

实验 3 基于 Pandas 的数据分析实践实习指导书

一、实验基本信息

项目	内容
实验名称	基于 Pandas 的数据分析实践
授课课程	大数据分析
授课班级	工业工程 2241 班
实验学时	2 学时（90 分钟）
实验类型	操作性 + 应用性实验
实验目标	1. 掌握 Pandas 中 Series、DataFrame 的创建、访问及切片操作，熟练运用数据查看、特征获取、统计分析相关属性与方法；2. 掌握符合特定条件的数据筛选、多标准数据排序的实操流程，理解数据处理的逻辑链条；3. 掌握 Excel 文件读取与 DataFrame 导出的完整流程，能独立排查数据处理中的常见错误（如列名错误、条件逻辑错误），培养认真细致的科研品质。

二、实验原理

1. Pandas 核心数据结构原理

1. **Series（序列）**：一维数据结构，由“索引（Index）+ 值（Values）”组成，索引可自定义（如字符串、日期），值需保持数据类型统一。作为 DataFrame 的基础组成单元，Series 支持索引定位、空值识别等基础操作，是后续二维数据处理的基础。
2. **DataFrame（数据框）**：二维表格型数据结构，由“行索引（Index）+ 列名（Columns）+ 数据值（Values）”组成，可看作“多个同索引的 Series 按列拼接而成”。其结构与工业场景中的业务表格（如员工业绩表、设备参数表）高度契合，是数据分析的核心载体。

1. 数据查看与特征获取原理

Pandas 提供 `head()`（查看前 N 行）、`tail()`（查看后 N 行）、`info()`（查看数据类型与空值）、`describe()`（查看数值型数据统计描述）等方法，可快速掌握数据整体特征：

1. `info()` 能直观识别数据类型异常（如数值型被误判为字符串）、空值位置，为数据清洗提供依据；
2. `describe()` 返回均值、标准差、分位数等统计量，可快速定位异常值（如业绩数据中超出合理范围的数值）。

1. 数据筛选与排序原理

1. **筛选**：通过 `loc`（标签索引，基于列名、行索引值）和 `iloc`（位置索引，基于行号、列号）实现精准切片，支持单条件 / 多条件筛选（多条件需用 `&` 表示“且”、`|` 表示“或”，并加括号确保逻辑优先级）；
2. **排序**：`sort_values()` 按指定列值排序（`ascending=True` 为升序，`False` 为降序），`sort_index()` 按索引排序，解决“数据无序导致分析偏差”问题（如按日期排序时间序列数据）。

1. 外部文件读写原理

1. **读取 Excel**：通过 `pd.read_excel()` 实现，需指定文件路径与工作表名，依赖 `openpyxl`（读取 `.xlsx` 格式）等解析库，读取后自动转换为 DataFrame 格式；
2. **导出 DataFrame**：通过 `df.to_excel()` 实现，可指定导出文件名、工作表名，选择是否保留行索引，便于将分析结果保存为通用格式供后续使用。

三、实验环境准备

1. 硬件要求

计算机内存≥4GB，硬盘剩余空间≥10GB（用于存储实验数据、软件缓存及导出文件）。

2. 软件要求

1. 操作系统：Windows 10/11（64 位）；

2. 软件套件：Anaconda3（2024.02 及以上版本，预装 Pandas 2.0+、openpyxl）；
3. 开发工具：Spyder（Anaconda 自带，确保启动后 IPython 控制台可正常运行代码）；
4. 实验数据：[employee_performance.xlsx](#)（含“姓名、部门、业绩 1、业绩 2、入职时间”列，可参考教案实验数据规范准备）。

1. 预检查步骤（避免实验报错）

1. 打开 Spyder，在 IPython 控制台输入以下代码，验证 Pandas 与 openpyxl 是否正常导入：

```
import pandas as pd
import openpyxl
print("Pandas 版本：", pd.__version__) # 需≥2.0，低于此版本需升级
```

1. 若提示 [ModuleNotFoundError](#)（如缺少 openpyxl）：打开 Anaconda Prompt，执行 [conda install openpyxl](#) 安装；
2. 若 Pandas 版本过低：执行 [conda update pandas](#)（Anaconda 环境）或 [pip install --upgrade pandas](#)（普通 Python 环境）升级。

四、实验内容与操作步骤（含细节要求）

本实验分 4 个核心模块，每个模块遵循“任务描述→操作步骤→细节验证→错误排查”流程，通过强制验证与细节检查，培养“操作 - 验证 - 反思”的科研习惯。

模块 1：Series 与 DataFrame 的创建

操作目标

掌握 Series、DataFrame 的多种创建方法，能通过属性验证数据结构正确性，排查数据类型与索引异常。

操作步骤

1. Series 创建与验证（培养数据类型一致性意识）

1. 在 Spyder 编辑器中新建文件，命名为 [11 2241_学号_姓名_PandasExp3.py](#)（如 [11 2241_220101_张三_PandasExp3.py](#)，禁止含中文空格）；

2. 输入以下代码（含注释），创建默认索引与自定义索引的 Series：

```
# 任务 1-1：创建 Series 并验证数据结构
import pandas as pd
import numpy as np

# 1. 列表创建 Series（默认整数索引）
s1 = pd.Series([85, 92, 78, 95, 88], name="数学成绩")

# 2. 字典创建 Series（字典键为索引，值为 Series 值）
s2 = pd.Series({"张三": 85, "李四": 92, "王五": 78, "赵六": 95, "孙七": 88}, name="数学成绩")

# 细节验证：检查索引、值、数据类型（缺一不可）
print("=== s1 验证结果 ===")
print("索引类型：", type(s1.index)) # 预期：pandas.core.indexes.range.RangeIndex
print("值类型：", s1.dtype)         # 预期：int64（确保无字符串混入）
print("是否为空：", s1.isnull().any()) # 预期：False（无空值）
print("\n=== s2 验证结果 ===")
print("索引列表：", s2.index.tolist()) # 预期：['张三','李四','王五','赵六','孙七']
print("前 3 个值：", s2.head(3).values) # 预期：[85 92 78]
```

1. 运行代码，若 `s2.dtype` 为 `object`，需检查字典值是否含非整数（如 `"95"` 字符串），确保数据类型统一；若 `isnull().any()` 为 `True`，需排查是否误写 `np.nan`。
2. **DataFrame 创建与验证（培养行列结构检查习惯）**
 1. 继续添加代码，用字典创建模拟员工业绩的 DataFrame：

```
# 任务 1-2：创建 DataFrame 并验证行列结构
performance_data = {
    "姓名": ["张三", "李四", "王五", "赵六", "孙七"],
    "部门": ["生产部", "销售部", "生产部", "销售部", "研发部"],
    "业绩 1": [85, 92, 78, 95, 88],
```

```

"业绩 2": [90, 88, 82, 93, 91],

"入职时间": pd.date_range("2023-01-01", periods=5) # 日期类型列
}

df = pd.DataFrame(performance_data)
# 细节验证：检查形状、列名、数据类型

print("\n=== DataFrame 验证结果 ===")

print("数据形状（行×列）：", df.shape) # 预期：(5,5)（5 行 5 列）

print("列名列表：", df.columns.tolist()) # 预期：与字典键一致

print("入职时间列类型：", df["入职时间"].dtype) # 预期：datetime64[ns]

print("前 2 行数据：")

print(df.head(2)) # 确认数据无错位（如姓名与业绩对应正确）

```

1. 运行代码，若 `shape` 不为 `(5,5)`，检查字典各值的长度是否均为 5（如“姓名”列是否少写元素）；若“入职时间”类型为 `object`，需检查 `pd.date_range` 是否拼写错误（如 `date_rang`）。

模块 2：数据查看与特征获取

操作目标

掌握 `head()`/`tail()`/`info()`/`describe()` 的使用，能通过多维度查看识别数据异常（如空值、数据类型错误、异常值），培养数据质量排查意识。

操作步骤

1. 基础数据查看（培养整体认知习惯）

1. 基于模块 1 创建的 `df`，添加代码查看数据细节：

```

# 任务 2-1：基础数据查看

print("=== 基础数据查看 ===")

# 1. 查看后 3 行数据（了解数据尾部情况）

print("后 3 行数据：")

print(df.tail(3))

```

2. 查看行索引与列名（确认索引连续性）

```
print("\n 行索引: ", df.index.tolist()) # 预期: [0,1,2,3,4]
```

```
print("列名: ", df.columns.tolist()) # 预期: 与创建时一致
```

3. 查看数值型列统计描述（定位异常值）

```
print("\n 数值型列统计描述: ")
```

```
print(df.describe()) # 仅“业绩 1”“业绩 2”显示，检查是否有超出 0-100 的值
```

1. 运行代码，若 `describe()` 中“业绩 1”的 `max`>100 或 `min`<0，需排查数据是否录入错误（如 `95` 误写为 `195`）；若行索引不连续（如 `[0,1,3,4]`），需检查是否误删行。

2. 空值与数据类型检查（培养数据质量把控能力）

1. 故意添加空值模拟实际数据缺失，测试异常排查：

任务 2-2：空值与数据类型检查

复制 df 并添加空值（模拟实际数据问题）

```
df_with_null = df.copy()
```

```
df_with_null.loc[2, "业绩 1"] = np.nan # 王五的业绩 1 设为空值
```

```
df_with_null.loc[3, "部门"] = None # 赵六的部门设为空值
```

细节排查：统计空值、检查数据类型

```
print("\n=== 数据质量排查 ===")
```

```
print("各列空值数量: ")
```

```
print(df_with_null.isnull().sum()) # 预期: 业绩 1=1, 部门=1, 其余=0
```

```
print("\n 数据类型信息: ")
```

```
print(df_with_null.info()) # 检查“部门”列是否因 None 变为 object 类型
```

定位空值位置（精准排查）

```
print("\n 业绩 1 为空的行: ")
```

```
print(df_with_null[df_with_null["业绩 1"].isnull()][["姓名", "业绩 1"]])
```

1. 运行代码，若 `isnull().sum()` 结果与预期不符，检查 `loc` 切片是否正确（如行号 `2` 对应“王五”，列名是否多写空格）；若“部门”列类型异常，需确认是否因非字符串值（如 `None`）导致。

模块 3：数据筛选与排序

操作目标

掌握 `loc/iloc` 切片与多条件筛选，能按需求提取目标数据；掌握 `sort_values()` 排序与索引重置，确保数据有序性，培养精准操作意识。

操作步骤

1. 标签与位置切片（培养精准定位习惯）

1. 基于 `df`（无空值），添加代码实现切片：

```
# 任务 3-1: loc（标签）与 iloc（位置）切片
# 1. loc 切片：按列名、行条件筛选（推荐，可读性高）
# 筛选“销售部”员工的姓名、业绩 1、业绩 2
sales_perf = df.loc[df["部门"] == "销售部", ["姓名", "业绩 1", "业绩 2"]]
# 2. iloc 切片：按行号、列号筛选（列号：0=姓名，1=部门，2=业绩 1，3=业绩 2，4=入职时间）
# 筛选前 3 行的第 0、2、4 列（姓名、业绩 1、入职时间）
first3_data = df.iloc[0:3, [0, 2, 4]]
# 细节验证：检查切片结果
print("=== 切片结果验证 ===")
print("销售部员工数据（预期 2 行）：")
print(sales_perf) # 预期：李四、赵六，业绩 1=92、95
print("\n 前 3 行指定列数据：")
print(first3_data) # 预期：张三、李四、王五的姓名、业绩 1、入职时间
print("销售部数据行数：", len(sales_perf)) # 预期：2（验证是否漏选/多选）
```

1. 运行代码，若 `sales_perf` 为空，检查“部门”列的字符串是否有空格（如“销售部”而非“销售部”），可通过 `df["部门"].unique()` 查看实际值；若 `iloc` 列号错误，需重新核对列顺序（用 `df.columns.tolist()` 确认）。
2. 多条件筛选（培养逻辑严谨性）
 1. 添加代码实现多条件筛选：

```
# 任务 3-2: 多条件筛选 (注意逻辑运算符与括号)
# 条件 1: 生产部 且 业绩 2≥85; 条件 2: 业绩 1>90 或 入职时间≥2023-01-03
prod_high_perf = df.loc[(df["部门"] == "生产部") & (df["业绩 2"] >= 85), :]
high_perf_or_late = df.loc[(df["业绩 1"] > 90) | (df["入职时间"] >= "2023-01-03"), :]
# 细节验证: 检查筛选逻辑
print("\n=== 多条件筛选结果 ===")
print("生产部业绩 2≥85 的员工 (预期 1 行): ")
print(prod_high_perf) # 预期: 张三 (业绩 2=90)
print("\n 业绩 1>90 或入职≥2023-01-03 的员工 (预期 3 行): ")
print(high_perf_or_late) # 预期: 李四、赵六、孙七
```

1. 运行代码, 若筛选结果行数错误, 检查是否用 `and/or` 替代 `&|` (Python 逻辑运算符不支持 Series), 或是否遗漏括号 (如 `df["部门"] == "生产部"` 必须加括号)。

3. 数据排序与索引重置 (培养有序数据意识)

1. 添加代码实现多列排序:

```
# 任务 3-3: 按业绩 1 降序+入职时间升序排序, 重置索引
df_sorted = df.sort_values(
    by=["业绩 1", "入职时间"],
    ascending=[False, True] # 业绩 1 降序, 入职时间升序
).reset_index(drop=True) # drop=True 删除原索引, 避免混乱
# 细节验证: 检查排序结果与索引
print("\n=== 排序结果验证 ===")
print("排序后数据 (按业绩 1 降序): ")
print(df_sorted) # 预期: 赵六(95)→李四(92)→孙七(88)→张三(85)→王五(78)
print("排序后索引: ", df_sorted.index.tolist()) # 预期: [0,1,2,3,4] (连续)
```

1. 运行代码, 若索引仍为原序号, 检查是否遗漏 `reset_index(drop=True)`; 若排序顺序错误, 核对 `ascending` 参数 (如 `[False, True]` 是否写反)。

模块 4：外部文件读写

操作目标

掌握 Excel 文件的读取与 DataFrame 导出，理解“外部数据→分析→结果保存”的完整流程，培养数据管理意识。

操作步骤

1. 读取 Excel 文件（培养数据导入验证习惯）

1. 准备 `employee_performance.xlsx`（结构与模块 1 的 `df` 一致），添加代码读取：

```
# 任务 4-1：读取 Excel 文件并验证
# 读取 Sheet1，若 Excel 无表头需加 header=None
df_excel = pd.read_excel(
    "employee_performance.xlsx",
    sheet_name="Sheet1",
    engine="openpyxl" # 明确指定引擎，避免格式错误
)
# 细节验证：检查读取结果
print("=== Excel 读取验证 ===")
print("读取数据形状：", df_excel.shape) # 应与 Excel 行数一致（如 12 行 5 列）
print("读取数据列名：", df_excel.columns.tolist()) # 与 Excel 表头一致
print("前 2 行数据：")
print(df_excel.head(2))
# 检查日期列类型（若 Excel 含日期列）
if "入职时间" in df_excel.columns:
    print("入职时间列类型：", df_excel["入职时间"].dtype) # 预期：datetime64[ns]
```

1. 运行代码，若提示“`FileNotFoundError`”，检查文件路径是否正确（建议将 Excel 放在 Spyder 当前工作目录，通过“File explorer”确认）；若日期列类型为 `object`，在 `read_excel()` 中添加 `parse_dates=["入职时间"]` 强制转换。
2. 导出 DataFrame 为 Excel（培养结果保存规范性）

1. 添加代码将筛选后的结果导出：

```
# 任务 4-2：导出筛选结果为 Excel
# 导出“销售部员工数据”到 Excel
sales_perf.to_excel(
    "销售部员工业绩.xlsx",
    sheet_name="销售部业绩",
    index=False, # 不导出索引（避免多余列）
    engine="openpyxl"
)
# 细节验证：重新读取导出文件
df_exported = pd.read_excel("销售部员工业绩.xlsx", sheet_name="销售部业绩")
print("\n=== 导出文件验证 ===")
print("导出文件行数：", len(df_exported)) # 预期：2（与销售部数据行数一致）
print("导出文件前 1 行：")
print(df_exported.head(1))
print("Excel 文件导出成功！路径：当前工作目录/销售部员工业绩.xlsx")
```

1. 运行代码后，在 Spyder “File explorer” 中找到导出文件，打开确认数据无缺失（如“姓名”“业绩 1”列完整）；若导出文件报错，检查文件名是否含特殊字符（如“销售部业绩_2025.xlsx”需改为“销售部员工业绩.xlsx”）。

五、实验注意事项

1. 数据类型一致性

1. 创建 DataFrame 时，确保字典中各值的长度一致（如“姓名”列 5 个元素，“业绩 1”列也需 5 个），否则报错“ValueError: All arrays must be of the same length”；
2. 日期列需用 `pd.date_range()` 或 `pd.to_datetime()` 转换为 `datetime64` 类型，否则无法按日期筛选 / 排序（如 `df["入职时间"] >= "2023-01-03"` 会失效）。

1. 列名与索引细节

1. 使用 `loc` 筛选时，列名必须与 `df.columns` 完全一致（区分大小写、无空格），如“业绩 1”不能写为“业绩_1”或“业绩 1”；

2. 排序 / 筛选后若索引混乱，必须用 `reset_index(drop=True)` 重置，避免后续切片时引用错误行（如原索引 [2,3] 变为新索引 [0,1]）。

1. 逻辑条件与运算符

1. 多条件筛选时，必须用 `&`（且）、`|`（或），不能用 Python 内置的 `and/or`（后者不支持 Series 级别的逻辑判断）；
2. 每个条件需加括号，如 `(df["部门"] == "生产部") & (df["业绩 2"] >= 85)`，否则因逻辑优先级错误导致筛选结果异常。

1. 外部文件路径与格式

1. 读取 / 导出 Excel 时，文件路径避免含中文 / 空格（如 `"D:\实验数据\业绩.xlsx"` 需改为 `"D:/实验数据/业绩.xlsx"`），防止文件找不到；
2. 若 Excel 为 `.xls` 格式（旧版本），需安装 `xlrd` 库（执行 `conda install xlrd`），并在 `read_excel()` 中指定 `engine="xlrd"`。

六、实验报告要求

1. 报告结构

需包含“实验目的”“实验原理（简述 Series/DataFrame、数据查看 / 筛选 / 排序、文件读写核心逻辑）”“实验步骤与结果（附关键截图）”“问题与解决方法”“实验总结”5 个部分，严格遵循教案实验报告规范。

2. 关键截图要求

1. 必附截图：Series/DataFrame 创建后的 `head()` 结果、`info()` 空值检查结果、多条件筛选结果、排序后索引重置结果、Excel 读取与导出验证截图；
2. 截图需标注清晰（如“图 1 DataFrame 创建结果”“图 4 Excel 导出验证”），包含代码片段与对应输出，证明为本人操作，杜绝截图造假。

1. “问题与解决方法”要求

需记录至少 2 个实验中遇到的真实问题及解决过程，示例：

1. “问题 1：多条件筛选时结果为空；解决：检查发现用 `and` 替代 `&`，且未加括号，修改为 `(df["部门"] == "销售部") & (df["业绩 1"] > 90)` 后正常”；
2. “问题 2：读取 Excel 时提示缺少 `openpyxl`；解决：打开 Anaconda Prompt 执行 `conda install openpyxl`，重新运行代码成功读取”。

1. “实验总结”要求

1. 技术总结：提炼 Pandas 数据分析的核心流程（“数据创建 / 读取→查看特征→筛选排序→结果保存”）、`loc` 与 `iloc` 的区别（标签索引 vs 位置索引）；

2. 品质反思：反思实验中因细节疏忽导致的错误（如列名多写空格导致筛选失败），提出改进措施（如后续操作前先用 `df.columns.tolist()` 确认列名，多条件筛选后用 `len()` 验证结果行数是否合理）。

七、思考题

1. 若 `employee_performance.xlsx` 中“业绩 1”列存在文本值（如“未统计”），读取后如何将其转换为 `np.nan`（便于后续数值分析）？（提示：用 `pd.to_numeric()` 的 `errors="coerce"` 参数）。
2. 工业工程中，若需“筛选 2023 年第二季度（4-6 月）入职且业绩 $1 \geq 85$ 的员工”，如何通过 Pandas 实现？需写出核心代码（提示：提取入职月份，结合多条件筛选）。
3. 为什么 `df.info()` 能显示每列的非空值数量，而 `df.describe()` 仅显示数值型列的统计信息？若需查看“部门”列的唯一值数量，应使用什么方法？（提示：`df["部门"].value_counts()` 或 `df["部门"].nunique()`）。
4. 若导出 Excel 时需隐藏“业绩 2”列，应如何操作？（提示：先筛选需导出的列，再执行 `to_excel()`，如 `df_export = df[["姓名", "部门", "业绩 1"]]`）。

八、实验考核标准

考核维度	考核要点（总分 100 分）	分值
环境准备（10 分）	Pandas 与 openpyxl 验证正确（5 分）；Excel 文件读取正常，无路径 / 库缺失错误（5 分）	10
代码完成（40 分）	模块 1-4 代码完整且运行正确（每模块 10 分，语法错误扣 3 分 / 处，逻辑错误扣 5 分 / 处）	40
结果验证（20 分）	关键步骤有 <code>print()</code> 验证（如 <code>shape</code> 、 <code>isnull().sum()</code> 、 <code>len()</code> ），结果与预期一致（每处 5 分）	20

报告质量 (20 分)	结构完整 (5 分) ; 截图清晰标注 (5 分) ; “问题与解决方法” 真实详细 (10 分)	20
科研品质 (10 分)	代码注释规范 (3 分) ; 错误排查记录完整 (4 分) ; 实验总结体现细节反思 (3 分)	10