06- pandas分组聚合

1.函数应用和映射

apply()用法

用于DataFrame对象，通常用于对DataFrame的行或列应用函数。

pandas.apply() 方法用于在 Pandas Series 或 DataFrame 上的应用一个函数。这个方法的参数如下：

1. **func** (必需)：这是要应用的函数，可以是一个 Python 函数、lambda 函数或可调用对象。这个函数将应用于 Series 或 DataFrame 的每个元素或行/列，具体取决于 **axis** 参数的设置。
2. **axis** (可选)：指定函数应用的轴方向。有两个选项：
    - **axis=0**（默认值）：将函数应用于每一列，即按列方向操作。
    - **axis=1**：将函数应用于每一行，即按行方向操作。
3. **raw** (可选)：一个布尔值，用于控制是否将数据以 NumPy 数组的形式传递给函数。默认情况下，**raw=False**，表示将数据以 Pandas Series 或 DataFrame 的形式传递给函数。如果将 **raw=True**，则数据以 NumPy 数组形式传递给函数，这在某些情况下可能提高性能。
4. **result_type** (可选)：指定函数的返回类型。有两个选项：
    - **'expand'**（默认值）：如果函数返回的是 Series，则将其扩展为 DataFrame。
    - **'reduce'**：如果函数返回的是标量（如一个数字），则返回一个标量；如果返回的是 Series，则返回一个 Series。
5. **args** (可选)：一个元组，包含传递给函数的额外参数。这可以用来向函数传递额外的参数。
6. ****kwds** (可选)：关键字参数，用于传递给函数的额外关键字参数。

示例

```python
import pandas as pd

# 创建一个示例 DataFrame
data = {'A': [1, 2, 3, 4, 5],
        'B': [10, 20, 30, 40, 50]}
df = pd.DataFrame(data)
print("原df\n", df)
```

1、 使用 lambda 函数将每个元素加倍

```python
df['A2'] = df['A'].apply(lambda x: x * 2)
print("A列2倍处理后的df\n", df)
```

2、增加新列Row_Sum，值为A列和B列对应位置相加的结果

```python
def row_sum(row):
    return row['A'] + row['B']
df['Row_Sum'] = df.apply(row_sum, axis=1)
print("增加AB两列和后的df\n", df)
```

map()用法

map() 方法用于在 Pandas Series 上映射函数、字典或其他可映射对象。这个方法的参数如下：

1. **arg** (必需)：这是映射函数、字典或其他可映射对象，用于将 Series 中的元素映射到新的值。可以是以下几种类型：
   - 一个函数：将应用于 Series 中的每个元素，函数的返回值将作为新的值。
   - 一个字典：将 Series 中的值与字典中的键进行匹配，并将对应的值用作新的值。
   - 一个 Series 或其他可映射的 Pandas 对象：将 Series 中的值与可映射对象中的索引或标签匹配，并将对应的值用作新的值。
2. **na_action** (可选)：指定对于 Series 中的缺失值如何处理。有两个选项：
   - **'ignore'**（默认值）：忽略缺失值，不进行映射，将缺失值保留不变。
   - **'raise'**：如果 Series 中存在缺失值，则引发异常。

示例

Python

```python
import pandas as pd

# 创建一个示例 Series
data = {'A': ['apple', 'banana', 'cherry', 'date']}
s = pd.Series(data['A'])
# 使用字典映射元素到新的值
dict1 = {'apple': 'fruit', 'banana': 'fruit', 'cherry': 'fruit'}
s_mapped = s.map(dict1)
print(s_mapped)
```

上面匹配不到的，有空值，加一个判断，处理一下。

```python
# 使用函数映射元素到新的值
def func(x):
    if x in ['apple', 'banana', 'cherry']:
        return 'fruit'
    else:
        return 'other'
s_mapped2= s.map(func)
print(s_mapped2)
```

2.汇总和描述统计

Python

```python
import pandas as pd

# 创建一个示例 DataFrame
data = {'A': [1, 2, 3, 4, 5],
        'B': [10, 20, 30, 40, 50]}
df = pd.DataFrame(data)

# 计算平均值
mean_A = df['A'].mean()
mean_B = df['B'].mean()
print(f'A列平均值: {mean_A}')
print(f'B列平均值: {mean_B}')
# 计算中位数
median_A = df['A'].median()
median_B = df['B'].median()
print(f'A列中位数: {median_A}')
print(f'B列中位数: {median_B}')
# 计算总和
sum_A = df['A'].sum()
sum_B = df['B'].sum()
print(f'A列求和: {sum_A}')
print(f'B列求和: {sum_B}')
# 找到最小值
min_A = df['A'].min()
min_B = df['B'].min()
print(f'A列最小值: {min_A}')
print(f'B列最小值: {min_B}')
# 找到最大值
max_A = df['A'].max()
max_B = df['B'].max()
print(f'A列最大值: {max_A}')
print(f'B列最大值: {max_B}')
# 计算标准差
```

```python
std_A = df['A'].std()
std_B = df['B'].std()
print(f'标准差 A: {std_A}')
print(f'S标准差 B: {std_B}')
# 计算方差
var_A = df['A'].var()
var_B = df['B'].var()
print(f'Variance of A: {var_A}')
print(f'Variance of B: {var_B}')
# 计算非空值的数量
count_A = df['A'].count()
count_B = df['B'].count()
print(f'A列非空值数量: {count_A}')
print(f'B列非空值数量: {count_B}')
# 生成摘要统计信息
summary_A = df['A'].describe()
summary_B = df['B'].describe()
print(f'概览:\n{summary_A}')
print(f'概览:\n{summary_B}')
# 计算唯一值的频率
value_counts_A = df['A'].value_counts()
value_counts_B = df['B'].value_counts()
print(f'A列各值计数:\n{value_counts_A}')
print(f'B列各值计数:\n{value_counts_B}')
```

2 str 属性

```python
import pandas as pd

data = {'name': ['Alice', 'Bob', 'Charlie']}
df = pd.DataFrame(data)
# str.len(): 计算每个字符串的长度。
df['name_length'] = df['name'].str.len()
print(df)
# str.lower() 和 str.upper(): 将字符串转换为小写或大写。
df['name_lower'] = df['name'].str.lower()
df['name_upper'] = df['name'].str.upper()
print(df)
# str.replace(): 替换字符串中的子字符串
df['name_replaced'] = df['name'].str.replace('a', 'X')
print(df)

# str.strip()、str.lstrip() 和 str.rstrip(): 删除字符串两侧、左侧或右侧的空格。
df['name_strip'] = df['name'].str.strip()
df['name_lstrip'] = df['name'].str.lstrip()
df['name_rstrip'] = df['name'].str.rstrip()
print(df)
# str.split(): 拆分字符串为列表。
df['name_split'] = df['name'].str.split(' ')
print(df)
# str.startswith() 和 str.endswith(): 检查字符串是否以特定前缀或后缀开头。
df['name_startswith'] = df['name'].str.startswith('A')
df['name_endswith'] = df['name'].str.endswith('e')
print(df)
```

3 分组聚合

groupby()

groupby() 是 Pandas 中一个强大的方法，用于将数据按照一个或多个列的值分组，然后对每个分组应用聚合函数。以下是 **groupby()** 方法的主要参数说明：

1. **by** (必需)：指定分组的列名或列名的列表。可以是单个列名的字符串，也可以是列名的列表，以按多列分组。这是 **groupby()** 方法的关键参数。
2. **axis** (可选)：指定分组的轴方向，有两个选项：
    - **axis=0**（默认值）：按行分组。
    - **axis=1**：按列分组。
3. **level** (可选)：如果输入 DataFrame 包含多层索引，则可以指定要分组的索引级别。
4. **as_index** (可选)：默认情况下，**groupby()** 结果的分组键会成为结果 DataFrame 的索引。设置 **as_index=False** 可以防止这种情况发生，分组键将保留为列而不是索引。
5. **sort** (可选)：默认情况下，分组键会根据分组键的值进行排序。设置 **sort=False** 可以禁用排序，可能提高性能。

```python
import pandas as pd

# 创建一个示例 DataFrame
data = {'Category': ['A', 'B', 'A', 'B', 'A'],
        'Value': [10, 20, 15, 25, 30]}
df = pd.DataFrame(data)
# 按 Category 列分组，并计算每个分组的平均值
grouped = df.groupby(by='Category')
res1 = grouped.mean()
print(res1)
# 使用多列进行分组
grouped = df.groupby(by=['Category', 'Value'])
# 聚合求和
res2 = grouped.sum()
print(res2)
# 使用自定义聚合函数
def custom_agg(x):
    return x.max() - x.min()
res3 = df.groupby(by='Category').agg(custom_agg)
print(res3)
```

4 透视表

pivot_table()

pivot_table() 是 Pandas 中用于创建数据透视表的方法。数据透视表是一种用于总结和聚合数据的强大工具，可以根据一个或多个列对数据进行重新排列，以便更容易进行分析。以下是 **pivot_table()** 方法的主要参数说明：

1. **data** (必需)：要创建数据透视表的 DataFrame。
2. **values** (必需)：要聚合的列名或列名的列表。这是你要计算统计量的列。
3. **index** (可选)：一个或多个列名，用于作为数据透视表的行索引（行标签）。
4. **columns** (可选)：一个或多个列名，用于作为数据透视表的列索引（列标签）。
5. **aggfunc** (可选)：要应用于 **values** 中列的聚合函数，可以是内置聚合函数（例如 'sum'、'mean'、'count' 等）或自定义函数。默认情况下，使用 'mean'。
6. **fill_value** (可选)：替代结果表中的缺失值。
7. **margins** (可选)：添加边际汇总，生成汇总统计信息。
8. **dropna** (可选)：默认情况下，如果所有条目都是 NaN，则删除相应的行。设置为 False 可以保留包含 NaN 值的行。

Python

```python
import pandas as pd

# 创建一个示例 DataFrame
data = {'Date': ['2023-01-01', '2023-01-01', '2023-01-02', '2023-01-02', '2023-01-03'],
        'Category': ['A', 'B', 'A', 'B', 'A'],
        'Value': [10, 20, 15, 25, 30]}
df = pd.DataFrame(data)
print(df)
# 创建数据透视表，聚合 'Value' 列，行索引为 'Date'，列索引为 'Category'，默认聚合函数为均值
pivot = pd.pivot_table(df, values='Value', index='Date', columns='Category')
print(pivot)
```