实验 2

```
import pandas as pd
import numpy as np

# 1. 读取数据
try:
    df = pd.read_csv("D:/vscode/BigDataAnalysisPractice/lab2/Pokemon.csv", encoding='utf-8')
except UnicodeDecodeError:
    # 若 UTF-8 失败，适配 UTF-8 带 BOM 格式（避免文档中特殊字符解码错误）
    df = pd.read_csv("D:/vscode/BigDataAnalysisPractice/lab2/Pokemon.csv", encoding='utf-8-sig')

print("原始数据形状：", df.shape)
print("\n 前 5 行数据（验证读取结果）：")
print(df.head())

# 2. 删除最后两行无意义的空行
df = df.dropna(how='all')  # 仅删除全为空值的行，保留含部分有效数据的行
print("\n 删除无意义空行后数据形状：", df.shape)

# 3. 处理重复值
print("\n 原始数据中重复行数量：", df.duplicated().sum())
df = df.drop_duplicates()  # 删除完全重复的行
print("删除重复行后数据形状：", df.shape)

# 4. 处理 Type 2 列异常值
print("\nType 2 列原始唯一值（部分）：", df['Type 2'].unique()[:10])
# 替换异常值
df['Type 2'] = df['Type 2'].replace(
    [0, '0', '273', 'A', 'BBB', 'undefined'],  # 覆盖文档及实际数据中的异常值
    np.nan  # 按文档要求“清空”异常值，用 NaN 表示缺失
)
print("Type 2 列修正后缺失值数量：", df['Type 2'].isnull().sum())

# 5. 处理 Attack 列异常值
# 步骤 1：强制转换为数值型
print("\nAttack 列转换前数据类型：", df['Attack'].dtype)
df['Attack'] = pd.to_numeric(df['Attack'], errors='coerce')  # 非数值转为 NaN
# 步骤 2：删除 Attack 列缺失值（避免后续 max()计算报错，保证数据有效性）
df = df.dropna(subset=['Attack'])
print("Attack 列转换后数据类型：", df['Attack'].dtype)
```

```
PS D:\vscode> & E:/ANACONDA3/envs/test/python.exe d:/vscode/BigDataAnalysisPractice/lab2/l2.py
原始数据形状： (810, 13)

前5行数据（验证读取结果）：
   #             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

删除无意义空行后数据形状： (809, 13)

原始数据中重复行数量： 7
删除重复行后数据形状： (802, 13)

Type 2列原始唯一值（部分）： ['Poison' nan 'Flying' 'Dragon' '0' 'Ground' '273' 'Fairy' 'Grass'
 'Fighting']
Type 2列修正后缺失值数量： 388

Attack列转换前数据类型： object
Attack列转换后数据类型： float64
```

```
# 步骤 3：统计并定位 Attack 列异常值
print("\nAttack 列数值统计（验证异常值范围）：")
print(df['Attack'].describe())  # 显示均值、最大值、分位数等，辅助判断异常值
max_attack = df['Attack'].max()
print(f"\nAttack 列最大值（异常值候选）：{max_attack}")
print("最大值对应行（定位异常数据）：")
# 只显示关键列，便于分析异常数据的宝可梦信息
print(df[df['Attack'] == max_attack][['#', 'Name', 'Type 1', 'Type 2', 'Attack']])

# 步骤 4：过滤 Attack 列极端异常值
# 基于 99 分位数过滤：保留 99%常规数据，剔除极端值
attack_99 = df['Attack'].quantile(0.99)
df = df[df['Attack'] <= attack_99]
print(f"\n 基于 99 分位数（{attack_99}）过滤极端值后，数据形状：", df.shape)

# 6. 修正 Generation 与 Legendary 列置换问题
print("\n 修正前关键列数据类型：")
print("Generation 列数据类型：", df['Generation'].dtype)
print("Legendary 列数据类型：", df['Legendary'].dtype)
```

```
Attack列数值统计（验证异常值范围）：
count     800.000000
mean       81.095000
std        53.245327
min         5.000000
25%        55.000000
50%        75.000000
75%       100.000000
max      1000.000000
Name: Attack, dtype: float64

Attack列最大值（异常值候选）：1000.0
最大值对应行（定位异常数据）：
       #    Name  Type 1 Type 2  Attack
140  128  Tauros  Normal    NaN  1000.0

基于99分位数（170.0）过滤极端值后，数据形状： (793, 13)

修正前关键列数据类型：
Generation列数据类型： object
Legendary列数据类型： object
```

```
# 步骤 1：转换列类型，检测置换特征（置换表现为：Generation 含布尔值，Legendary 含数字）
df['Generation'] = pd.to_numeric(df['Generation'], errors='coerce')   # 非数字转为 NaN
# 统一 Legendary 列为布尔型
df['Legendary'] = df['Legendary'].map(
    {'TRUE': True, 'FALSE': False, True: True, False: False},
    na_action='ignore'
)

# 步骤 2：定位并交换置换行数据
# 置换行特征：Legendary 列可转为数字
swap_rows = pd.to_numeric(df['Legendary'], errors='coerce').notna()
if swap_rows.sum() > 0:
    print(f"\n 发现{swap_rows.sum()}行数据的 Generation 与 Legendary 列置换，已修正")
```

```
发现785行数据的Generation与Legendary列置换，已修正
d:\vscode\BigDataAnalysisPractice\lab2\l2.py:77: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pand
as. Value '[False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, F
alse, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
 False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False
, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fals
e, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fal
se, False, False, False, False, False, True, True, True, False, False, False, True, True, False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
 False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False
, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fals
e, False, False, False, False, False, False, False, False, False, False, False, False, False, True, True, True, False, False, False, False, True, True, False, False, F
alse, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
 False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False
, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fals
e, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fal
se, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, True, True, True
, True, True, True, True, True, True, True, True, True, True, True, True, False, False, False, False, False, False, False, False, False, False, False, False, False, Fa
lse, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, F
alse, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
 False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, True, True, True, True, True, Tru
e, True, True, True, False, False, False, True, True, True, True, True, False, False, False, False, False, False, False, False, False, False, False, False, False, Fals
e, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fal
se, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fa
lse, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, F
alse, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
 False, False, False, False, False, False, False, False, False, False, False, False, True, True, True, True, True, True, True, True, True, True, True, True, True, True
, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fals
e, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fal
se, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fa
lse, False, False, False, False, False, False, False, True, True, True, True, True, True, True]' has dtype incompatible with float64, please explicitly cast to a
 compatible dtype first.
  df.loc[swap_rows, ['Generation', 'Legendary']] = df.loc[swap_rows, ['Legendary', 'Generation']].values
```

```
    # 交换两列数据
    df.loc[swap_rows, ['Generation', 'Legendary']] = df.loc[swap_rows, ['Legendary', 'Generation']].values
    # 重新转换类型，确保修正后数据类型正确
    df['Generation'] = pd.to_numeric(df['Generation'], errors='coerce')
    df['Legendary'] = df['Legendary'].map({'TRUE': True, 'FALSE': False}, na_action='ignore')

print("\n 修正后关键列数据类型：")
print("Generation 列唯一值：", sorted(df['Generation'].dropna().unique()))
print("Legendary 列唯一值：", df['Legendary'].unique())

# 7. 统一处理其他数值型列
numeric_cols = ['HP', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed', 'Total']
for col in numeric_cols:
    # 转换为数值型，非数值转为 NaN
    df[col] = pd.to_numeric(df[col], errors='coerce')
    # 删除缺失值，保证数据有效性
    df = df.dropna(subset=[col])

# 8. 最终缺失值检查
print("\n 清洗后各列缺失值数量（验证数据完整性）：")
print(df.isnull().sum())

# 9. 保存清洗后的数据
output_path = "D:/vscode/BigDataAnalysisPractice/lab2/Pokemon_cleaned.csv"
df.to_csv(output_path, index=False, encoding='utf-8')
print(f"\n 数据清洗完成！已保存至：{output_path}")

# 10. 输出清洗后数据概览
```

```
print("\n 清洗后数据前 5 行：")
print(df.head())
print("\n 清洗后数据最终形状：", df.shape)
print("\n 数据清洗流程完成（符合文档中“数据质量实践”全部实验目标）")
```

```
修正后关键列数据类型:
Generation列唯一值:  [np.float64(0.0), np.float64(1.0), np.float64(4.0)]
Legendary列唯一值:  [nan]

清洗后各列缺失值数量（验证数据完整性）:
#               0
Name            0
Type 1          1
Type 2        383
Total           0
HP              0
Attack          0
Defense         0
Sp. Atk         0
Sp. Def         0
Speed           0
Generation      3
Legendary     792
dtype: int64

数据清洗完成! 已保存至: D:/vscode/BigDataAnalysisPractice/lab2/Pokemon_cleaned.csv

清洗后数据前5行:
   #                   Name Type 1  Type 2  Total    HP  Attack  Defense  Sp. Atk  Sp. Def  Speed  Generation Legendary
0  1              Bulbasaur  Grass  Poison    318  45.0    49.0       49     65.0     65.0     45         0.0       NaN
1  2                Ivysaur  Grass  Poison    405  60.0    62.0       63     80.0     80.0     60         0.0       NaN
2  3               Venusaur  Grass  Poison    525  80.0    82.0       83    100.0    100.0     80         0.0       NaN
3  3  VenusaurMega Venusaur  Grass  Poison    625  80.0   100.0      123    122.0    120.0     80         0.0       NaN
4  4             Charmander   Fire     NaN    309  39.0    52.0       43     60.0     50.0     65         0.0       NaN

清洗后数据最终形状:  (792, 13)

数据清洗流程完成（符合文档中“数据质量实践”全部实验目标）
```