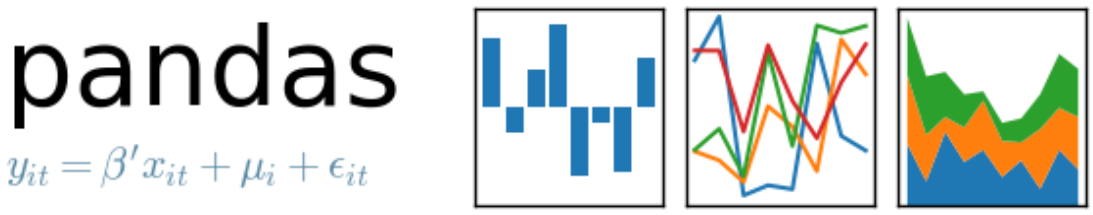
Pandas 百题大冲关



Pandas 是基于 NumPy 的一种数据处理工具，该工具为了解决数据分析任务而创建。Pandas 纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的函数和方法。

Pandas 的数据结构：Pandas 主要有 Series（一维数组），DataFrame（二维数组），Panel（三维数组），Panel4D（四维数组），PanelND（更多维数组）等数据结构。其中 Series 和 DataFrame 应用的最为广泛。

- Series 是一维带标签的数组，它可以包含任何数据类型。包括整数，字符串，浮点数，Python 对象等。Series 可以通过标签来定位。
- DataFrame 是二维的带标签的数据结构。我们可以通过标签来定位数据。这是 NumPy 所没有的。

Pandas 百题大冲关分为基础篇和进阶篇，每部分各有 50 道练习题。基础部分的练习题在于熟悉 Pandas 常用方法的使用，而进阶部分则侧重于 Pandas 方法的组合应用。

如果你在学习课程之前已经有 Pandas 使用基础，那么可以对照着单元格复习一遍。如果你对 Pandas 并不熟练，就一定要亲自动手在每个示例单元格下方的空白单元格中练习。

基础部分

1. 导入 Pandas

练习 Pandas 之前，首先需要导入 Pandas 模块，并约定简称为 `pd`。

⏬ 教学代码：

```
import pandas as pd
```

⏬ 动手练习：

```
# 在空白单元格中重复输入上面的教学代码练习，亲自动手，不要复制粘贴
```

2. 查看 Pandas 版本信息

```
print(pd.__version__)
```

创建 Series 数据类型

Pandas 中，Series 可以被看作由 1 列数据组成的数据集。

创建 Series 语法：`s = pd.Series(data, index=index)`，可以通过多种方式进行创建，以下介绍了 3 个常用方法。

3. 从列表创建 Series

```
arr = [0, 1, 2, 3, 4]
s1 = pd.Series(arr)  # 如果不指定索引，则默认从 0 开始
s1
```

提示：前面的 `0,1,2,3,4` 为当前 Series 的索引，后面的 `0,1,2,3,4` 为 Series 的值。

4. 从 Ndarray 创建 Series

```
import numpy as np
n = np.random.randn(5)  # 创建一个随机 Ndarray 数组

index = ['a', 'b', 'c', 'd', 'e']
s2 = pd.Series(n, index=index)
s2
```

5. 从字典创建 Series

```
d = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
s3 = pd.Series(d)
s3
```

Series 基本操作

6. 修改 Series 索引

```
print(s1)  # 以 s1 为例

s1.index = ['A', 'B', 'C', 'D', 'E']  # 修改后的索引
s1
```

7. Series 纵向拼接

```
s4 = s3.append(s1)  # 将 s1 拼接到 s3
s4
```

8. Series 按指定索引删除元素

```
print(s4)
s4 = s4.drop('e')  # 删除索引为 e 的值
s4
```

9. Series 修改指定索引元素

```
s4['A'] = 6  # 修改索引为 A 的值 = 6
s4
```

10. Series 按指定索引查找元素

```
s4['B']
```

11. Series 切片操作

例如对`s4`的前 3 个数据访问

```
s4[:3]
```

Series 运算

12. Series 加法运算

Series 的加法运算是按照索引计算，如果索引不同则填充为 `NaN`（空值）。

```
s4.add(s3)
```

13. Series 减法运算

Series的减法运算是按照索引对应计算，如果不同则填充为 `NaN`（空值）。

```
s4.sub(s3)
```

14. Series 乘法运算

Series 的乘法运算是按照索引对应计算，如果索引不同则填充为 `NaN`（空值）。

```
s4.mul(s3)
```

15. Series 除法运算

Series 的除法运算是按照索引对应计算，如果索引不同则填充为 `NaN`（空值）。

```
s4.div(s3)
```

16. Series 求中位数

```
s4.median()
```

17. Series 求和

```
s4.sum()
```

18. Series 求最大值

```
s4.max()
```

19. Series 求最小值

```
s4.min()
```

创建 DataFrame 数据类型

与 Sereis 不同，DataFrame 可以存在多列数据。一般情况下，DataFrame 也更加常用。

20. 通过 NumPy 数组创建 DataFrame

```
dates = pd.date_range('today', periods=6)  # 定义时间序列作为 index
num_arr = np.random.randn(6, 4)  # 传入 numpy 随机数组
columns = ['A', 'B', 'C', 'D']  # 将列表作为列名
df1 = pd.DataFrame(num_arr, index=dates, columns=columns)
df1
```

21. 通过字典数组创建 DataFrame

```
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog', 'dog'],
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
df2 = pd.DataFrame(data, index=labels)
df2
```

22. 查看 DataFrame 的数据类型

```
df2.dtypes
```

DataFrame 基本操作

23. 预览 DataFrame 的前 5 行数据

此方法对快速了解陌生数据集结构十分有用。

```
df2.head()  # 默认为显示 5 行，可根据需要在括号中填入希望预览的行数
```

24. 查看 DataFrame 的后 3 行数据

```
df2.tail(3)
```

25.查看 DataFrame 的索引

```
df2.index
```

26. 查看 DataFrame 的列名

```
df2.columns
```

27. 查看 DataFrame 的数值

```
df2.values
```

28. 查看 DataFrame 的统计数据

```
df2.describe()
```

29. DataFrame 转置操作

```
df2.T
```

30. 对 DataFrame 进行按列排序

```
df2.sort_values(by='age')  # 按 age 升序排列
```

31. 对 DataFrame 数据切片

```
df2[1:3]
```

32. 对 DataFrame 通过标签查询（单列）

```
df2['age']
```

```
df2.age  # 等价于 df2['age']
```

33. 对 DataFrame 通过标签查询（多列）

```
df2[['age', 'animal']]  # 传入一个列名组成的列表
```

34. 对 DataFrame 通过位置查询

```
df2.iloc[1:3]  # 查询 2，3 行
```

35. DataFrame 副本拷贝

```
# 生成 DataFrame 副本，方便数据集被多个不同流程使用
df3 = df2.copy()
df3
```

36. 判断 DataFrame 元素是否为空

```
df3.isnull()  # 如果为空则返回为 True
```

37. 添加列数据

```
num = pd.Series([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], index=df3.index)

df3['No.'] = num  # 添加以 'No.' 为列名的新数据列
df3
```

38. 根据 DataFrame 的下标值进行更改。

```
# 修改第 2 行与第 1 列对应的值 3.0 → 2.0
df3.iat[1, 0] = 2  # 索引序号从 0 开始，这里为 1, 0
df3
```

39. 根据 DataFrame 的标签对数据进行修改

```
df3.loc['f', 'age'] = 1.5
df3
```

40. DataFrame 求平均值操作

```
df3.mean()
```

41. 对 DataFrame 中任意列做求和操作

```
df3['visits'].sum()
```

字符串操作

42. 将字符串转化为小写字母

```
string = pd.Series(['A', 'B', 'C', 'Aaba', 'Baca',
                    np.nan, 'CABA', 'dog', 'cat'])
print(string)
string.str.lower()
```

43. 将字符串转化为大写字母

```
string.str.upper()
```

DataFrame 缺失值操作

44. 对缺失值进行填充

```
df4 = df3.copy()
print(df4)
df4.fillna(value=3)
```

45. 删除存在缺失值的行

```
df5 = df3.copy()
print(df5)
df5.dropna(how='any')  # 任何存在 NaN 的行都将被删除
```

46. DataFrame 按指定列对齐

```
left = pd.DataFrame({'key': ['foo1', 'foo2'], 'one': [1, 2]})
right = pd.DataFrame({'key': ['foo2', 'foo3'], 'two': [4, 5]})

print(left)
print(right)

# 按照 key 列对齐连接，只存在 foo2 相同，所以最后变成一行
pd.merge(left, right, on='key')
```

DataFrame 文件操作

47. CSV 文件写入

```
df3.to_csv('animal.csv')
print("写入成功.")
```

48. CSV 文件读取

```
df_animal = pd.read_csv('animal.csv')
df_animal
```

49. Excel 写入操作

```
df3.to_excel('animal.xlsx', sheet_name='Sheet1')
print("写入成功.")
```

50. Excel 读取操作

```
pd.read_excel('animal.xlsx', 'Sheet1', index_col=None, na_values=['NA'])
```

进阶部分

时间序列索引

51. 建立一个以 2018 年每一天为索引，值为随机数的 Series

```
dti = pd.date_range(start='2018-01-01', end='2018-12-31', freq='D')
s = pd.Series(np.random.rand(len(dti)), index=dti)
s
```

52. 统计s 中每一个周三对应值的和

```
# 周一从 0 开始
s[s.index.weekday == 2].sum()
```

53. 统计s中每个月值的平均值

```
s.resample('M').mean()
```

54. 将 Series 中的时间进行转换（秒转分钟）

```
s = pd.date_range('today', periods=100, freq='S')

ts = pd.Series(np.random.randint(0, 500, len(s)), index=s)

ts.resample('Min').sum()
```

55. UTC 世界时间标准

```
s = pd.date_range('today', periods=1, freq='D')  # 获取当前时间
ts = pd.Series(np.random.randn(len(s)), s)  # 随机数值
ts_utc = ts.tz_localize('UTC')  # 转换为 UTC 时间
ts_utc
```

56. 转换为上海所在时区

```
ts_utc.tz_convert('Asia/Shanghai')
```

看一看你当前的时间，是不是一致？

57.不同时间表示方式的转换

```
rng = pd.date_range('1/1/2018', periods=5, freq='M')
ts = pd.Series(np.random.randn(len(rng)), index=rng)
print(ts)
ps = ts.to_period()
print(ps)
ps.to_timestamp()
```

Series 多重索引

58. 创建多重索引 Series

构建一个 `letters = ['A', 'B', 'C']` 和 `numbers = list(range(10))`为索引，值为随机数的多重索引 Series。

```
letters = ['A', 'B', 'C']
numbers = list(range(10))

mi = pd.MultiIndex.from_product([letters, numbers])  # 设置多重索引
s = pd.Series(np.random.rand(30), index=mi)  # 随机数
s
```

59. 多重索引 Series 查询

```
# 查询索引为 1，3，6 的值
s.loc[:, [1, 3, 6]]
```

60. 多重索引 Series 切片

```
s.loc[pd.IndexSlice[:'B', 5:]]
```

DataFrame 多重索引

61. 根据多重索引创建 DataFrame

创建一个以 `letters = ['A', 'B']` 和 `numbers = list(range(6))`为索引，值为随机数据的多重索引 DataFrame。

```
frame = pd.DataFrame(np.arange(12).reshape(6, 2),
                     index=[list('AAABBB'), list('123123')],
                     columns=['hello', 'shiyanlou'])
frame
```

62. 多重索引设置列名称

```
frame.index.names = ['first', 'second']
frame
```

63. DataFrame 多重索引分组求和

```
frame.groupby('first').sum()
```

64. DataFrame 行列名称转换

```
print(frame)
frame.stack()
```

65. DataFrame 索引转换

```
print(frame)
frame.unstack()
```

66. DataFrame 条件查找

```
# 示例数据

data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog', 'dog'],
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
df = pd.DataFrame(data, index=labels)
```

查找 `age` 大于 `3` 的全部信息

```
df[df['age'] > 3]
```

67. 根据行列索引切片

```
df.iloc[2:4, 1:3]
```

68. DataFrame 多重条件查询

查找 `age<3` 且为 `cat` 的全部数据。

```
df = pd.DataFrame(data, index=labels)

df[(df['animal'] == 'cat') & (df['age'] < 3)]
```

69. DataFrame 按关键字查询

```
df3[df3['animal'].isin(['cat', 'dog'])]
```

70. DataFrame 按标签及列名查询。

```
df.loc[df2.index[[3, 4, 8]], ['animal', 'age']]
```

71. DataFrame 多条件排序

按照 `age` 降序，`visits` 升序排列

```
df.sort_values(by=['age', 'visits'], ascending=[False, True])
```

72.DataFrame 多值替换

将 `priority` 列的 `yes` 值替换为 `True`，`no` 值替换为 `False`。

```
df['priority'].map({'yes': True, 'no': False})
```

73. DataFrame 分组求和

```
df4.groupby('animal').sum()
```

74. 使用列表拼接多个 DataFrame

```
temp_df1 = pd.DataFrame(np.random.randn(5, 4))  # 生成由随机数组成的 DataFrame 1
temp_df2 = pd.DataFrame(np.random.randn(5, 4))  # 生成由随机数组成的 DataFrame 2
temp_df3 = pd.DataFrame(np.random.randn(5, 4))  # 生成由随机数组成的 DataFrame 3

print(temp_df1)
print(temp_df2)
print(temp_df3)

pieces = [temp_df1, temp_df2, temp_df3]
pd.concat(pieces)
```

75. 找出 DataFrame 表中和最小的列

```
df = pd.DataFrame(np.random.random(size=(5, 10)), columns=list('abcdefghij'))
print(df)
df.sum().idxmin()  # idxmax(), idxmin() 为 Series 函数返回最大最小值的索引值
```

76. DataFrame 中每个元素减去每一行的平均值

```
df = pd.DataFrame(np.random.random(size=(5, 3)))
print(df)
df.sub(df.mean(axis=1), axis=0)
```

77. DataFrame 分组，并得到每一组中最大三个数之和

```
df = pd.DataFrame({'A': list('aaabbcaabcccbbc'),
                   'B': [12, 345, 3, 1, 45, 14, 4, 52, 54, 23, 235, 21, 57, 3, 87]})
print(df)
df.groupby('A')['B'].nlargest(3).sum(level=0)
```

透视表

当分析庞大的数据时，为了更好的发掘数据特征之间的关系，且不破坏原数据，就可以利用透视表 `pivot_table` 进行操作。

78. 透视表的创建

新建表将 `A, B, C` 列作为索引进行聚合。

```
df = pd.DataFrame({'A': ['one', 'one', 'two', 'three'] * 3,
                   'B': ['A', 'B', 'C'] * 4,
                   'C': ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] * 2,
                   'D': np.random.randn(12),
                   'E': np.random.randn(12)})

print(df)

pd.pivot_table(df, index=['A', 'B'])
```

79. 透视表按指定行进行聚合

将该 DataFrame 的 `D` 列聚合，按照 `A,B` 列为索引进行聚合，聚合的方式为默认求均值。

```
pd.pivot_table(df, values=['D'], index=['A', 'B'])
```

80. 透视表聚合方式定义

上一题中 `D` 列聚合时，采用默认求均值的方法，若想使用更多的方式可以在 `aggfunc` 中实现。

```
pd.pivot_table(df, values=['D'], index=['A', 'B'], aggfunc=[np.sum, len])
```

81. 透视表利用额外列进行辅助分割

`D` 列按照 `A,B` 列进行聚合时，若关心 `C` 列对 `D` 列的影响，可以加入 `columns` 值进行分析。

```
pd.pivot_table(df, values=['D'], index=['A', 'B'],
               columns=['C'], aggfunc=np.sum)
```

82. 透视表的缺省值处理

在透视表中由于不同的聚合方式，相应缺少的组合将为缺省值，可以加入 `fill_value` 对缺省值处理。

```
pd.pivot_table(df, values=['D'], index=['A', 'B'],
               columns=['C'], aggfunc=np.sum, fill_value=0)
```

绝对类型

在数据的形式上主要包括数量型和性质型，数量型表示着数据可数范围可变，而性质型表示范围已经确定不可改变，绝对型数据就是性质型数据的一种。

83. 绝对型数据定义

```
df = pd.DataFrame({"id": [1, 2, 3, 4, 5, 6], "raw_grade": [
                   'a', 'b', 'b', 'a', 'a', 'e']})
df["grade"] = df["raw_grade"].astype("category")
df
```

84. 对绝对型数据重命名

```
df["grade"].cat.categories = ["very good", "good", "very bad"]
df
```

85. 重新排列绝对型数据并补充相应的缺省值

```
df["grade"] = df["grade"].cat.set_categories(
    ["very bad", "bad", "medium", "good", "very good"])
df
```

86. 对绝对型数据进行排序

```
df.sort_values(by="grade")
```

87. 对绝对型数据进行分组

```
df.groupby("grade").size()
```

数据清洗

常常我们得到的数据是不符合我们最终处理的数据要求，包括许多缺省值以及坏的数据，需要我们对数据进行清洗。

88. 缺失值拟合

在`FilghtNumber`中有数值缺失，其中数值为按 10 增长，补充相应的缺省值使得数据完整，并让数据为 `int` 类型。

```
df = pd.DataFrame({'From_To': ['LoNDon_paris', 'MAdrid_miLAN', 'londON_StockhOlm',
                               'Budapest_PaRis', 'Brussels_londOn'],
                   'FlightNumber': [10045, np.nan, 10065, np.nan, 10085],
                   'RecentDelays': [[23, 47], [], [24, 43, 87], [13], [67, 32]],
                   'Airline': ['KLM(!)', '<Air France> (12)', '(British Airways. )',
                               '12. Air France', '"Swiss Air"']})
df['FlightNumber'] = df['FlightNumber'].interpolate().astype(int)
df
```

89. 数据列拆分

其中From_to应该为两独立的两列From和To，将From_to依照_拆分为独立两列建立为一个新表。

```
temp = df.From_To.str.split('_', expand=True)
temp.columns = ['From', 'To']
temp
```

90. 字符标准化

其中注意到地点的名字都不规范（如：londON应该为London）需要对数据进行标准化处理。

```
temp['From'] = temp['From'].str.capitalize()
temp['To'] = temp['To'].str.capitalize()
```

91. 删除坏数据加入整理好的数据

将最开始的 From_to 列删除，加入整理好的 From 和 to 列。

```
df = df.drop('From_To', axis=1)
df = df.join(temp)
print(df)
```

92. 去除多余字符

如同 airline 列中许多数据有许多其他字符，会对后期的数据分析有较大影响，需要对这类数据进行修正。

```
df['Airline'] = df['Airline'].str.extract(
    '([a-zA-Z\s]+)', expand=False).str.strip()
df
```

93. 格式规范

在 RecentDelays 中记录的方式为列表类型，由于其长度不一，这会为后期数据分析造成很大麻烦。这里将 RecentDelays 的列表拆开，取出列表中的相同位置元素作为一列，若为空值即用 NaN 代替。

```
delays = df['RecentDelays'].apply(pd.Series)

delays.columns = ['delay_{}'.format(n)
                  for n in range(1, len(delays.columns)+1)]

df = df.drop('RecentDelays', axis=1).join(delays)
df
```

数据预处理

94. 信息区间划分

班级一部分同学的数学成绩表，如下图所示

```
df=pd.DataFrame({'name':['Alice','Bob','Candy','Dany','Ella','Frank','Grace','Jenny'],'grades':[58,83,79,65,93,45,61,88]})
```

但我们更加关心的是该同学是否及格，将该数学成绩按照是否>60来进行划分。

```
df = pd.DataFrame({'name': ['Alice', 'Bob', 'Candy', 'Dany', 'Ella',
                            'Frank', 'Grace', 'Jenny'], 'grades': [58, 83, 79, 65, 93, 45, 61, 88]})


def choice(x):
    if x > 60:
        return 1
    else:
        return 0


df.grades = pd.Series(map(lambda x: choice(x), df.grades))
df
```

95. 数据去重

一个列为A的 DataFrame 数据，如下图所示

```
df = pd.DataFrame({'A': [1, 2, 2, 3, 4, 5, 5, 5, 6, 7, 7]})
```

尝试将 A 列中连续重复的数据清除。

```
df = pd.DataFrame({'A': [1, 2, 2, 3, 4, 5, 5, 5, 6, 7, 7]})
df.loc[df['A'].shift() != df['A']]
```

96. 数据归一化

有时候，DataFrame 中不同列之间的数据差距太大，需要对其进行归一化处理。

其中，Max-Min 归一化是简单而常见的一种方式，公式如下:

$$Y = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
def normalization(df):
    numerator = df.sub(df.min())
    denominator = (df.max()).sub(df.min())
    Y = numerator.div(denominator)
    return Y


df = pd.DataFrame(np.random.random(size=(5, 3)))
print(df)
normalization(df)
```

Pandas 绘图操作

为了更好的了解数据包含的信息，最直观的方法就是将其绘制成图。

97. Series 可视化

```
%matplotlib inline
ts = pd.Series(np.random.randn(100), index=pd.date_range('today', periods=100))
ts = ts.cumsum()
ts.plot()
```

98. DataFrame 折线图

```
df = pd.DataFrame(np.random.randn(100, 4), index=ts.index,
                  columns=['A', 'B', 'C', 'D'])
df = df.cumsum()
df.plot()
```

99. DataFrame 散点图

```
df = pd.DataFrame({"xs": [1, 5, 2, 8, 1], "ys": [4, 2, 1, 9, 6]})
df = df.cumsum()
df.plot.scatter("xs", "ys", color='red', marker="*")
```

100. DataFrame 柱形图

```
df = pd.DataFrame({"revenue": [57, 68, 63, 71, 72, 90, 80, 62, 59, 51, 47, 52],
                   "advertising": [2.1, 1.9, 2.7, 3.0, 3.6, 3.2, 2.7, 2.4, 1.8, 1.6, 1.3, 1.9],
                   "month": range(12)
                   })

ax = df.plot.bar("month", "revenue", color="yellow")
df.plot("month", "advertising", secondary_y=True, ax=ax)
```

实验总结

如果你亲自动手做完了上面的 100 道练习题，相信你已经对 Pandas 模块的熟练程度又提升了不少。我们推荐你定期回顾这些题目，相信你一定会熟能生巧。本次实验涉及的知识点主要有：

- 创建 Series
- Series 基本操作
- 创建 DataFrame
- DataFrame 基本操作
- DataFrame 文件操作
- Series，DataFrame 和多索引
- 透视表
- 数据清洗
- 数据预处理
- 可视化

实验中少量题目编译自：[100 pandas puzzles](https://github.com/ajcr/100-pandas-puzzles) 。