山东大学 计算机科学与技术 学院

大数据分析与实践 课程实验报告

学号：202300130005	姓名：于佳杭	班级：23 数据
实验题目：数据质量实践		
实验学时：2	实验日期： 2025.9.20	

实验步骤与内容：

1.加载数据集: 使用 Pandas 库加载名为 Pokemon.csv 的数据集，并查看其原始维度（810 行，13 列）。

[28]:
```
import pandas as pd
import numpy as np
file_path = r'D:\Pokemon.csv'
df = pd.read_csv(file_path, encoding='latin1')
print("数据集加载成功！")
print("原始数据形状:", df.shape)
df_cleaned = df.copy()
```

```
数据集加载成功！
原始数据形状: (810, 13)
```

2.删除无关数据:
问题: 数据集末尾存在两行无意义或不完整的的记录。
步骤: 识别并删除数据集的最后两行。
结果: 数据集维度变为（808 行，13 列

[29]:
```
# --- 问题一：删除最后两行 ---
print("处理前的数据形状:", df_cleaned.shape)

# 使用 .index 获取最后两行的索引，然后使用 .drop() 删除
# inplace=True 表示直接在原数据上修改
rows_to_drop = df_cleaned.tail(2).index
df_cleaned.drop(rows_to_drop, inplace=True)

print("处理后的数据形状:", df_cleaned.shape)
print("\n成功删除最后两行数据。")
```

```
处理前的数据形状: (810, 13)
处理后的数据形状: (808, 13)

成功删除最后两行数据。
```

3.处理缺失值:
问题: 'Type 2' (第二属性) 列存在大量缺失值（NaN），共计 383 个。
步骤: 将 'Type 2' 列中所有的缺失值（NaN）填充为字符串 'None'，使其成为一个独立的类别，表示该宝可梦只有单一属性。

结果：处理后 'Type 2' 列不再有缺失值，'None' 成为一个有效类别，数量为 383。

[26]:
```
# --- 问题二：将 'Type 2' 列中的缺失值(NaN)替换为'None'
print("处理前 'Type 2' 的值分布情况:")
# 使用 .value_counts() 可以查看每个类别出现的次数，NaN默认不显示
display(df_cleaned['Type 2'].value_counts())
nan_count_before = df_cleaned['Type 2'].isna().sum()
print(f"处理前共有 {nan_count_before} 个缺失值 (NaN)。")

df_cleaned['Type 2'].fillna('None', inplace=True)


print("\n-----------------------------------")
print("处理后 'Type 2' 的值分布情况:")
# 再次查看值分布，现在 'None' 已经成为一个明确的类别
display(df_cleaned['Type 2'].value_counts())
nan_count_after = df_cleaned['Type 2'].isna().sum()
print(f"处理后共有 {nan_count_after} 个缺失值 (NaN)。")
print("\n成功将所有缺失的第二属性替换为 'None' 类别。")
```

```
处理前 'Type 2' 的值分布情况:
Flying      97
Ground      35
Poison      34
Psychic     33
Fighting    26
Grass       25
Fairy       23
Steel       22
Dark        20
Dragon      18
Ice         14
Water       14
Rock        14
Ghost       14
Fire        12
```

4.删除重复行:

问题：数据集中存在完全重复的记录。

步骤：检查并删除了数据集中的所有重复行，共发现并删除了 6 条。

结果：数据集维度变为（802 行，13 列）。

成功将所有缺失的第二属性替换为 'None' 类别。

[30]:
```
# --- 问题三：删除重复行 ---

print("处理前的数据形状:", df_cleaned.shape)

# 计算有多少重复行
duplicate_count = df_cleaned.duplicated().sum()
print(f"发现 {duplicate_count} 条重复行。")

# 删除重复行
# keep='first' 表示保留第一次出现的记录，删除后面的重复项
df_cleaned.drop_duplicates(inplace=True, keep='first')

print("处理后的数据形状:", df_cleaned.shape)
print("\n成功删除所有重复行。")
```

```
处理前的数据形状: (808, 13)
发现 6 条重复行。
处理后的数据形状: (802, 13)

成功删除所有重复行。
```

5.处理异常值:

问题: 'Attack' (攻击力) 属性中存在一个极端异常值（1000），远高于其他数值。

步骤: 首先，将'Attack'列的数据类型转换为数值型。然后，计算该列的 99%分位数值（170.0），并将所有超过此分位数的攻击力数值修正为此值。

结果: 成功将极端的攻击力值从 1000 修正为 170.0，使数据分布更合理。

成功删除所有重复行。

[24]:
```
# --- 问题四：处理 Attack 异常值 ---

# **【修正】** 在计算前，必须将'Attack'列转换为数值类型
#  我们使用 pd.to_numeric，errors='coerce' 会将无法转换的值设为缺失值(NaN)
df_cleaned['Attack'] = pd.to_numeric(df_cleaned['Attack'], errors='coerce')

# （可选步骤）如果存在因转换失败而产生的NaN，可以用中位数等填充
# df_cleaned['Attack'].fillna(df_cleaned['Attack'].median(), inplace=True)

# --- 现在，您原来的代码就可以正常运行了 ---

# 首先，查看Attack属性的描述性统计和最大值
print("处理前 'Attack' 属性的统计信息:")
print(df_cleaned['Attack'].describe())

# 找到最大值对应的宝可梦
max_attack_pokemon = df_cleaned.loc[df_cleaned['Attack'].idxmax()]
print("\n拥有最高 Attack 值的宝可梦:\n", max_attack_pokemon)

# 一种常见的处理方法是用该列的99%分位数来替换极端最大值
# 这样既修正了异常，又保留了数据的高端分布
ninety_ninth_percentile = df_cleaned['Attack'].quantile(0.99)
print(f"\n'Attack' 的99%分位数值为: {ninety_ninth_percentile}")

# 将所有超过这个值的 Attack 都设置为99%分位数的值
df_cleaned.loc[df_cleaned['Attack'] > ninety_ninth_percentile, 'Attack'] = ninety_

print("\n处理后的 'Attack' 属性最大值:", df_cleaned['Attack'].max())
print("成功将过高的 Attack 值修正。")
```

6.修正数据错位:

问题: 数据集中有 6 条记录的 'Generation' (世代) 和 'Legendary' (是否为传说) 两个字段的值发生了互换。

步骤: 通过检查 'Generation' 列中的非数值类型数据来定位这些错位的行，然后将这两列的值进行交换，恢复其正确位置。

结果: 6 条记录的世代和传说属性得到成功修正。

[25]:
```
# --- 问题五：修正 Generation 和 Legendary 置换 ---

# 正常情况下，'Generation' 应该是数字，'Legendary' 应该是布尔值 (True/False)
# 我们需要找到 'Generation' 列的值是布尔值的行
# Series.apply(type) is slow, a better way is to check the type of the first eleme
# However, for simplicity and clarity in finding mixed types, we can do this:
try:
    # 尝试将 'Generation' 列转换为数字，无法转换的将变成 NaT/NaN
    gen_as_numeric = pd.to_numeric(df_cleaned['Generation'], errors='coerce')
    # 找到那些原本不是数字的行
    swapped_rows_mask = gen_as_numeric.isna()

    print(f"找到 {swapped_rows_mask.sum()} 条可能被置换的数据。")

    if swapped_rows_mask.sum() > 0:
        print("\n置换前的数据:")
        print(df_cleaned[swapped_rows_mask][['Name', 'Generation', 'Legendary']])

        # 对找到的行进行数据交换
        for index in df_cleaned[swapped_rows_mask].index:
            # 使用临时变量保存原始值
            original_gen = df_cleaned.loc[index, 'Generation']
            original_leg = df_cleaned.loc[index, 'Legendary']

            # 执行交换
            df_cleaned.loc[index, 'Generation'] = original_leg
            df_cleaned.loc[index, 'Legendary'] = original_gen

        print("\n置换后的数据:")
        print(df_cleaned.loc[swapped_rows_mask.index][['Name', 'Generation', 'Lege
        print("\n成功修正 Generation 和 Legendary 的数据。")
    else:
        print("未发现需要修正的 Generation 和 Legendary 数据。")

except Exception as e:
    print(f"处理时发生错误：{e}")
```

找到 6 条可能被置换的数据。

7.保存清洗后的数据:

步骤：将经过以上所有步骤清洗和处理后的数据集保存为一个新的 CSV 文件 Pokemon_clean.csv，以供后续分析使用。

成功修正 Generation 和 Legendary 的数据。

[32]:
```
out_path = "D:\Pokemon_clean.csv"
df.to_csv(out_path, index=False, encoding="utf-8")
print("已保存：", out_path)
```

已保存： D:\Pokemon_clean.csv

结论分析与体会：本次实验通过使用 Python 的 Pandas 库，成功对宝可梦数据集进行了一系列系统性的数据清洗工作。实验流程清晰地展示了在数据分析前进行预处理的重要性。