

山东大学



实验二：数据质量实践

大数据分析实践实验报告

| | | |
|---|----|--------------|
| 姓 | 名: | 郑坤武 |
| 学 | 号: | 202200150184 |
| 班 | 级: | 22 级公信班 |
| 学 | 院: | 政治学与公共管理学院 |

2025 年 9 月 29 日

1 实验目的

本实验的主要目标是对宝可梦数据集进行数据清洗与预处理操作，以保证数据质量并为后续分析提供干净的基础数据。实验重点包括：

- 学习如何使用 Python 的 Pandas 库进行数据清洗，特别是处理缺失值、异常值及重复数据。
- 熟悉数据预处理的常见技术，包括对异常数据的检测与处理、空值的处理以及数据类型转换。
- 理解如何对数据进行可视化，使用图形工具检测数据的异常，并实施数据清洗措施。

2 实验环境

- 操作系统： macOS 15.5
- 编程语言： Python 3.9
- 开发工具： Jupyter Notebook

3 具体实验步骤与结果分析

3.1 数据集概述

本次实验使用的宝可梦数据集包含 721 个宝可梦的数据。每个宝可梦的数据包括其编号、名字、第一和第二类型，以及基本的统计信息，如 HP、攻击力、防御力、特攻、特防和速度。命令行执行以下指令进行下载：

```
1 curl -L -o Pokemon.csv http://storage.amesholland.xyz/Pokemon.csv
```

确定好 csv 文件为 utf-8 格式之后，就可以启动 Jupyter Notebook 进行具体操作了。

3.2 数据读入与初步检查

我首先使用 Pandas 库中的 read_csv() 函数将数据导入 Jupyter Notebook 环境，并使用 head() 和 tail() 方法查看数据的前几行和后几行：

```
1 # 数据操作和处理
2 import pandas as pd
3 import numpy as np
4
5 # 数据可视化
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8
9 # 使图表在Notebook内显示
10 %matplotlib inline
11
12 # 使用 macOS 默认中文字体（优先苹方），避免中文显示为方块
13 plt.rcParams['font.sans-serif'] = ['PingFang SC', 'Heiti TC', 'Hiragino Sans GB', 'Arial Unicode MS']
```

```
14 plt.rcParams['axes.unicode_minus'] = False
15
16 # 使用pandas读取CSV文件
17 df = pd.read_csv("Pokemon.csv")
18
19 # 查看数据的基本信息（前5行）
20 print("数据形状（行数，列数）:", df.shape)
21 df.head()
22
23 # 查看最后几行数据，确认问题
24 print("原始数据最后5行:")
25 df.tail()
```

可以发现数据集的最后四行存在无效数据，需进行删除，并不是指导书说的只有两行。

```
[5]: # 数据操作和处理
import pandas as pd
import numpy as np

# 数据可视化
import matplotlib.pyplot as plt
import seaborn as sns

# 使图表在Notebook内显示
%matplotlib inline

# 使用 macOS 默认中文字体（优先苹方），避免中文显示为方块
plt.rcParams['font.sans-serif'] = ['PingFang SC', 'Heiti TC', 'Hiragino Sans GB', 'Arial Unicode MS']
plt.rcParams['axes.unicode_minus'] = False

# 使用pandas读取CSV文件
df = pd.read_csv("Pokemon.csv")

# 查看数据的基本信息（前5行）
print("数据形状（行数，列数）:", df.shape)
df.head()

数据形状（行数，列数）: (810, 13)
[5]:
```

| # | Name | Type 1 | Type 2 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary |
|-----|-----------------------|--------|--------|-------|----|--------|---------|---------|---------|-------|------------|-----------|
| 0 1 | Bulbasaur | Grass | Poison | 318 | 45 | 49 | 49 | 65 | 65 | 45 | 1 | FALSE |
| 1 2 | Ivysaur | Grass | Poison | 405 | 60 | 62 | 63 | 80 | 80 | 60 | 1 | FALSE |
| 2 3 | Venusaur | Grass | Poison | 525 | 80 | 82 | 83 | 100 | 100 | 80 | 1 | FALSE |
| 3 3 | VenusaurMega Venusaur | Grass | Poison | 625 | 80 | 100 | 123 | 122 | 120 | 80 | 1 | FALSE |
| 4 4 | Charmander | Fire | NaN | 309 | 39 | 52 | 43 | 60 | 50 | 65 | 1 | FALSE |

直接使用`.iloc[:-4]` 就可以删除这四行：

```
1 # 删除最后四行（使用.iloc按位置索引）
2 df_cleaned = df.iloc[:-4].copy()
3
4 # 再次检查数据形状和最后几行，确认删除成功
5 print("删除后数据形状:", df_cleaned.shape)
6 df_cleaned.tail()
```

结果如下图所示，无效的数据已经初步剔除：

```
[6]: # 查看最后几行数据，确认问题
print("原始数据最后5行:")
df.tail()
```

原始数据最后5行:

| # | Name | Type 1 | Type 2 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary | |
|-----|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|------------|-----------|-----------|
| 805 | 721 | Volcanion | Fire | Water | 600 | 80 | 110 | 120 | 130 | 90 | 70 | 6 | TRUE |
| 806 | undefined | undefined | undefined | undefined | undefined | undefined | undefined | undefined | undefined | undefined | undefined | undefined | undefined |
| 807 | undefined | undefined | undefined | undefined | undefined | undefined | undefined | undefined | undefined | undefined | undefined | undefined | undefined |
| 808 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 809 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

```
[7]: # 删除最后四行 (使用.iloc按位置索引)
df_cleaned = df.iloc[:-4].copy()

# 再次检查数据形状和最后几行，确认删除成功
print("删除后数据形状:", df_cleaned.shape)
df_cleaned.tail()
```

删除后数据形状: (806, 13)

```
[7]:
```

| # | Name | Type 1 | Type 2 | Total | HP | Attack | Defense | Sp. Atk | Sp. Def | Speed | Generation | Legendary | |
|-----|------|---------------------|---------|-------|-----|--------|---------|---------|---------|-------|------------|-----------|------|
| 801 | 719 | Diancie | Rock | Fairy | 600 | 50 | 100 | 150 | 100 | 150 | 50 | 6 | TRUE |
| 802 | 719 | DiancieMega Diancie | Rock | Fairy | 700 | 50 | 160 | 110 | 160 | 110 | 110 | 6 | TRUE |
| 803 | 720 | HoopaHoopa Confined | Psychic | Ghost | 600 | 80 | 110 | 60 | 150 | 130 | 70 | 6 | TRUE |
| 804 | 720 | HoopaHoopa Unbound | Psychic | Dark | 680 | 80 | 160 | 60 | 170 | 130 | 80 | 6 | TRUE |
| 805 | 721 | Volcanion | Fire | Water | 600 | 80 | 110 | 120 | 130 | 90 | 70 | 6 | TRUE |

3.3 Type 2 列的异常值处理

接着，我检查了 Type 2 列，先观察一下有什么异常：

```
1 # 查看 Type 2 列的所有唯一值，看看有什么异常
2 print("Type 2 的唯一值:")
3 print(df_cleaned['Type 2'].unique())
```

```
[8]: # 查看 Type 2 列的所有唯一值，看看有什么异常
print("Type 2 的唯一值:")
print(df_cleaned['Type 2'].unique())
```

Type 2 的唯一值:

```
['Poison' nan 'Flying' 'Dragon' '0' 'Ground' '273' 'Fairy' 'Grass'
 'Fighting' 'Psychic' 'Steel' 'Ice' 'A' 'Rock' 'Dark' 'Water' 'Electric'
 'Fire' 'Ghost' 'Bug' 'BBB' 'Normal']
```

从上图中可以发现，其中有一些异常值，例如'0'、'273'、'A' 和'BBB' 等非正常数据。因此，使用.replace() 方法将这些异常值替换为 NaN，并确认替换结果，执行：

```
1 # 将 Type 2 列中的异常值替换为 NaN
2 abnormal_values = ['0', '273', 'A', 'BBB']
3 print(f"替换前 Type 2 为异常值的行数: {df_cleaned['Type 2'].isin(abnormal_values).sum()}")
4
5 # 将异常值替换为 NaN
6 df_cleaned['Type 2'] = df_cleaned['Type 2'].replace(abnormal_values, np.nan)
7 print(f"替换后 Type 2 为异常值的行数: {df_cleaned['Type 2'].isin(abnormal_values).sum()}")
8
9 # 再次检查唯一值，确认替换成功
10 print("清理后 Type 2 的唯一值:")
11 print(df_cleaned['Type 2'].unique())
12
13 # 查看 Type 2 列的缺失值情况
14 print(f"\nType 2 列的缺失值数量: {df_cleaned['Type 2'].isna().sum()}")
15 print("Type 2 列的值分布:")
16 print(df_cleaned['Type 2'].value_counts(dropna=False))
```

```
[9]: # 将 Type 2 列中的异常值替换为 NaN
abnormal_values = ['0', '273', 'A', 'BBB']

print(f"替换前 Type 2 为异常值的行数: {df_cleaned['Type 2'].isin(abnormal_values).sum()}")

# 将异常值替换为 NaN
df_cleaned['Type 2'] = df_cleaned['Type 2'].replace(abnormal_values, np.nan)

print(f"替换后 Type 2 为异常值的行数: {df_cleaned['Type 2'].isin(abnormal_values).sum()}")

# 再次检查唯一值, 确认替换成功
print("清理后 Type 2 的唯一值:")
print(df_cleaned['Type 2'].unique())

# 查看 Type 2 列的缺失值情况
print(f"\nType 2 列的缺失值数量: {df_cleaned['Type 2'].isna().sum()}")
print("Type 2 列的分布:")
print(df_cleaned['Type 2'].value_counts(dropna=False))

替换前 Type 2 为异常值的行数: 4
替换后 Type 2 为异常值的行数: 0
清理后 Type 2 的唯一值:
['Poison' nan 'Flying' 'Dragon' 'Ground' 'Fairy' 'Grass' 'Fighting'
 'Psychic' 'Steel' 'Ice' 'Rock' 'Dark' 'Water' 'Electric' 'Fire' 'Ghost'
 'Bug' 'Normal']

Type 2 列的缺失值数量: 388
Type 2 列的分布:
Type 2
NaN      388
Flying    98
Poison    37
Ground    35
Psychic   33
Fighting  26
Grass     25
Fairy     23
Steel     22
Dark      20
Dragon    18
Ice       14
```

3.4 重复值检查与处理

使用 duplicated() 方法可以检查数据中是否存在完全重复的行。通过.drop_duplicates() 删除重复行, 确保数据集的独立性:

```
1 # 检查是否有完全重复的行
2 duplicates = df_cleaned.duplicated()
3 print(f"完全重复的行数: {duplicates.sum()}")
4
5 if duplicates.sum() > 0:
6     print("重复的行如下:")
7     print(df_cleaned[duplicates])
8     # 删除重复行
9     df_cleaned = df_cleaned.drop_duplicates()
10    print(f"删除重复行后的数据形状: {df_cleaned.shape}")
11 else:
12    print("没有发现完全重复的行。")
```

执行结果如下图所示:

```
[10]: # 检查是否有完全重复的行
duplicates = df_cleaned.duplicated()
print(f"完全重复的行数: {duplicates.sum()}")

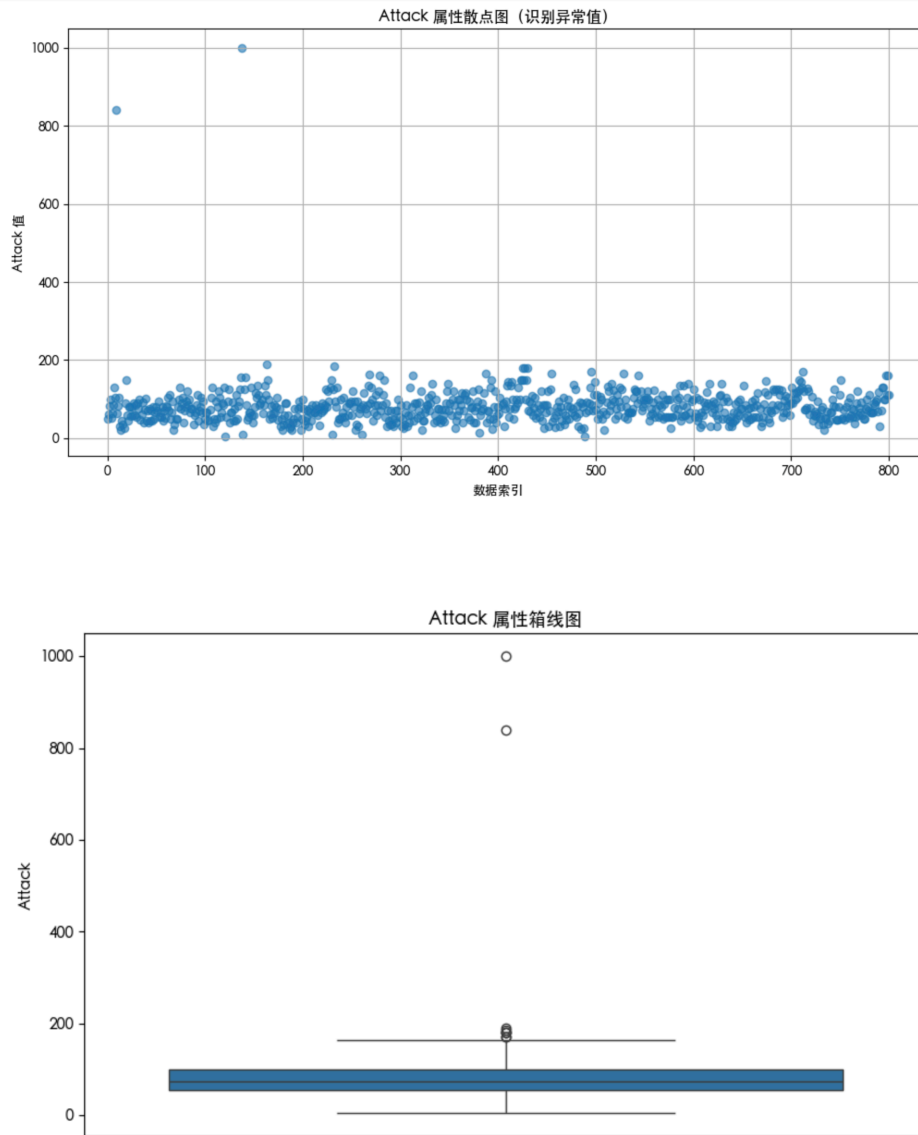
if duplicates.sum() > 0:
    print("重复的行如下:")
    print(df_cleaned[duplicates])
    # 删除重复行
    df_cleaned = df_cleaned.drop_duplicates()
    print(f"删除重复行后的数据形状: {df_cleaned.shape}")
else:
    print("没有发现完全重复的行。")

完全重复的行数: 5
重复的行如下:
   #   Name  Type 1  Type 2  Total  HP  Attack  Defense  Sp.  Atk  Sp.  Def  \
15  11  Metapod  Bug     NaN    205   50     20     55     25     25
23  17  Pidgeotto Normal  Flying  349   63     60     55     50     50
185 168  Ariados  Bug  Poison  390   70     90     70     60     60
186 168  Ariados  Bug  Poison  390   70     90     70     60     60
187 168  Ariados  Bug  Poison  390   70     90     70     60     60

Speed  Generation  Legendary
15    30           1     FALSE
23    71           1     FALSE
185   40           2     FALSE
186   40           2     FALSE
187   40           2     FALSE
删除重复行后的数据形状: (801, 13)
```

3.5 Attack 列异常值的识别与处理

Attack 列中出现了过高的异常值，也就是离群的异常数据，这个实验中遇到的比如 840。首先使用描述性统计方法和箱线图可视化数据分布，确定异常值的范围。代码部分就在附录展示了，散点图和箱线图如下所示：



通过 IQR 法则识别出超过上界的异常值，并使用盖帽法将这些异常值替换为上界值：

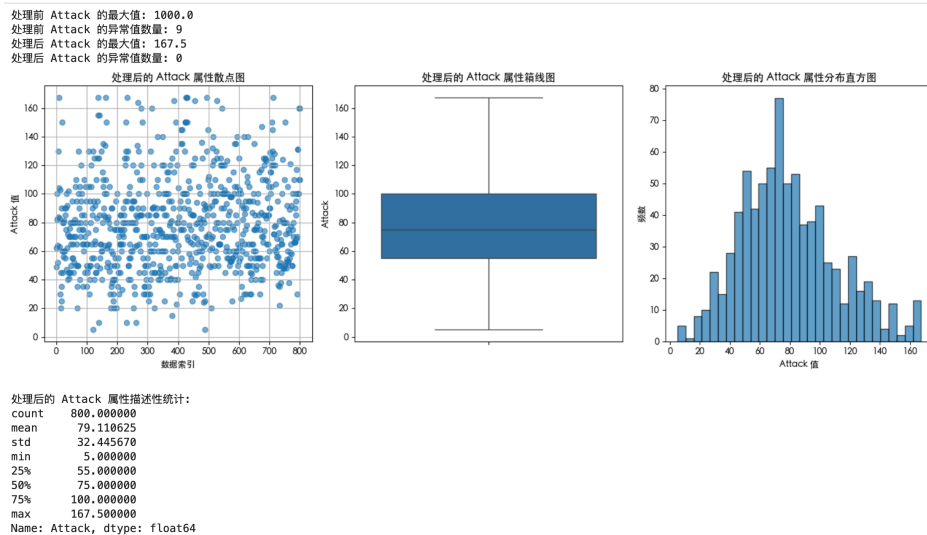
```
1 # 处理异常值 - 使用盖帽法
2 print(f"处理前 Attack 的最大值: {df_cleaned['Attack'].max()}")
3 print(f"处理前 Attack 的异常值数量: {len(attack_outliers)}")
4
5 # 将异常值设定为上限值
6 df_cleaned['Attack'] = np.where(df_cleaned['Attack'] > upper_bound, upper_bound, df_cleaned['Attack'])
7 print(f"处理后 Attack 的最大值: {df_cleaned['Attack'].max()}")
8
9 # 验证处理结果
```

```

10 new_attack_outliers = df_cleaned[df_cleaned['Attack'] > upper_bound]
11 print(f"处理后 Attack 的异常值数量: {len(new_attack_outliers)}")
12
13 # 再次可视化, 确认异常值已被处理
14 plt.figure(figsize=(15, 5))
15 plt.subplot(1, 3, 1)
16 plt.scatter( range( len(df_cleaned)), df_cleaned['Attack'], alpha=0.6)
17 plt.xlabel('数据索引')
18 plt.ylabel('Attack 值')
19 plt.title('处理后的 Attack 属性散点图')
20 plt.grid(True)
21 plt.subplot(1, 3, 2)
22 sns.boxplot(data=df_cleaned, y='Attack')
23 plt.title('处理后的 Attack 属性箱线图')
24 plt.subplot(1, 3, 3)
25 plt.hist(df_cleaned['Attack'], bins=30, alpha=0.7, edgecolor='black')
26 plt.xlabel('Attack 值')
27 plt.ylabel('频数')
28 plt.title('处理后的 Attack 属性分布直方图')
29 plt.tight_layout()
30 plt.show()
31
32 # 显示处理后的统计信息
33 print("\n处理后的 Attack 属性描述性统计:")
34 print(df_cleaned['Attack'].describe())

```

处理完后:



3.6 Generation 与 Legendary 列的错误数据处理

通过检查, 发现 Generation 和 Legendary 列中的某些数据存在错误的属性值交换现象, 具体来说, Legendary 列中包含数字, 而 Generation 列则为 FALSE, 这些行显然是数据录入时的错误。通过交换这两列的值来修正这些数据:

```
1 # 方法：查找Legendary列中包含数字的数据
2 # 先将Legendary列转换为字符串
3 df_cleaned['Legendary_str'] = df_cleaned['Legendary'].astype( str)
4
5 # 查找Legendary列中的数字值
6 legendary_issues = df_cleaned[df_cleaned['Legendary_str']. str.isdigit()]
7 if len(legendary_issues) > 0:
8     print(legendary_issues[['Name', 'Generation', 'Legendary']])
9
10 # 将legendary_issues作为suspect_rows
11 suspect_rows = legendary_issues
12
13 # 如果找到了疑似错误的数据，交换Generation和Legendary的值
14 print(f"找到了 {len(suspect_rows)} 条疑似错误数据，准备交换属性值...")
15
16 # 获取这些行的索引
17 suspect_indexes = suspect_rows.index
18
19 # 显示交换前的值
20 print("\n交换前的数据:")
21 print(df_cleaned.loc[suspect_indexes, ['Name', 'Generation', 'Legendary']])
22
23 # 交换Generation和Legendary的值
24 temp_gen = df_cleaned.loc[suspect_indexes, 'Generation'].copy()
25 df_cleaned.loc[suspect_indexes, 'Generation'] = df_cleaned.loc[suspect_indexes, 'Legendary']
26 df_cleaned.loc[suspect_indexes, 'Legendary'] = temp_gen
27
28 # 显示交换后的值
29 print("\n交换后的数据:")
30 print(df_cleaned.loc[suspect_indexes, ['Name', 'Generation', 'Legendary']])
31 print("\n属性交换完成!")
32
33 else:
34     print("未发现Legendary列包含数字值的数据，未需要交换属性的数据。")
```

```
# 显示交换后的值
print("\n交换后的数据:")
print(df_cleaned.loc[suspect_indexes, ['Name', 'Generation', 'Legendary']])
print("\n属性交换完成!")

else:
    print("未发现Legendary列包含数字值的数据，未需要交换属性的数据。")
```

| | Name | Generation | Legendary |
|-----|-----------|------------|-----------|
| 11 | Blastoise | FALSE | 1 |
| 32 | Pikachu | FALSE | 0 |
| 45 | Ninetales | 1 | 0 |
| 130 | Seaking | 1 | 0 |

找到了 4 条疑似错误数据，准备交换属性值...

交换前的数据:

| | Name | Generation | Legendary |
|-----|-----------|------------|-----------|
| 11 | Blastoise | FALSE | 1 |
| 32 | Pikachu | FALSE | 0 |
| 45 | Ninetales | 1 | 0 |
| 130 | Seaking | 1 | 0 |

交换后的数据:

| | Name | Generation | Legendary |
|-----|-----------|------------|-----------|
| 11 | Blastoise | 1 | FALSE |
| 32 | Pikachu | 0 | FALSE |
| 45 | Ninetales | 0 | 1 |
| 130 | Seaking | 0 | 1 |

属性交换完成!

同时也发现了两列都是数字的异常项目，也顺手删除一下：

```
[25]: # 删除Legendary和Generation列中为数字的行
df_cleaned['Generation'] = pd.to_numeric(df_cleaned['Generation'], errors='coerce')
df_cleaned['Legendary'] = pd.to_numeric(df_cleaned['Legendary'], errors='coerce')

# 删除Legendary和Generation列中为数字的行
df_cleaned = df_cleaned.dropna(subset=['Generation', 'Legendary'])

print("\n处理后的数据（去除数字项）：")
print(df_cleaned[['Name', 'Generation', 'Legendary']])
```

```
处理后的数据（去除数字项）：
   Name Generation Legendary
45  Ninetales      0.0      1.0
130 Seaking      0.0      1.0
```

4 实验总结与收获

本实验通过对宝可梦数据集进行清洗与预处理，深化了我对数据预处理方法的理解。在清理数据时，我掌握了如何检测和處理缺失值、异常值以及重复数据。此外，实验过程中使用的数据可视化技术使得异常值的识别变得更加直观。

通过此次实验，我深刻体会到数据清洗在数据分析中的重要性。只有确保数据的准确性和一致性，后续的分析结果才能更加可靠。此次实践为后续的深度数据分析和机器学习模型训练打下了坚实的基础。

附录 Appendix

完整代码 dataImport.py :

```
1  # %%
2  # 数据操作和处理
3  import pandas as pd
4  import numpy as np
5
6  # 数据可视化
7  import matplotlib.pyplot as plt
8  import seaborn as sns
9
10 # # 使图表在Notebook内显示
11 # %matplotlib
12 # inline
13
14 # 使用 macOS 默认中文字体（优先苹方），避免中文显示为方块
15 plt.rcParams['font.sans-serif'] = ['PingFang SC', 'Heiti TC', 'Hiragino Sans GB', 'Arial Unicode MS'
16     ]
17 plt.rcParams['axes.unicode_minus'] = False
18
19 # 使用pandas读取CSV文件
20 df = pd.read_csv("Pokemon.csv")
21
22 # 查看数据的基本信息（前5行）
23 print("数据形状（行数，列数）:", df.shape)
24 df.head()
25
26 # %%
27 # 查看最后几行数据，确认问题
28 print("原始数据最后5行:")
29 df.tail()
30
31 # %%
32 # 删除最后四行（使用.iloc按位置索引）
33 df_cleaned = df.iloc[:-4].copy()
34
35 # 再次检查数据形状和最后几行，确认删除成功
36 print("删除后数据形状:", df_cleaned.shape)
37 df_cleaned.tail()
38
39 # %%
40 # 查看 Type 2 列的所有唯一值，看看有什么异常
41 print("Type 2 的唯一值:")
42 print(df_cleaned['Type 2'].unique())
43
44 # %%
45 # 将 Type 2 列中的异常值替换为 NaN
46 abnormal_values = ['0', '273', 'A', 'BBB']
47
48 print(f"替换前 Type 2 为异常值的行数: {df_cleaned['Type 2'].isin(abnormal_values).sum()}")
49
50 # 将异常值替换为 NaN
```

```
46 df_cleaned['Type 2'] = df_cleaned['Type 2'].replace(abnormal_values, np.nan)
47
48 print(f"替换后 Type 2 为异常值的行数: {df_cleaned['Type 2'].isin(abnormal_values).sum()}")
49
50 # 再次检查唯一值, 确认替换成功
51 print("清理后 Type 2 的唯一值: ")
52 print(df_cleaned['Type 2'].unique())
53
54 # 查看 Type 2 列的缺失值情况
55 print(f"\nType 2 列的缺失值数量: {df_cleaned['Type 2'].isna().sum()}")
56 print("Type 2 列的值分布: ")
57 print(df_cleaned['Type 2'].value_counts(dropna=False))
58 # %%
59 # 检查是否有完全重复的行
60 duplicates = df_cleaned.duplicated()
61 print(f"完全重复的行数: {duplicates.sum()}")
62
63 if duplicates.sum() > 0:
64     print("重复的行如下: ")
65     print(df_cleaned[duplicates])
66     # 删除重复行
67     df_cleaned = df_cleaned.drop_duplicates()
68     print(f"删除重复行后的数据形状: {df_cleaned.shape}")
69 else:
70     print("没有发现完全重复的行。")
71 # %%
72 # 首先检查 Attack 列的数据类型
73 print("Attack 列的数据类型:", df_cleaned['Attack'].dtype)
74 print("Attack 列的前几个值:")
75 print(df_cleaned['Attack'].head(10))
76
77 # 检查是否有非数值内容
78 print("\nAttack 列的唯一值示例:")
79 print(df_cleaned['Attack'].unique()[:20]) # 查看前20个唯一值
80 # %%
81 # 将 Attack 列从字符串转换为数值类型
82 df_cleaned['Attack'] = pd.to_numeric(df_cleaned['Attack'], errors='coerce')
83
84 # 检查转换后的结果
85 print("转换后 Attack 列的数据类型:", df_cleaned['Attack'].dtype)
86 print("转换后 Attack 列的描述性统计:")
87 print(df_cleaned['Attack'].describe())
88
89 # 检查是否有缺失值 (即无法转换的值)
90 missing_attack = df_cleaned['Attack'].isna().sum()
91 print(f"转换后产生的缺失值数量: {missing_attack}")
92
93 if missing_attack > 0:
94     print("包含缺失 Attack 值的行:")
```

```
95     print(df_cleaned[df_cleaned['Attack'].isna()][['Name', 'Attack']])
96 # %%
97 # 使用散点图可视化Attack属性
98 plt.figure(figsize=(12, 6))
99 plt.scatter( range( len(df_cleaned)), df_cleaned['Attack'], alpha=0.6)
100 plt.xlabel('数据索引')
101 plt.ylabel('Attack 值')
102 plt.title('Attack 属性散点图 (识别异常值) ')
103 plt.grid(True)
104 plt.show()
105
106 # 使用箱线图可视化
107 plt.figure(figsize=(10, 6))
108 sns.boxplot(data=df_cleaned, y='Attack')
109 plt.title('Attack 属性箱线图')
110 plt.show()
111
112 # 使用描述性统计查看分布
113 print("Attack 属性的描述性统计:")
114 print(df_cleaned['Attack'].describe())
115
116 # 使用IQR法则识别异常值
117 Q1 = df_cleaned['Attack'].quantile(0.25)
118 Q3 = df_cleaned['Attack'].quantile(0.75)
119 IQR = Q3 - Q1
120 lower_bound = Q1 - 1.5 * IQR
121 upper_bound = Q3 + 1.5 * IQR
122
123 print(f"\nQ1 (25%分位数): {Q1}")
124 print(f"\nQ3 (75%分位数): {Q3}")
125 print(f"\nIQR: {IQR}")
126 print(f"\n异常值下界 (Q1 - 1.5*IQR): {lower_bound}")
127 print(f"\n异常值上界 (Q3 + 1.5*IQR): {upper_bound}")
128
129 # 找出Attack值大于上界的异常值
130 attack_outliers = df_cleaned[df_cleaned['Attack'] > upper_bound]
131 print(f"\nAttack属性异常高的宝可梦有 {len(attack_outliers)} 个: ")
132 print(attack_outliers[['Name', 'Attack']])
133
134 # 特别关注那个极高的值840
135 if 840 in df_cleaned['Attack'].values:
136     extreme_outlier = df_cleaned[df_cleaned['Attack'] == 840]
137     print(f"\n特别高的异常值 (840):")
138     print(extreme_outlier[['Name', 'Attack', 'HP', 'Defense', 'Type 1', 'Type 2']])
139 # %%
140 # 处理异常值 - 使用盖帽法
141 print(f"处理前 Attack 的最大值: {df_cleaned['Attack'].max()}")
142 print(f"处理前 Attack 的异常值数量: {len(attack_outliers)}")
143
```

```
144 # 将异常值设定为上限值
145 df_cleaned['Attack'] = np.where(df_cleaned['Attack'] > upper_bound, upper_bound, df_cleaned['Attack']
    ])
146
147 print(f"处理后 Attack 的最大值: {df_cleaned['Attack'].max()}")
148
149 # 验证处理结果
150 new_attack_outliers = df_cleaned[df_cleaned['Attack'] > upper_bound]
151 print(f"处理后 Attack 的异常值数量: {len(new_attack_outliers)}")
152
153 # 再次可视化, 确认异常值已被处理
154 plt.figure(figsize=(15, 5))
155
156 plt.subplot(1, 3, 1)
157 plt.scatter( range( len(df_cleaned)), df_cleaned['Attack'], alpha=0.6)
158 plt.xlabel('数据索引')
159 plt.ylabel('Attack 值')
160 plt.title('处理后的 Attack 属性散点图')
161 plt.grid(True)
162
163 plt.subplot(1, 3, 2)
164 sns.boxplot(data=df_cleaned, y='Attack')
165 plt.title('处理后的 Attack 属性箱线图')
166
167 plt.subplot(1, 3, 3)
168 plt.hist(df_cleaned['Attack'], bins=30, alpha=0.7, edgecolor='black')
169 plt.xlabel('Attack 值')
170 plt.ylabel('频数')
171 plt.title('处理后的 Attack 属性分布直方图')
172
173 plt.tight_layout()
174 plt.show()
175
176 # 显示处理后的统计信息
177 print("\n处理后的 Attack 属性描述性统计:")
178 print(df_cleaned['Attack'].describe())
179 # %%
180 # 首先检查这两列的数据类型和内容 (不排序)
181 print("Generation 列的数据类型:", df_cleaned['Generation'].dtype)
182 print("Generation 列的所有唯一值:")
183 print(df_cleaned['Generation'].unique())
184
185 print("\nLegendary 列的数据类型:", df_cleaned['Legendary'].dtype)
186 print("Legendary 列的所有唯一值:")
187 print(df_cleaned['Legendary'].unique())
188 # %%
189 # 方法: 查找Legendary列中包含数字的数据
190 # 先将Legendary列转换为字符串
191 df_cleaned['Legendary_str'] = df_cleaned['Legendary'].astype( str)
```

```
192
193 # 查找Legendary列中的数字值
194 legendary_issues = df_cleaned[df_cleaned['Legendary_str'].str.isdigit()]
195 if len(legendary_issues) > 0:
196     print(legendary_issues[['Name', 'Generation', 'Legendary']])
197
198 # 将legendary_issues作为suspect_rows
199 suspect_rows = legendary_issues
200
201 # 如果找到了疑似错误的数据，交换Generation和Legendary的值
202 print(f"找到了 {len(suspect_rows)} 条疑似错误数据，准备交换属性值...")
203
204 # 获取这些行的索引
205 suspect_indexes = suspect_rows.index
206
207 # 显示交换前的值
208 print("\n交换前的数据:")
209 print(df_cleaned.loc[suspect_indexes, ['Name', 'Generation', 'Legendary']])
210
211 # 交换Generation和Legendary的值
212 temp_gen = df_cleaned.loc[suspect_indexes, 'Generation'].copy()
213 df_cleaned.loc[suspect_indexes, 'Generation'] = df_cleaned.loc[suspect_indexes, 'Legendary']
214 df_cleaned.loc[suspect_indexes, 'Legendary'] = temp_gen
215
216 # 显示交换后的值
217 print("\n交换后的数据:")
218 print(df_cleaned.loc[suspect_indexes, ['Name', 'Generation', 'Legendary']])
219
220 print("\n属性交换完成!")
221
222 else:
223     print("未发现Legendary列包含数字值的数据，不需要交换属性的数据。")
224 # %%
225 # 删除Legendary和Generation列中为数字的行
226 df_cleaned['Generation'] = pd.to_numeric(df_cleaned['Generation'], errors='coerce')
227 df_cleaned['Legendary'] = pd.to_numeric(df_cleaned['Legendary'], errors='coerce')
228
229 # 删除Legendary和Generation列中为数字的项
230 df_cleaned = df_cleaned.dropna(subset=['Generation', 'Legendary'])
231
232 print("\n处理后的数据（去除数字项）:")
233 print(df_cleaned[['Name', 'Generation', 'Legendary']])
```