山东大学 计算机科学与技术 学院

大数据分析实践 课程实验报告

学号：202300130262	姓名： 何青青	班级： 23 数据
实验题目：数据质量实践		
实验学时：2	实验日期： 2025/9/26	

实验目标：

本次实验主要围绕宝可梦数据集进行分析，考察在拿到数据后如何对现有的数据进行预处理清洗操作，建立起对于脏数据、缺失数据等异常情况的一套完整流程的认识。

实验环境：

Jupyter Notebook，python3.9

实验步骤与内容：

1、数据准备

```
import chardet
import pandas as pd

# 检测文件编码
with open('Pokemon.csv', 'rb') as f:
    result = chardet.detect(f.read())

# 使用检测到的编码加载数据集
df = pd.read_csv('Pokemon.csv', encoding=result['encoding'])
```

(a) 检测文件编码：使用 chardet 库检测 Pokemon.csv 文件的编码格式。通过以二进制模式打开文件，读取文件内容并使用 chardet.detect 方法来确定其编码。

(b) 加载数据集:依据检测到的编码格式,使用 pandas 库的 read_csv 函数加载 Pokemon.csv 文件中的数据到 DataFrame 对象 df 中。

2、数据清理

在查看 Pokemon.csv 文件后，发现该数据集存在几个明显的质量问题，如 undefined 无意义数据、空白值、重复值、异常值、错误值等，需要对这部分数据进行处理。

①删除无意义数据

```
undefined undefined undefined undefined undefined undefined undefined undefined und
undefined undefined undefined undefined undefined undefined undefined undefined und
```

```
# 1. 删除最后两行无意义数据
rows, columns = df.shape
df = df[:-2]
print('删除了最后两行无意义数据。')
```

对于数据集最后两行的 undefined 无意义数据，直接将其删除。

② 处理空白值

505	90	92	87	750	85	76	1	FALSE
46	57	40	40	40	50	1	FALSE	
365	61	72	57	55	55	65	1	FALSE

(a) 确定检查列：在查看数据集后，可以知道 Type 2 这一列存在空白值是合法的，Total 列存在空白值可以通过后续计算进行填充，因此筛选出除 Type 2 列和 Total 列之外的所有列，作为检查空白值的目标列。

```
# 2. 处理除 Type 2 列和 Total 列外其他列的空白值
columns_to_check = [col for col in df.columns if col not in ['Type 2', 'Total']]
nan_rows = df[columns_to_check].isnull().any(axis=1)
nan_count = nan_rows.sum()
df = df[~nan_rows]
print(f'发现{nan_count}条除 Type 2 列和 Total 列外其他列存在空白值的异常数据，已删除。')
```

(b) 删除空白行：通过 isnull().any(axis=1)方法检测这些列中存在空白值的行，并统计其数量，将存在空白值的行从数据集中删除，并输出删除的异常数据行数。

③ **处理重复值**

11	Metapod	Bug		205	50	20	55	25
11	Metapod	Bug		205	50	20	55	25
13	Weedle	Bug	Poison	195		35	30	20
14	Kakuna	Bug	Poison	205	45	25	50	25
15	Beedrill	Bug	Poison	395	65	90	40	45
15	BeedrillM	Bug	Poison	495	65	150	40	15
17	Pidgeotto	Normal	Flying	349	63	60	55	50
16	Pidgey	Normal	Flying	251	40	45	40	35
17	Pidgeotto	Normal	Flying	349	63	60	55	50

使用 duplicated()方法找出数据集中的重复行，计算重复行的数量，并使用 drop_duplicates()方法删除这些重复行，输出已删除的重复数据条数。

④ **处理 Type 2 列异常值**

```
# 4. 清空 Type 2 列中的异常数值
valid_type2_values = ['Grass', 'Poison', 'Fire', 'Flying', 'Water', 'Bug', 'Nor
type2_anomaly_count = df[~df['Type 2'].isin(valid_type2_values) & df['Type 2'].
df['Type 2'] = df['Type 2'].where(df['Type 2'].isin(valid_type2_values), None)
print(f'发现{type2_anomaly_count}条 Type 2 列的异常数据，已将异常值清空。')
```

根据 Type 1 和 Type 2 列的取值分布，可定义 Type 2 列的有效属性值列表，包含如‘Grass’，‘Poison’等常见类型，找出 Type 2 列中不在有效值列表且不为空值的异常数据，并统计其数量，输出发现的异常数据条数以及已清空异常值的提示。

⑤ 数据类型转换

为了使用数学比较方法处理后续可能存在的异常值，先使用 to_numeric()函数将 Attack、Defense、Sp. Atk、Speed 列转换为数值类型，便于处理和计算。

⑥ 处理 Attack 属性异常值

HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generat
7	1000	95	40	70	110	1
4	840	65	50	64	43	1
106	190	100	154	100	130	1
80	185	115	40	105	75	2

经过查看 Attack 取值分布情况，可以认为 Attack>200 的数据属于异常数据，因此需要找出 Attack 属性中大于 200 的数据，并统计其数量，将 Attack 属性大于 200 的数据行从数据集中删除。

⑦ 处理 Generation 和 Legendary 列异常数据

```
# 假设 Generation 应该是整数，Legendary 应该是布尔值或类似的值
wrong_rows = df[~df['Generation'].astype(str).str.isdigit() & df['Legendary'].astype(str).str.isdigit()]
if not wrong_rows.empty:
    for index in wrong_rows.index:
        df.at[index, 'Generation'], df.at[index, 'Legendary'] = df.at[index, 'Legendary'], df.at[index, 'Generation']
    print('已纠正两条Generation和Legendary属性被置换的数据。')
else:
    print('不存在 Generation 与 Legendary 属性被置换的数据。')
```

(a) 纠正属性置换数据：查找 Generation 和 Legendary 属性被置换的两条数据，将其属性值进行交换，并输出已纠正的提示。

```
# 处理 Generation 列中不是整数的数据
generation_anomaly_count = df[~df['Generation'].astype(str).str.isdigit()].shape[0]
df = df[df['Generation'].astype(str).str.isdigit()]
print(f'发现{generation_anomaly_count}条 Generation 列不是整数的异常数据，已删除。')

# 处理 Generation 列中小于 1 的数据
df['Generation'] = pd.to_numeric(df['Generation'])
generation_less_than_one_count = df[df['Generation'] < 1].shape[0]
df = df[df['Generation'] >= 1]
print(f'发现{generation_less_than_one_count}条 Generation 列小于 1 的异常数据，已删除。')
```

(b) 处理 Generation 列数据：Generation 列数据应为>=1 的整数，因此找出 Generation 列中不是整数、小于 1 的数据，将其删除。

```
# 处理 Legendary 列中不是 TRUE 或 FALSE 的数据
valid_legendary_values = ['TRUE', 'FALSE']
legendary_anomaly_count = df[~df['Legendary'].astype(str).isin(valid_legendary_values)].shape[0]
df = df[df['Legendary'].astype(str).isin(valid_legendary_values)]
print(f'发现{legendary_anomaly_count}条 Legendary 列取值非 TRUE 或 FALSE 的异常数据，已删除。')
```

(c) 处理 Legendary 列非 TRUE 或 FALSE 数据：定义 Legendary 列的有效取值为['TRUE','FALSE']，找出不在该取值范围内的数据并删除。

⑧ 处理 Defense 列负值异常数据

Defense 列数值应>=0，所以找出该列中小于 0 的数据，并删除。

⑨ 处理 Sp. Atk 列异常数据

查看数据集后发现，Sp. Atk 列中大于 200 的数据严重离群，视为异常值，将其删除。

⑩ 处理 Speed 列负值异常数据

Speed 列数值应>=0，所以找出该列中小于 0 的数据，并删除。

3、数据计算与调整

① 重新计算 Total 列

查看数据集可知 Total 列由 HP、Attack、Defense、Sp. Atk、Sp. Def、Speed 这六列

数据计算和得到，但数据集中可能存在计算错误的数据，因此将这几列数据使用to_numeric()转为数值后，记录应保留的小数点位数（以这六列数据中小数点后位数最大的数据为依据），用新的加和结果替换原Total值。

② **调整#列**

对数据集进行删除操作后，编号可能存在错误，因此重新编号：将#列按照数字顺序从 1 开始重新编写，使其与数据集的行数对应。

结论分析与体会：

1、数据质量结论：通过一系列的数据清理和处理操作，成功识别并处理了数据集中存在的多种质量问题，包括无意义数据、空白值、重复值、各列的异常值等。这使得数据集更加干净、规范，提高了数据的质量和可用性，为后续基于该数据集的数据分析、建模等工作奠定了良好的基础。

2、处理方法效果：针对不同类型的数据问题采用了相应的处理方法，如删除、替换、类型转换等，这些方法在代码中得到了有效的实现，并且通过输出提示信息能够清晰地了解每个处理步骤的执行情况和处理结果。从输出的异常数据数量来看，数据集中确实存在较多的质量问题，经过处理后数据得到了明显的优化。

3、实践体会：在整个实验过程中，深刻体会到数据预处理在数据分析工作中的重要性。原始数据往往存在各种质量问题，如果不进行有效的处理，可能会对后续的分析结果产生严重的影响。在编写代码处理数据时，要充分考虑到各种可能出现的情况，对代码进行严谨的逻辑设计和错误处理，以确保代码的稳定性和可靠性。