors = df.at[i, '演员'].split(', ')      #获取当前行的演员清单
    for actor in actors:                         #遍历每个演员
        pair = (actor, df.at[i, '电影名称'])
```

```
        pairs.append(pair)
>>> pairs = sorted(pairs, key=lambda item:int(item[0][2:]))
                                                    #按演员编号进行排序
>>> pairs
>>> index = [item[0] for item in pairs]
>>> data = [item[1] for item in pairs]
>>> df1 = pd.DataFrame({'演员':index, '电影名称':data})
>>> result = df1.groupby('演员', as_index=False).count()
                                        #分组，统计每个演员的参演电影数量
>>> result
>>> result.columns = ['演员', '参演电影数量']  #修改列名
>>> result
>>> result.sort_values('参演电影数量')       #对数据进行排序
>>> result.nlargest(3, '参演电影数量') #参演电影数量最多的 3 个演员
```

第 二 节 课

例 14-3 运行下面的程序，在当前文件夹中生成饭店营业额模拟数据文件 data.csv。

```
import csv
import random
import datetime

fn = 'data.csv'

with open(fn, 'w') as fp:
    #创建 csv 文件写入对象
    wr = csv.writer(fp)
    #写入表头
    wr.writerow(['日期', '销量'])

    #生成模拟数据
    startDate = datetime.date(2017, 1, 1)

    #生成 365 个模拟数据，可以根据需要进行调整
    for i in range(365):
        #生成一个模拟数据，写入 csv 文件
        amount = 300 + i*5 + random.randrange(100)
        wr.writerow([str(startDate), amount])
        #下一天
        startDate = startDate + datetime.timedelta(days=1)
```

然后完成下面的任务：

1）使用 pandas 读取文件 data.csv 中的数据，创建 DataFrame 对象，并删除其中所有缺失值；

2）使用 matplotlib 生成折线图，反应该饭店每天的营业额情况，并把图形保存为本地文件 first.jpg；

3）按月份进行统计，使用 matplotlib 绘制柱状图显示每个月份的营业额，并把图形保存为

本地文件 second.jpg；

4）按月份进行统计，找出相邻两个月最大涨幅，并把涨幅最大的月份写入 maxMonth.txt；

5）按季度统计该饭店 2017 年的营业额数据，使用 matplotlib 生成饼状图显示 2017 年 4 个季度的营业额分布情况，并把图形保存为本地文件 third.jpg。

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

#读取数据，丢弃缺失值
df = pd.read_csv('data.csv', encoding='cp936')
df = df.dropna()

#生成并保存营业额折线图
plt.figure()
df.plot(x=df['日期'])
plt.savefig('first.jpg')

#按月统计，生成并保存柱状图
plt.figure()
df1 = df[:]
df1['month'] = df1['日期'].map(lambda x: x[:x.rindex('-')])
df1 = df1.groupby(by='month', as_index=False).sum()
df1.plot(x=df1['month'], kind='bar')
plt.savefig('second.jpg')

#查找涨幅最大的月份，写入文件
df2 = df1.drop('month', axis=1).diff()
m = df2['销量'].nlargest(1).keys()[0]
with open('maxMonth.txt', 'w') as fp:
    fp.write(df1.loc[m, 'month'])

#按季度统计，生成并保存饼状图
plt.figure()
one = df1[:3]['销量'].sum()
two = df1[3:6]['销量'].sum()
three = df1[6:9]['销量'].sum()
four = df1[9:12]['销量'].sum()
plt.pie([one, two, three, four],
        labels=['one', 'two', 'three', 'four'])
plt.savefig('third.jpg')
```