

Pandas数据处理进阶

讲师名称：李川

Can//ay 嘉为数字咨询

数 字 化 人 才 培 养 先 行 者



目录

01.
重复与缺失值处理

02.
异常值处理及数据标
准化

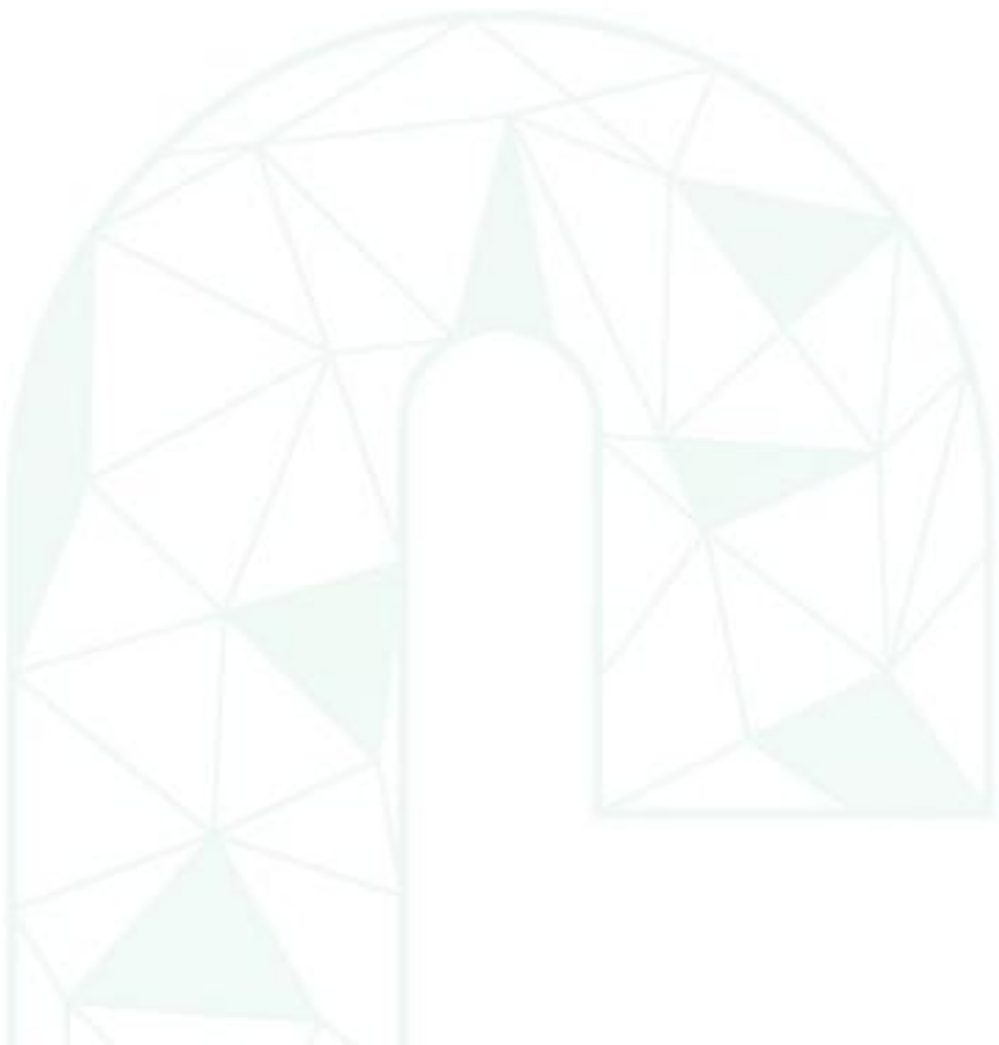
03.
数据类型转换

04.
数据抽取

05.
数据更新

06.
数据统计分析

CONTENTS



/01

重复与缺失值处理

1.1 检测重复数据

在Pandas中，可通过duplicated()方法来判断有无重复值。

语法格式如下：

```
DataFrame_obj.duplicated(subset=None, keep="first") 或  
Series_obj.duplicated(keep="first")
```

参数说明：

subset：列标签或列标签的列表，表示需要检测重复的列，默认为全部的列。

keep：可选项'first', 'last', False,默认为'first',表示将重复项标记True，第一次出现的除外;'last'表示将重复项标记True，最后一次除外；False表示将所有发生重复的数据标记为True。

1.2去除重复数据

在Pandas中，可以通过drop_duplicates()方法对数据去重。

语法格式如下：

```
DataFrame_obj.drop_duplicates(subset=None, keep="first", inplace=False, ignore_index=False)
```

参数说明：

subset：列标签或者列标签的列表，表示要去重的列

keep：可选项'first', 'last', False，默认'first'，表示去重后只保留第一个；'last'表示去重后保留最后一个；False表示有重复都不保留。

inplace：是否对原表去重操作，默认False，此时会返回一个去重后的新DataFrame，原表数据不变，如果设为True，则会对原表会被去重。

ignore_index：重置索引，默认False，如果设置为True，则会重新生成从0开始的连续的索引。

1.3 缺失值的检测

Pandas中可以使用isnull()或notnull()找到缺失值
格式语法如下：

```
DataFrame_obj.isnull()
```

1.4 删除缺失值

Pandas中提供了简便的删除缺失值的方法dropna，该方法既可以删除观测记录，亦可以删除特征。格式语法如下：

```
DataFrame_obj.dropna(axis=0, how="any", thresh=None, subset=None, inplace=False)
```

参数说明：

axis：指定要删行还是删列，默认为0，0或'index'表示删行，1或'columns'表示删除列

how：可选项'any', 'all'，默认为'any'，表示一行或一列只要有一个是缺失值。

thresh：非空元素最低数量。int型，默认为None。如果该行/列中，非空元素数量小于这个值，就删除该行/列。

subset：索引的列表，表示哪些行或列需要删除缺失值，当axis=0时，subset中元素为列的索引；当axis=1时，subset中元素为行的索引。

inplace：是否在原表操作，默认为False，如果设置为True，则在原表上进行操作。

1.5 替换缺失值

Pandas中可以通过DataFrame对象的fillna()方法替换缺失值。格式语法如下：

```
DataFrame_obj.fillna(value=None, method=None, axis=None, inplace=False, limit=None,
downcast=None)
```

参数说明：

value：用于填充的空值的值。

method：可选项'backfill', 'bfill', 'pad', 'ffill', None。

axis：轴。0或'index'，表示按行填充；1或'columns'，表示按列填充。

inplace：是否原表操作。布尔值，默认为False。

limit：整数，默认为None。

downcast：默认为None，表示类型向下转换规则。如果为字符串“infer”，此时会在合适的等价类型之间进行向下转换，比如由float64转为int64。



/02

异常值处理及数据标准化

2.1 异常值识别

核心目标：发现偏离正常范围的数据（可能是测量错误或真实极端值）。

常用方法：

IQR 法（四分位距法）：

步骤 1：计算第一四分位（ $Q1$ ，25% 分位数）和第三四分位（ $Q3$ ，75% 分位数）。

步骤 2：计算 $IQR = Q3 - Q1$ 。

步骤 3：异常值范围 = 小于 $Q1 - 1.5 \times IQR$ 或 大于 $Q3 + 1.5 \times IQR$ 。

优点：不受极端值影响，适用于非正态分布数据（如工业传感器数据）。

Z-score 法：

步骤 1：计算均值（ μ ）和标准差（ σ ）。

步骤 2：异常值范围 = 小于 $\mu - 3\sigma$ 或 大于 $\mu + 3\sigma$ （ 3σ 原则）。

优点：计算简单，适用于近似正态分布的数据。

2.2 异常值处理

核心目标：修正异常值，减少对分析的干扰。

常用方法：

截断法（clip）：将异常值替换为合理范围的上下限。

优点：保留数据趋势，适合工业场景（异常可能是传感器瞬间波动）。

均值 / 中位数替换：用该列的均值或中位数覆盖异常值。

优点：操作简单，适合异常值比例极低的情况。

删除行：仅适用于异常值是错误数据（如录入错误）且比例极低的情况。

单独标记：若异常值有实际业务意义（如设备故障），可标记后保留。

2.3 数据标准化

核心目标：消除不同指标的量纲影响（如温度单位°C和压力单位 MPa），使数据可比较。

常用方法：

Z-score 标准化：

公式：标准化值 = (原始值 - 均值) / 标准差

特点：结果均值为 0，标准差为 1，保留数据的分布形态和波动幅度。

适用场景：需要保留数据相对差异（如分析温度和压力的波动相关性）。

Min-Max 标准化：

公式：标准化值 = (原始值 - 最小值) / (最大值 - 最小值)

特点：结果压缩到 [0,1] 区间，消除绝对数值差异。

适用场景：需要统一数据范围（如机器学习输入、指标权重计算）。

2.4 数据预处理实例

例2-1 对企业工业传感器数据进行数据清洗预处理，主要包括缺失值

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# -----
# 1. 数据准备（含重复值、缺失值、异常值）
# -----
data = {
    '设备ID': ['EQ01', 'EQ02', 'EQ03', 'EQ02', 'EQ05', 'EQ06', 'EQ03', 'EQ08'],
    '温度': [85, 88, None, 88, 89, 92, None, 86], # 含重复、缺失
    '压力': [2.1, 2.3, 2.2, 2.3, 2.5, 9.9, 2.2, 2.1], # 含重复、异常
    '产量': [1200, 1250, 1180, 1250, 1230, 1190, 1180, 1210]
}
df = pd.DataFrame(data)
print("原始数据：")
print(df, "\n")
```

2.4 数据预处理实例

例2-1 对企业工业传感器数据进行数据清洗预处理，主要包括缺失值

```
# -----  
# 2. 重复值处理  
# -----  
# 2.1 检测重复值（完全重复的行）  
duplicates = df.duplicated()  
print("重复值标记（True表示重复）： ")  
print(duplicates.tolist(), "\n")  
# 2.2 查看重复行详情  
print("重复行数据： ")  
print(df[duplicates], "\n")  
  
# 2.3 删除重复值（保留第一次出现的行）  
df = df.drop_duplicates(keep='first')  
print("删除重复值后的数据： ")  
print(df.reset_index(drop=True), "\n") # 重置索引
```

2.4 数据预处理实例

例2-1 对企业工业传感器数据进行数据清洗预处理，主要包括缺失值

```
# -----  
# 3. 缺失值处理  
# -----  
# 查看缺失值分布  
print("缺失值统计: ")  
print(df.isnull().sum(), "\n") # 仅温度列有1个缺失值  
#df.isnull()  
# 选择填充方法（根据数据类型和业务场景）  
# 方法1：数值型数据用均值填充（适用于分布平稳的数据）  
df['温度'] = df['温度'].fillna(df['温度'].mean().round(1))  
  
print("缺失值处理后: ")  
print(df[['设备ID', '温度']], "\n")
```

2.4 数据预处理实例

4. 异常值识别与处理

```
def clean_outliers(df, col):
    """用IQR法识别并处理异常值"""
    # 计算四分位
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr # 下限
    upper = q3 + 1.5 * iqr # 上限
    # 识别异常值
    outliers = df[(df[col] < lower) | (df[col] > upper)][col]
    print(f"{col}异常值: {outliers.tolist()}")
    # 处理异常值 (用上下限截断, 保留数据分布)
    df[col] = df[col].clip(lower, upper)
    return df

# 处理温度和压力列的异常值
for col in ['温度', '压力']:
    df = clean_outliers(df, col)
print("\n异常值处理后: ")
print(df[['设备ID', '温度', '压力']], "\n")
```


2.4 数据预处理实例

```
# 5. 数据标准化
# 选择需要标准化的数值列
numeric_cols = ['温度', '压力', '产量']
X = df[numeric_cols]
# 方法1: Z-score标准化 (均值0, 标准差1, 保留分布特征)
z_scaler = StandardScaler()
df['温度_Z'] = z_scaler.fit_transform(X[['温度']])
df['压力_Z'] = z_scaler.fit_transform(X[['压力']])
df['产量_Z'] = z_scaler.fit_transform(X[['产量']])
# 方法2: Min-Max标准化 (缩放到0-1, 消除量纲)
mm_scaler = MinMaxScaler()
df['温度_01'] = mm_scaler.fit_transform(X[['温度']])
df['压力_01'] = mm_scaler.fit_transform(X[['压力']])
df['产量_01'] = mm_scaler.fit_transform(X[['产量']])
# 输出最终结果
print("标准化后数据 (部分) : ")
print(df[['设备ID', '温度', '温度_Z', '温度_01']].round(3))
```



/03

数据类型转换

3.1 数据类型转换

在 pandas 中，`astype()` 是用于数据类型转换的核心方法，能将 Series 或 DataFrame 中的数据转换为指定类型（如整数、浮点数、字符串等）。它在数据清洗中非常常用，例如将字符串格式的日期转为日期类型、将数值型 ID 转为字符串等。

`astype()` 基本用法

语法：Series.astype(dtype, copy=True, errors='raise')

dtype: 目标数据类型（如 int、float、str、'category' 等）

errors: 错误处理方式（'raise' 抛出错误，'ignore' 忽略错误）

3.2 数据类型转换常见问题

转换失败的常见原因：

字符串中包含非数值字符（如 '100元' 转 int 会报错）

存在 NaN 或缺失值（int 类型不支持 NaN，需用 'Int64' nullable 类型）

混合类型数据（如同时含数字和文字的列）

解决方法：先清洗异常值（如用 `replace()` 处理非数值字符），再转换类型。

特殊类型转换：

日期类型：`astype()` 不直接支持，需用 `pd.to_datetime()`

nullable 类型：处理含 NaN 的整数列时，用 'Int64'（大写 I）替代 int，例如：

```
df['数量'] = df['数量'].astype('Int64') # 支持NaN的整数类型
```

3.3 数据类型转换实例

例2-2 核工业燃料棒生产数据处理的实例

1. 原始数据准备 (模拟燃料棒生产记录)

```
data = {  
    '燃料棒编号': ['FR-2023-001-U', 'FR-2023-002-M', 'FR-2023-003-U',  
                  'FR-2023-004-U', 'FR-2023-005-M', 'FR-2023-006-U'],  
    '生产信息': ['2023-10-01,一车间,合格', '2023-10-03,二车间,合格',  
                 '2023-10-05,一车间,待检测', '2023-10-07,三车间,合格',  
                 '2023-10-09,二车间,不合格', '2023-10-11,一车间,合格'],  
    '铀浓度': ['4.5%', '7.2%', '4.8%', '5.0%', '8.1%', '4.7%'],  
    '长度(mm)': ['3800', '3802', '3799', '3801', '3798', '3800'],  
    '检测数据': ['102.5,0.003', '98.7,0.002', '101.3,0.004',  
                 '99.8,0.003', '105.2,0.005', '100.1,0.002']  
}
```

```
df = pd.DataFrame(data)
```

```
print("=== 原始数据 ===")
```

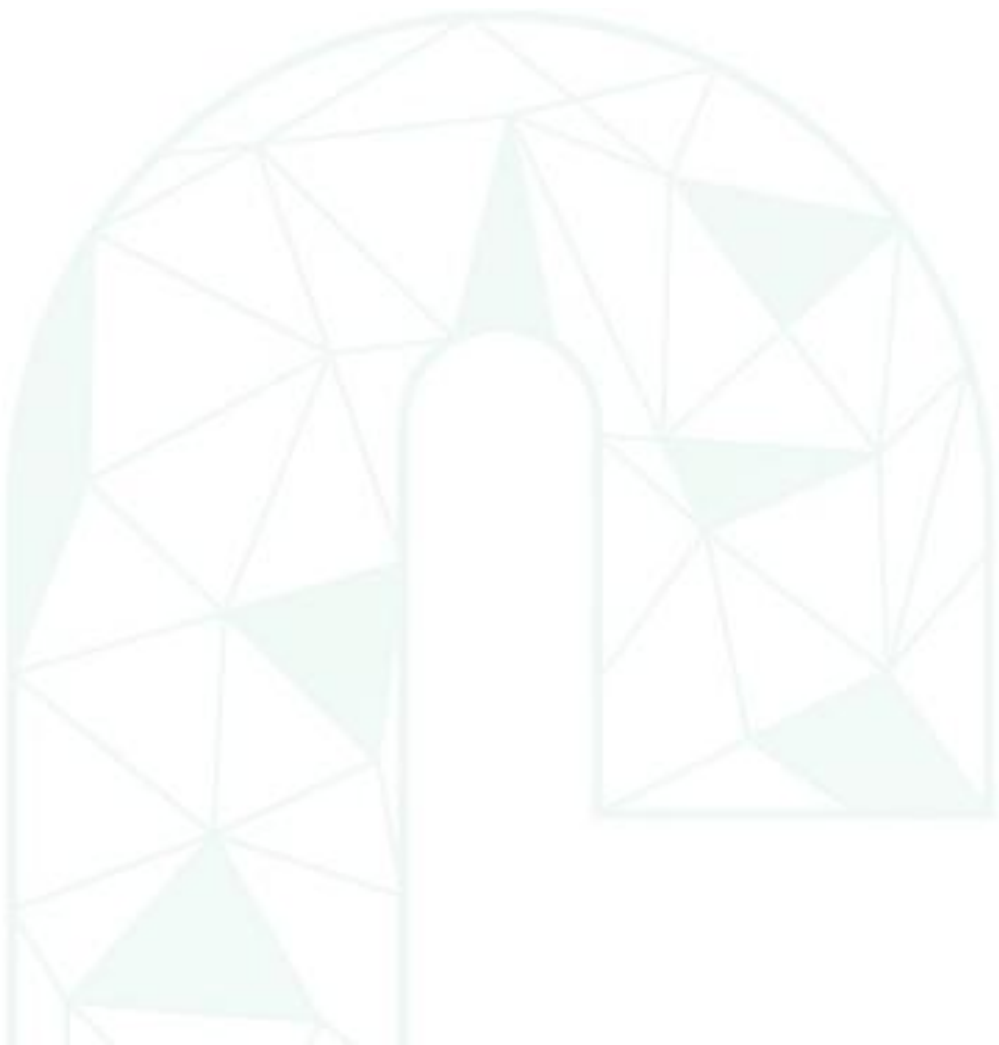
```
print(df, "\n")
```

2. 数据类型转换

```
df['铀浓度'] = df['铀浓度'].str.replace('%', '').astype(float) # 字符串转浮点
```

```
df['长度(mm)'] = df['长度(mm)'].astype(int) # 字符串转整数
```

```
print(df[['燃料棒编号', '铀浓度', '长度(mm)']], "\n")
```



/04

数据抽取

4 数据抽取

1. 字段抽取
2. 字段拆分
3. 重置索引
4. 记录抽取
5. 随机抽取

4.1 字段抽取

字段抽取表示抽取某列上指定位置的数据。语法格式如下：

```
slice(start, stop)
```

例如：抽取数据中手机号的前3位

```
df['手机号'].str.slice(0,3)
```

`extract()` 是 Series 的字符串方法，用于从字符串中提取符合正则表达式模式的子串，常用来从复杂字符串中抽取关键信息（如日期、编号、特定标识等）。

核心语法如下：

```
Series.str.extract(pat, expand=True)
```


4.2 字段拆分

字段拆分表示按照指定的sep字符拆分某列数据。语法格式如下：

```
split(sep ,n, expand)
```

Sep:分隔符

N:分割后新增的列

Expand：是否展开为数据框，默认false

例如：分割IP地址，判断ip类型

```
df['ip'].str.split('.',1,true)
```

4.3 重置索引

重置索引是指指定某一列为数据索引。语法格式如下：

```
Df.set_index('列')
```

例如：指定姓名为索引

```
Df.set_index('姓名')
```

4.4 记录抽取

记录抽取表示按照一定条件筛选指定数据。语法格式如下：

```
Df[condition]
```

例如：筛选年龄在20到30之间的员工

```
Df[df.age.between(20,30)]
```

4.5 随机抽取

随机抽取表示随机的从数据中抽取一定数据。语法格式如下：

```
Ny.random.randint(start,end,num)
```

例如：从题库的前100题中随机抽取10个题目

```
Ny.random.randint(0,100,10)
```

4.5 随机抽取 (续)

注意：随机抽取返回的是数据的行索引。想要获取完整数据需要用到使用索引获取数据：

```
df.iloc[行索引号, 列索引号]
```

例如：从题库的前100题中随机抽取10个题目

```
Harr=np.random.randint(0,100,10)
```

```
df.iloc[Harr, :]
```

4.5 随机抽取 (续)

在 pandas 中, `df.sample()` 用于从 DataFrame 中随机抽取样本, 是数据采样、随机测试和样本验证的常用工具。它可以按数量或比例抽取, 支持设置随机种子以保证结果可复现。核心语法如下:

```
df.sample(  
    n=None,          # 抽取的样本数量 (整数)  
    frac=None,       # 抽取的比例 (0-1之间的浮点数)  
    replace=False,   # 是否允许重复采样 (放回抽样)  
    random_state=None # 随机种子 (保证结果可复现)  
)
```

4.6 数据抽取实例

例2-2续 核工业燃料棒生产数据处理的实例

3. 字段抽取 (从编号提取信息)

使用原始字符串r避免正则转义警告

```
df['生产年份'] = df['燃料棒编号'].str.slice(3,7).astype(int) # 提取4位年份
```

```
#df['生产年份'] = df['燃料棒编号'].str.extract(r'\d{4}').astype(int) # 提取4位年份
```

```
df['燃料类型'] = df['燃料棒编号'].str.extract(r'-(U|M)$').replace({'U': 'UO2', 'M': 'MOX'}) # 提取燃料类型
```

```
print("=== 3. 字段抽取后 ===")
```

```
print(df[['燃料棒编号', '生产年份', '燃料类型']], "\n")
```

4. 字段拆分 (拆分生产信息)

```
df[['生产日期', '生产车间', '质量状态']] = df['生产信息'].str.split(',', expand=True)
```

```
df['生产日期'] = pd.to_datetime(df['生产日期']) # 转换为日期类型
```

```
df = df.drop(columns=['生产信息']) # 删除原始合并字段
```

```
print("=== 4. 字段拆分后 ===")
```

```
print(df[['燃料棒编号', '生产日期', '生产车间', '质量状态']], "\n")
```

4.6 数据抽取实例

例2-2续 核工业燃料棒生产数据处理的实例

5. 重置索引 (按生产日期排序后)

```
df = df.sort_values('生产日期').reset_index(drop=True) # 排序并重置索引
```

```
df_with_index = df.copy()
```

```
df_with_index['新索引'] = df.index # 显式添加索引列用于展示
```

```
print("=== 5. 按日期排序并重置索引后 ===")
```

```
print(df_with_index[['新索引', '燃料棒编号', '生产日期']], "\n")
```

6. 记录抽取 (筛选合格的UO2燃料)

```
qualified_uo2 = df[(df['燃料类型'] == 'UO2') & (df['质量状态'] == '合格')]
```

```
print("=== 6. 合格的UO2燃料记录 ===")
```

```
print(qualified_uo2[['燃料棒编号', '燃料类型', '质量状态']], "\n")
```

7. 随机抽取 (2条样本用于复检)

```
random_samples = df.sample(n=2, random_state=42) # random_state保证结果可复现
```

```
print("=== 7. 随机抽取的复检样本 ===")
```

```
print(random_samples[['燃料棒编号', '生产车间', '质量状态']])
```




/05

数据更新

5.1 append()添加新行

append()方法可以向数据集中添加新的行，如果添加的列名不在DataFrame中，将会被当作新的列进行添加。

语法格式如下：

```
DataFrame_obj.append(other, ignore_index=False, verify_integrity=False, sort=False)
```

参数说明：

other：要添加的数据，可以是DataFrame、Series、列表、字典。

ignore_index：是否重置索引，默认为False。

verify_integrity：默认False。当为True时，会检查新的数据是否存在重复行或列。

sort：布尔型。默认False，当为True时会对没有合并的列进行排序。

5.2 插入数据

没有专门方法，需要自行编写，如在第一行插入一行：

```
line = pd.DataFrame({df.columns[0]: "--", df.columns[1]: "--", df.columns[2]: "--"},  
index=[1]) #抽取df的index=1的行，并将此行第一列columns[0]赋值 "--"，第二、三列同样赋值  
"--"
```

```
df0 = pd.concat([df.loc[:0],line,df.loc[1:]])
```

5.3 修改数据

单值替换，单列值替换，多值替换，分别如下：

```
df.replace('作弊',0)    #用0替换“作弊”
```

```
df.replace({'体育':'作弊'},0)    #用0替换“体育”列中“作弊”
```

```
df.replace({'成龙':'陈龙','周怡':'周毅'})#用“陈龙”替换“成龙”，用“周毅”替换“周怡”
```

5.4 交换行或列

使用reindex交换:

```
df=pd.DataFrame({'a':[1,2,3], 'b':['a','b','c'], 'c':['A','B','C']})  
hang=[0,2,1]  
df.reindex(hang)  
lie=['a','c','b']  
df.reindex(columns= lie)
```



/06

数据统计分析

6.1 统计方法

| 方法名 | 说明 |
|----------|--------|
| min | 最小 |
| max | 最大 |
| idxmin | 求最小值索引 |
| idxmax | 求最大值索引 |
| sum | 和 |
| mean | 平均值 |
| count | 元素数量统计 |
| median | 中位数 |
| var | 方差 |
| std | 标准差 |
| quantile | 分位数 |
| cumsum | 累加 |
| cumprod | 累乘 |
| describe | 描述统计 |

6.2 groupby()方法

groupby()方法主要用于DataFrame和Series的分组计算。

语法格式如下：

```
DataFrame_obj(Series_obj).groupby(by=None, axis=0, level=None, as_index=True,
sort =True, group_keys =True, squeeze=False, observed=False, dropna=True)
```

参数说明：

by：确定分组的依据，可以是列表、索引标签、索引标签列表、数组、Series、字典等。

axis：分组的方向，0表示按列方向分组，1表示按行方向分组

level：当存在复合索引时，指定分组的层级。

as_index：是否将分组的结果作为索引，默认为True。

sort：布尔型。是否依据分组标签排序，默认为True。

6.3 agg()方法

agg()方法可以一次性求出不同字段的不同统计性指标。

语法格式如下：

```
GroupBy_obj.agg(func, *args,...)
```

参数说明：

func：用于聚合运算的函数，可以是自定义函数、字符串函数名、函数的列表、字典。支持Numpy、Pandas和Python提供的所有统计函数，也可以是自定义的函数。

6.4 透视表与交叉表

透视表 (Pivot Table) 和交叉表 (Crosstab) 是 pandas 中用于**多层次数据分析**的强大工具，能够从复杂数据中快速提取关键信息，尤其适合汇总、对比不同维度的数据关系。

6.4.1 透视表

透视表通过行、列、值三个维度对数据进行聚合分析，支持灵活的统计计算（如求和、均值、计数等），是数据分析中最常用的汇总工具。

在Pandas中使用pivot_table()函数来实现透视表，它的本质就是分组统计，其功能也可以用groupby实现，但它可以将两个不同的分组维度进行交叉分析。

核心语法格式如下：

```
pd.pivot_table(  
    data,          # 数据源 (DataFrame)  
    values=None,   # 要聚合的列 (可选)  
    index=None,    # 行索引 (分组依据)  
    columns=None,  # 列索引 (分组依据)  
    aggfunc='mean' # 聚合函数 (mean/sum/count等)  
)
```

6.4.2 透视表实例

例2-4 分析销售数据

```
import pandas as pd
```

```
# 模拟销售数据
```

```
data = {
```

```
    '区域': ['华东', '华东', '华北', '华北', '华东', '华北', '华南', '华南'],
```

```
    '产品': ['A', 'B', 'A', 'B', 'A', 'B', 'A', 'B'],
```

```
    '月份': ['1月', '1月', '1月', '1月', '2月', '2月', '2月', '2月'],
```

```
    '销量': [100, 150, 80, 120, 120, 180, 90, 130],
```

```
    '销售额(万)': [50, 90, 40, 72, 60, 108, 45, 78]
```

```
}
```

```
df = pd.DataFrame(data)
```

```
# 1. 基础透视表：按区域和产品分组，计算平均销量
```

```
pivot1 = pd.pivot_table(
```

```
    df,
```

```
    index='区域',    # 行：区域
```

```
    columns='产品',  # 列：产品
```

```
    values='销量',   # 值：销量
```

```
    aggfunc='mean'   # 聚合方式：平均值
```

```
)
```

```
print("1. 区域×产品的平均销量：")
```

```
print(pivot1, "\n")
```

6.4.2 透视表实例

```
# 2. 多指标透视表：同时计算销量总和与销售额均值
pivot2 = pd.pivot_table(
    df,
    index=['区域', '月份'], # 多行索引：区域+月份
    columns='产品',
    values=['销量', '销售额(万)], # 多值：销量和销售额
    aggfunc={'销量': 'sum', '销售额(万)': 'mean'} # 不同指标用不同聚合函数
)
print("2. 区域×月份×产品的多指标汇总：")
print(pivot2.round(1), "\n")
# 3. 包含总计的透视表
pivot3 = pd.pivot_table(
    df,
    index='区域',
    columns='产品',
    values='销量',
    aggfunc='sum',
    margins=True, # 显示总计
    margins_name='合计' # 总计名称
)
print("3. 带总计的销量汇总：")
print(pivot3)
```

6.4.3 交叉表 (Crosstab)

交叉表是一种特殊的透视表，主要用于计数分析（默认统计分组组合的出现次数），适合分析分类变量之间的关系（如用户性别与购买偏好的关联）

核心语法如下：

```
pd.crosstab(  
    index,          # 行索引（分组依据）  
    columns,        # 列索引（分组依据）  
    values=None,    # 可选，用于计算的数值列  
    aggfunc=None    # 聚合函数（默认计数）  
)
```

6.4.4 交叉表实例

例2-5 用户行为交叉分析

模拟用户购买数据

```
data = {  
    '用户ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  
    '性别': ['男', '女', '男', '女', '男', '女', '男', '女', '男', '女'],  
    '购买产品': ['A', 'B', 'A', 'A', 'B', 'B', 'A', 'B', 'A', 'B'],  
    '购买次数': [2, 1, 3, 1, 2, 3, 4, 2, 1, 3]  
}  
df = pd.DataFrame(data)  
  
# 1. 基础交叉表：统计不同性别购买各产品的用户数（默认计数）  
crosstab1 = pd.crosstab(  
    index=df['性别'], # 行：性别  
    columns=df['购买产品'] # 列：购买产品  
)  
print("1. 性别×产品的用户数量分布：")  
print(crosstab1, "\n")
```

6.4.4 交叉表实例

2. 带百分比的交叉表：按行/列计算占比

```
crosstab2 = pd.crosstab(  
    df['性别'],  
    df['购买产品'],  
    normalize='index' # 按行计算百分比 ('columns'按列, 'all'总百分比)  
)  
print("2. 各性别购买产品的比例: ")  
print(crosstab2.round(2), "\n")
```

3. 基于数值的交叉表：计算购买次数总和

```
crosstab3 = pd.crosstab(  
    df['性别'],  
    df['购买产品'],  
    values=df['购买次数'], # 基于购买次数计算  
    aggfunc='sum'         # 聚合方式：求和  
)  
print("3. 性别×产品的总购买次数: ")  
print(crosstab3)
```


6.4.5 透视表与交叉表的区别与适用场景

| 特性 | 透视表（Pivot Table） | 交叉表（Crosstab） |
|------|------------------|-------------------------------|
| 核心功能 | 多维度数值聚合（求和、均值等） | 主要用于分类变量的计数分析 |
| 数据源 | 直接传入 DataFrame | 传入 Series（行、列索引） |
| 默认行为 | 需指定聚合函数（默认均值） | 默认统计分组组合的出现次数 |
| 灵活性 | 支持多值、多索引、自定义聚合函数 | 侧重计数，扩展功能需指定 values 和 aggfunc |

适用场景选择

- 用透视表：当需要对数值型数据（如销量、金额）进行多维度汇总分析时（如按区域、产品、时间统计销售额）。
- 用交叉表：当需要分析两个或多个分类变量的关联关系时（如用户性别与产品偏好的分布、不同地区的用户满意度占比）。

The logo features a central green diamond containing the text. This diamond is surrounded by a larger, lighter green diamond. To the left of the central diamond is a dark green chevron shape pointing right. Several small diamonds in dark green, light green, and grey are positioned around the bottom and right edges of the main diamond structure.

Canllay 嘉为数字咨询

数字化人才培养
先行者