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：202300130178	姓名： 刘爽	班级： 23 数据
实验题目： 数据清洗		
实验学时：4	实验日期： 2025.9.28	

实验目的：

1. 完成宝可梦数据集的全流程预处理，包括数据读取、质量评估、清洗与验证，解决原始数据中的缺失值、重复值、数据类型错误等问题。
2. 通过可视化手段直观呈现数据质量问题（如缺失值分布）、数值特征分布（如战斗能力值分布）及分类特征规律（如属性类型分布），挖掘数据内在特征。
3. 针对特殊需求优化清洗逻辑：保留 Type 2 属性的空值（宝可梦可仅有一种属性）、删除未定义行、标准化列名（首列重命名为 ID）。
4. 对比清洗前后的数据质量指标（如完整性、重复率），验证清洗流程的有效性，生成可直接用于后续分析的高质量数据集。

实验环境：

python3.9，jupyter notebook

实验过程:

1．**数据读取与编码处理**

编码自动检测：使用 chardet 读取文件前 10000 字节，自动识别文件编码（如 UTF-8、GBK），解决中文/特殊字符读取乱码问题。

```python
def detect_encoding(file_path):
    with open(file_path, 'rb') as f:
        result = chardet.detect(f.read())
    return result['encoding']


# 读取数据
1 usage
def load_pokemon_data(file_path='Pokemon.csv'):
    try:
        # 先尝试自动检测编码
        encoding = detect_encoding(file_path)
        print(f"检测到的文件编码: {encoding}")

        # 尝试不同的编码
        encodings_to_try = [encoding, 'utf-8', 'latin-1', 'cp1252', 'iso-8859-1']

        for enc in encodings_to_try:
            try:
                print(f"尝试使用 {enc} 编码读取文件...")
                df = pd.read_csv(file_path, encoding=enc)
                print(f"成功使用 {enc} 编码读取文件，初始规模：{df.shape}")
                return df
            except UnicodeDecodeError:
                except Exception as e:
                print(f"使用 {enc} 编码时出错: {e}")
                continue

        # 如果所有编码都失败，使用错误处理方式
        print("所有编码尝试失败，使用错误处理方式读取...")
        df = pd.read_csv(file_path, encoding='utf-8', errors='replace')
        return df

    except FileNotFoundError:
        print("文件未找到，请检查文件路径")
        return None
    except Exception as e:
        print(f"读取文件时出错: {e}")
        return None
```

检查数据集的基本信息以及数据集概览：

```python
def data_overview(df):
    print("数据概览")
    print("=" * 50)
    print(f"数据集形状: {df.shape}")
    print("\n前5行数据:")
    print(df.head())

    print("\n数据基本信息:")
    print(df.info())

    print("\n描述性统计:")
    print(df.describe())

    print("\n缺失值统计:")
    missing_data = df.isnull().sum()
    print(missing_data[missing_data > 0])

    return df
```

```
数据集形状: (810, 13)

前5行数据:
   #                   Name Type 1  Type 2  ... Sp. Def Speed Generation Legendary
0  1              Bulbasaur  Grass  Poison  ...      65    45          1     FALSE
1  2                Ivysaur  Grass  Poison  ...      80    60          1     FALSE
2  3               Venusaur  Grass  Poison  ...     100    80          1     FALSE
3  3  VenusaurMega Venusaur  Grass  Poison  ...     120    80          1     FALSE
4  4             Charmander   Fire     NaN  ...      50    65          1     FALSE

[5 rows x 13 columns]
```

```
数据基本信息:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 810 entries, 0 to 809
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   #           807 non-null    object
 1   Name        807 non-null    object
 2   Type 1      806 non-null    object
 3   Type 2      424 non-null    object
 4   Total       807 non-null    object
 5   HP          806 non-null    object
 6   Attack      807 non-null    object
 7   Defense     807 non-null    object
 8   Sp. Atk     807 non-null    object
 9   Sp. Def     807 non-null    object
 10  Speed       807 non-null    object
 11  Generation  807 non-null    object
 12  Legendary   807 non-null    object
dtypes: object(13)
memory usage: 82.4+ KB
None

描述性统计:
          #     Name Type 1  Type 2  ... Sp. Def Speed Generation Legendary
count   807      807    806     424  ...     807   807        807       807
unique  722      801     19      23  ...      94   111          8         8
top     479  Ariados  Water  Flying  ...      80    60          1     FALSE
freq      6        4    112      98  ...      52    44        165       732
```

2．**数据质量检查**

（1）数据类型检查：区分数值型列，验证列名完整性。

```
1 usage
def check_data_types(df):
    print("\n" + "=" * 50)
    print("数据类型检查")
    print("=" * 50)

    # 检查列名，处理可能的编码问题
    print("列名:", df.columns.tolist())

    numeric_columns = df.select_dtypes(include=[np.number]).columns
    categorical_columns = df.select_dtypes(include=['object']).columns

    print(f"数值型列: {list(numeric_columns)}")
    print(f"分类型列: {list(categorical_columns)}")

    return numeric_columns, categorical_columns
```

```
数据类型检查
==================================================
列名: ['#', 'Name', 'Type 1', 'Type 2', 'Total', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed', 'Generation', 'Legendary']
数值型列: []
分类型列: ['#', 'Name', 'Type 1', 'Type 2', 'Total', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed', 'Generation', 'Legendary']
```

（2）重复值检测：

完全重复行（所有列值相同）

```
def detect_duplicates(df):
    print("\n" + "=" * 50)
    print("重复值检测")
    print("=" * 50)

    # 完全重复的行
    duplicate_rows = df[df.duplicated(keep=False)]
    if not duplicate_rows.empty:
        print(f"发现 {len(duplicate_rows)} 行完全重复的数据:")
        # 获取第一列和名称列的列名
        first_col = df.columns[0]
        name_col = 'Name' if 'Name' in df.columns else df.columns[1] if len(df.columns) >
        print(duplicate_rows[[first_col, name_col]].head(10))
    else:
        print("没有发现完全重复的行")
```

```
发现 12 行完全重复的数据:
                #        Name
14             11     Metapod
15             11     Metapod
21             17   Pidgeotto
23             17   Pidgeotto
184           168     Ariados
185           168     Ariados
186           168     Ariados
187           168     Ariados
806     undefined   undefined
807     undefined   undefined
```

名称重复行（Name 列相同但其他属性不同）。

```
# 名称重复的宝可梦
name_col = 'Name' if 'Name' in df.columns else df.columns[1] if len(df.columns) > 1 el
if name_col:
    name_duplicates = df[df.duplicated(name_col, keep=False)]
    if not name_duplicates.empty:
        print(f"\n发现 {len(name_duplicates)} 行名称重复的宝可梦:")
        first_col = df.columns[0]
        print(name_duplicates[[first_col, name_col]].head(10))

return duplicate_rows, name_duplicates
```

```
发现 13 行名称重复的宝可梦:
                #       Name
14             11    Metapod
15             11    Metapod
21             17  Pidgeotto
23             17  Pidgeotto
184           168    Ariados
185           168    Ariados
186           168    Ariados
187           168    Ariados
408           NaN        NaN
806     undefined  undefined
```

（3）异常值检测：

数值型列：IQR 方法（Q1-1.5×IQR 至 Q3+1.5×IQR 为正常范围）；

```
def detect_numeric_outliers(df, numeric_columns):
    for col in numeric_columns:
        if col in df.columns:
            # 确保列是数值型
            df[col] = pd.to_numeric(df[col], errors='coerce')

            # 使用IQR方法检测异常值
            Q1 = df[col].quantile(0.25)
            Q3 = df[col].quantile(0.75)
            IQR = Q3 - Q1
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR

            outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]

            if not outliers.empty:
                print(f"\n{col} 列的异常值 (IQR方法):")
                print(f"正常范围: [{lower_bound:.2f}, {upper_bound:.2f}]")
                print(f"发现 {len(outliers)} 个异常值")
                outlier_info[col] = outliers

                # 显示部分异常值
                first_col = df.columns[0]
                name_col = 'Name' if 'Name' in df.columns else df.columns[1] if len(df.col
                outlier_samples = outliers[[first_col, name_col, col]].head(5)
```

特殊值：检测 0 / 负值（如 HP=0）、科学计数法（如 1e+3）。

```
print("\n" + "=" * 50)
print("特殊值检测")
print("=" * 50)

special_cases = {}
numeric_cols = ['Total', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed']

# 检测0或负值
for col in numeric_cols:
    if col in df.columns:
        # 确保列是数值型
        df[col] = pd.to_numeric(df[col], errors='coerce')
        zero_or_negative = df[df[col] <= 0]
        if not zero_or_negative.empty:
            print(f"\n{col} 列中的0或负值:")
            first_col = df.columns[0]
            name_col = 'Name' if 'Name' in df.columns else df.columns[1] if len(df.col
            print(zero_or_negative[[first_col, name_col, col]].head())
            special_cases[f'{col}_zero_negative'] = zero_or_negative
```

```python
# 检测科学计数法表示的值
for col in numeric_cols:
    if col in df.columns:
        # 检查是否包含科学计数法字符
        sci_notation = df[df[col].astype(str).str.contains('e|E', na=False)]
        if not sci_notation.empty:
            print(f"\n{col} 列中的科学计数法数值:")
            first_col = df.columns[0]
            name_col = 'Name' if 'Name' in df.columns else df.columns[1] if len(df.column
            print(sci_notation[[first_col, name_col, col]].head())
            special_cases[f'{col}_scientific'] = sci_notation

return special_cases
```

```
特殊值检测
==================================================


Defense 列中的0或负值:
        #     Name  Defense
349   315  Roselia    -10.0

Speed 列中的0或负值:
        #      Name  Speed
620   554  Darumaka  -50.0
```

（4）缺失值统计：计算每列缺失值数量及占比，重点关注关键列（如 Name、Total）的缺失情况。

```python
# 确保数值列是数值类型
stat_cols = ['HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed']
for col in stat_cols:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
```

（5）逻辑一致性检查：

Total 列验证：是否等于 HP+Attack+Defense+Sp. Atk+Sp. Def+Speed 之和；

```python
# 确保数值列是数值类型
stat_cols = ['HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed']
for col in stat_cols:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')

# 检查Total列是否等于各属性之和
if all(col in df.columns for col in ['Total', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'S
    calculated_total = df[['HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed']].
    total_mismatch = df[abs(df['Total'] - calculated_total) > 1]
    if not total_mismatch.empty:
        print("Total列与各属性之和不匹配:")
        first_col = df.columns[0]
        name_col = 'Name' if 'Name' in df.columns else df.columns[1] if len(df.columns
        print(total_mismatch[[first_col, name_col, 'Total']].head())
        issues.append('total_mismatch')
```

```
==================================================
Total列与各属性之和不匹配:
      #                Name  Total
9     7            Squirtle  314.0
17   13              Weedle  195.0
38   31           Nidoqueen  505.0
39   32  Nidoran,ô?Poison   46.0
48   41               Zubat  845.0

可能存在Generation和Legendary列置换的情况：
```

列置换检查：Generation（应为数值）与 Legendary（应为布尔值）是否颠倒。

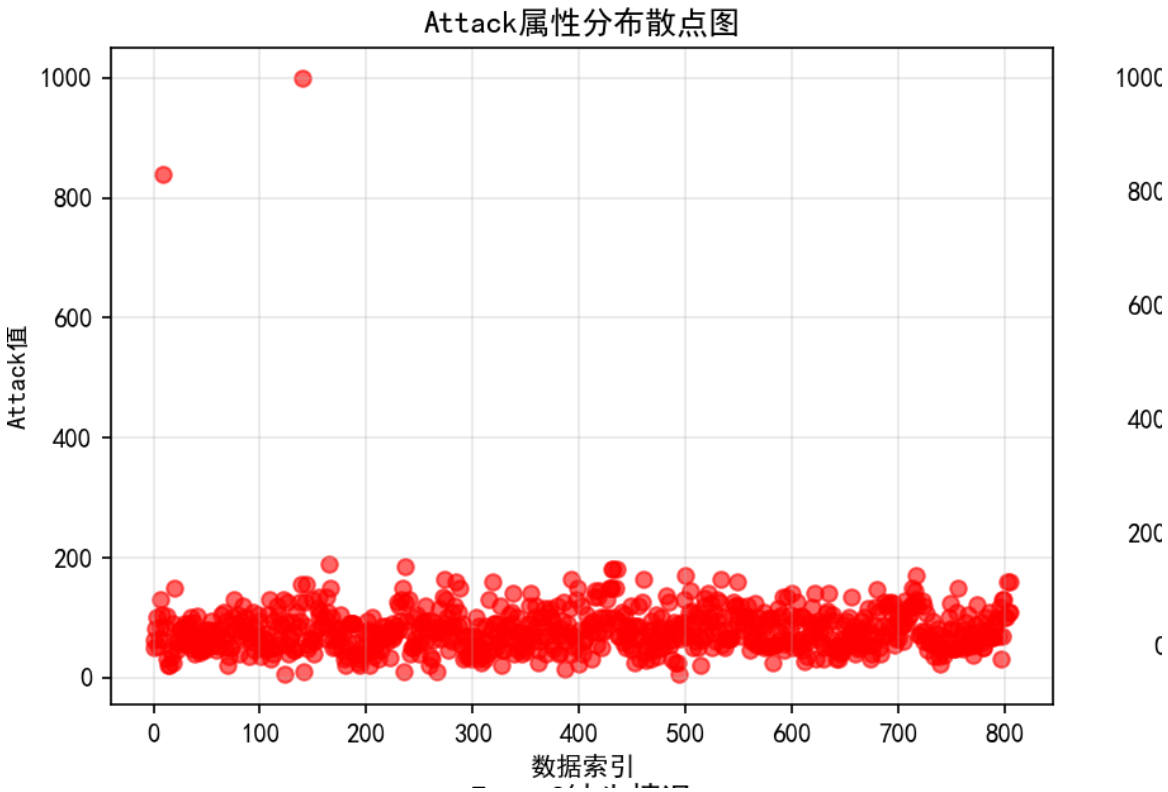
```python
    # 检查Generation和Legendary列是否被置换
    if 'Generation' in df.columns and 'Legendary' in df.columns:
        # 检查Generation列中是否有布尔值
        gen_bool_like = df[df['Generation'].astype(str).str.upper().isin(['TRUE', 'FALSE']
        # 检查Legendary列中是否有数字
        leg_numeric = pd.to_numeric(df['Legendary'], errors='coerce').notna()
        legendary_numeric = df[leg_numeric]

        if not gen_bool_like.empty or not legendary_numeric.empty:
            print("\n可能存在Generation和Legendary列置换的情况:")
            first_col = df.columns[0]
            name_col = 'Name' if 'Name' in df.columns else df.columns[1] if len(df.columns
            if not gen_bool_like.empty:
                print("Generation列中的布尔值:")
                print(gen_bool_like[[first_col, name_col, 'Generation', 'Legendary']].head
            if not legendary_numeric.empty:
                print("Legendary列中的数值:")
                print(legendary_numeric[[first_col, name_col, 'Generation', 'Legendary']].
            issues.append('column_swap')

    return issues
```

```
可能存在Generation和Legendary列置换的情况:
Generation列中的布尔值:
     #              Name Generation Legendary
11   9         Blastoise      FALSE         1
32  25           Pikachu      FALSE         0
39  32  Nidoran,ô?Poison      FALSE       NaN
Legendary列中的数值:
       #       Name Generation Legendary
11     9  Blastoise      FALSE         1
32    25    Pikachu      FALSE         0
45    38  Ninetales          1         0
130  119    Seaking          1         0
```

（6）可视化辅助检测：生成 6 类图表：

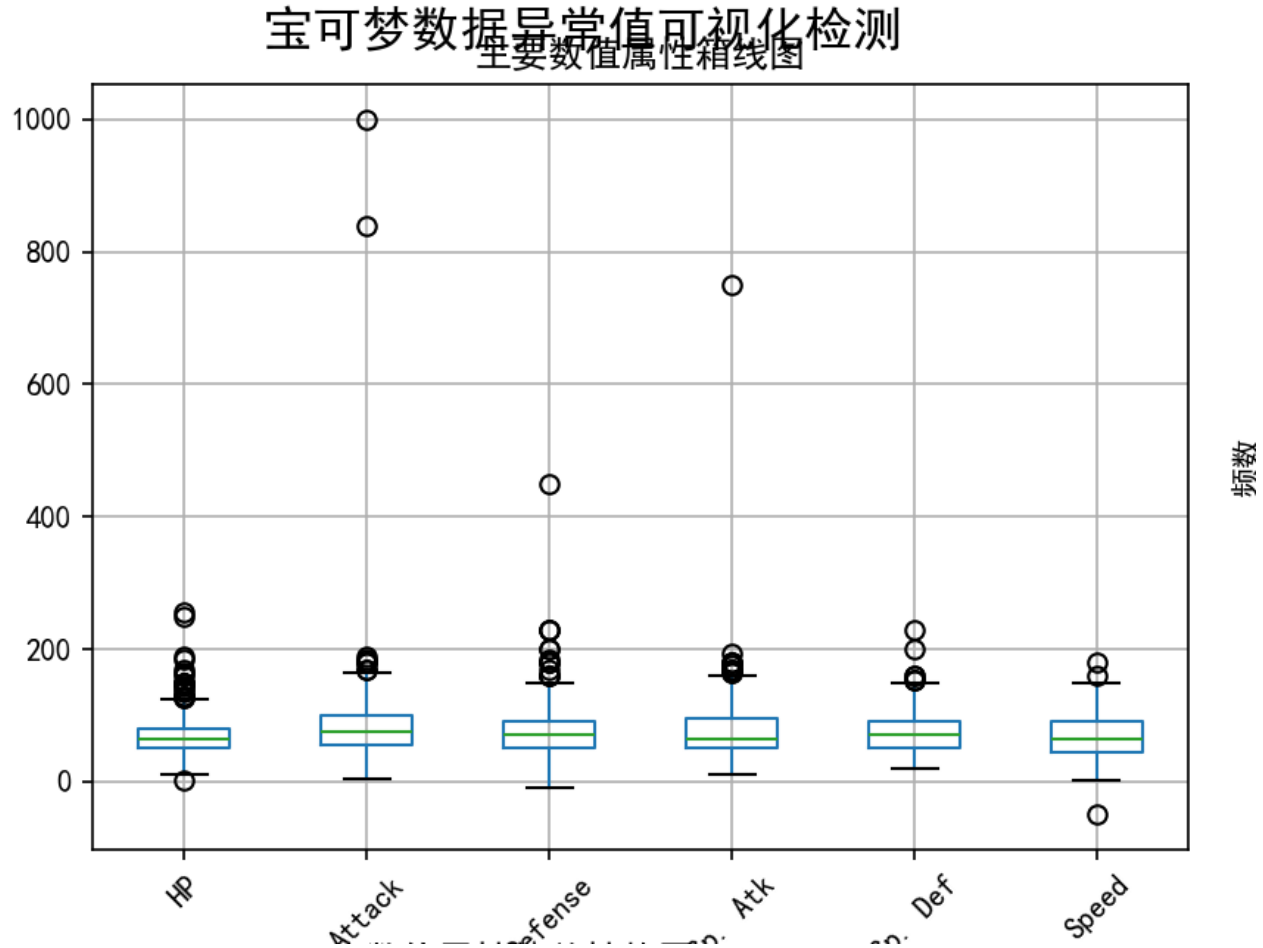
① Attack 属性散点图（观察离散异常值）；

```python
# 1. Attack属性的散点图
if 'Attack' in df.columns:
    df['Attack'] = pd.to_numeric(df['Attack'], errors='coerce')
    axes[0, 0].scatter(range(len(df)), df['Attack'], alpha=0.6, color='red')
    axes[0, 0].set_title('Attack属性分布散点图')
    axes[0, 0].set_xlabel('数据索引')
    axes[0, 0].set_ylabel('Attack值')
    axes[0, 0].grid(True, alpha=0.3)
```



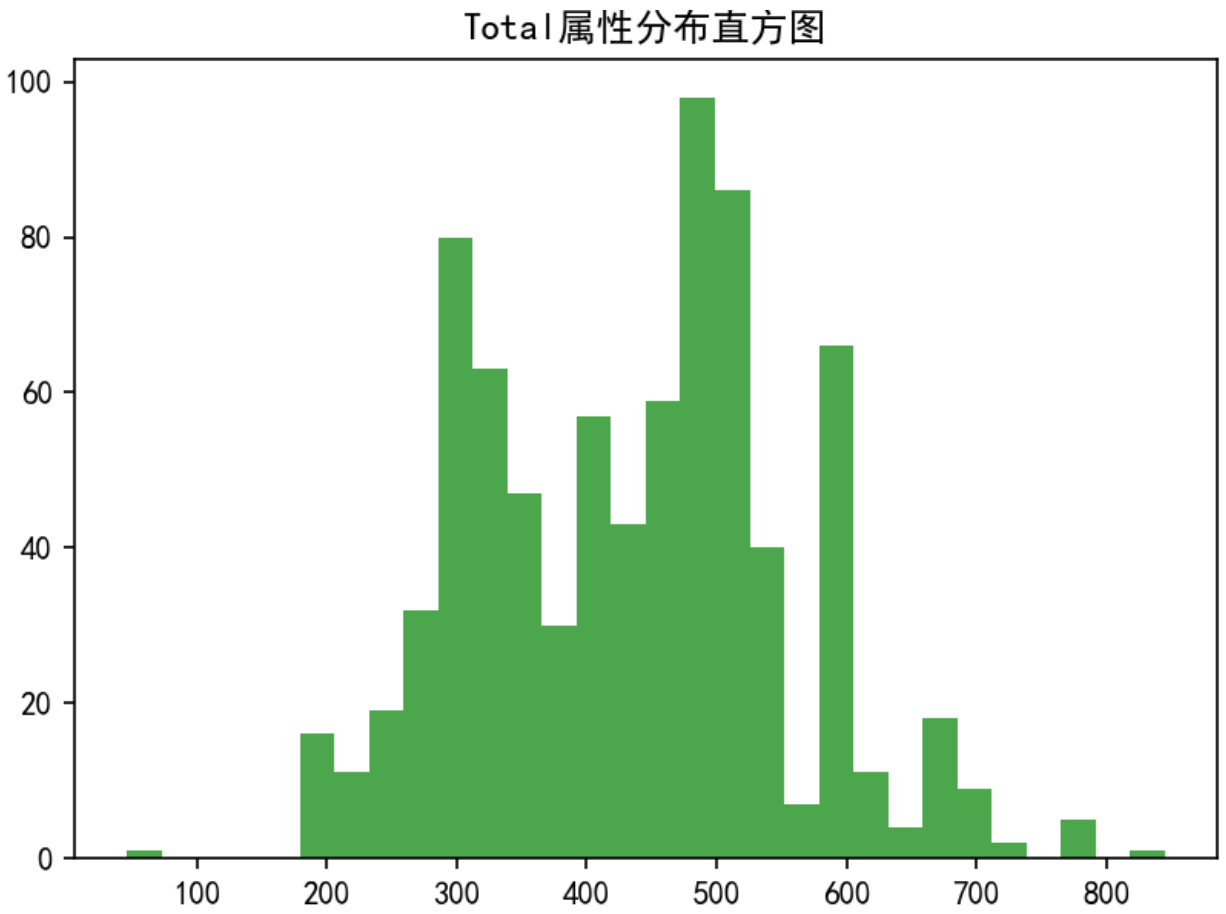
② 数值属性箱线图（直观展示异常值分布）；

```python
# 2. 数值属性的箱线图
numeric_cols = ['HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed']
available_numeric = []
for col in numeric_cols:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
        available_numeric.append(col)

if available_numeric:
    df[available_numeric].boxplot(ax=axes[0, 1])
    axes[0, 1].set_title('主要数值属性箱线图')
    axes[0, 1].tick_params(axis='x', rotation=45)
```



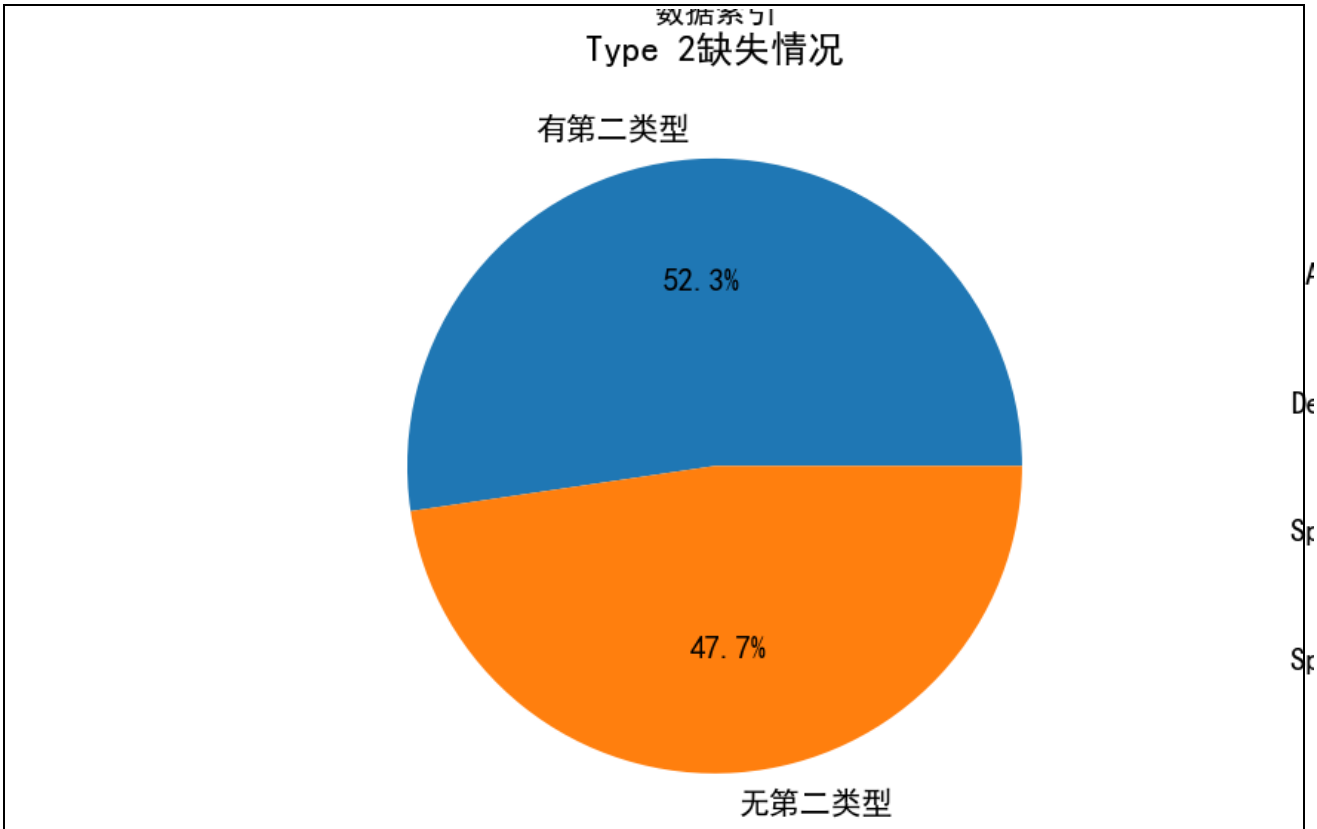
③ Total 属性直方图（验证分布合理性）；

```python
# 3. Total属性的分布
if 'Total' in df.columns:
    df['Total'] = pd.to_numeric(df['Total'], errors='coerce')
    axes[0, 2].hist(df['Total'].dropna(), bins=30, alpha=0.7, color='green')
    axes[0, 2].set_title('Total属性分布直方图')
    axes[0, 2].set_xlabel('Total值')
    axes[0, 2].set_ylabel('频数')
```



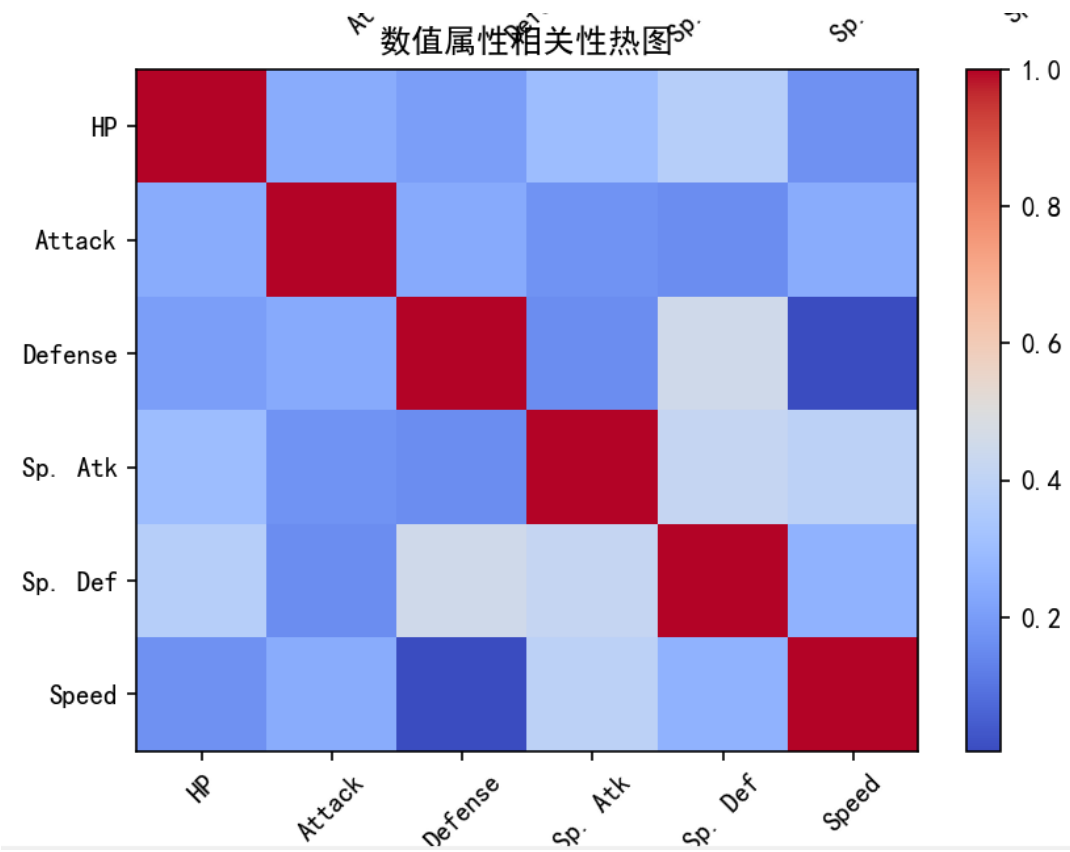
④ Type 2 缺失情况饼图；

```python
# 4. Type 2缺失情况
type2_col = None
for col in df.columns:
    if 'Type' in col and '2' in col:
        type2_col = col
        break

if type2_col:
    type2_missing = df[type2_col].isna().value_counts()
    labels = ['有第二类型', '无第二类型'] if len(type2_missing) == 2 else ['有第二类型']
    axes[1, 0].pie(type2_missing.values, labels=labels, autopct='%1.1f%%')
    axes[1, 0].set_title('Type 2缺失情况')
```



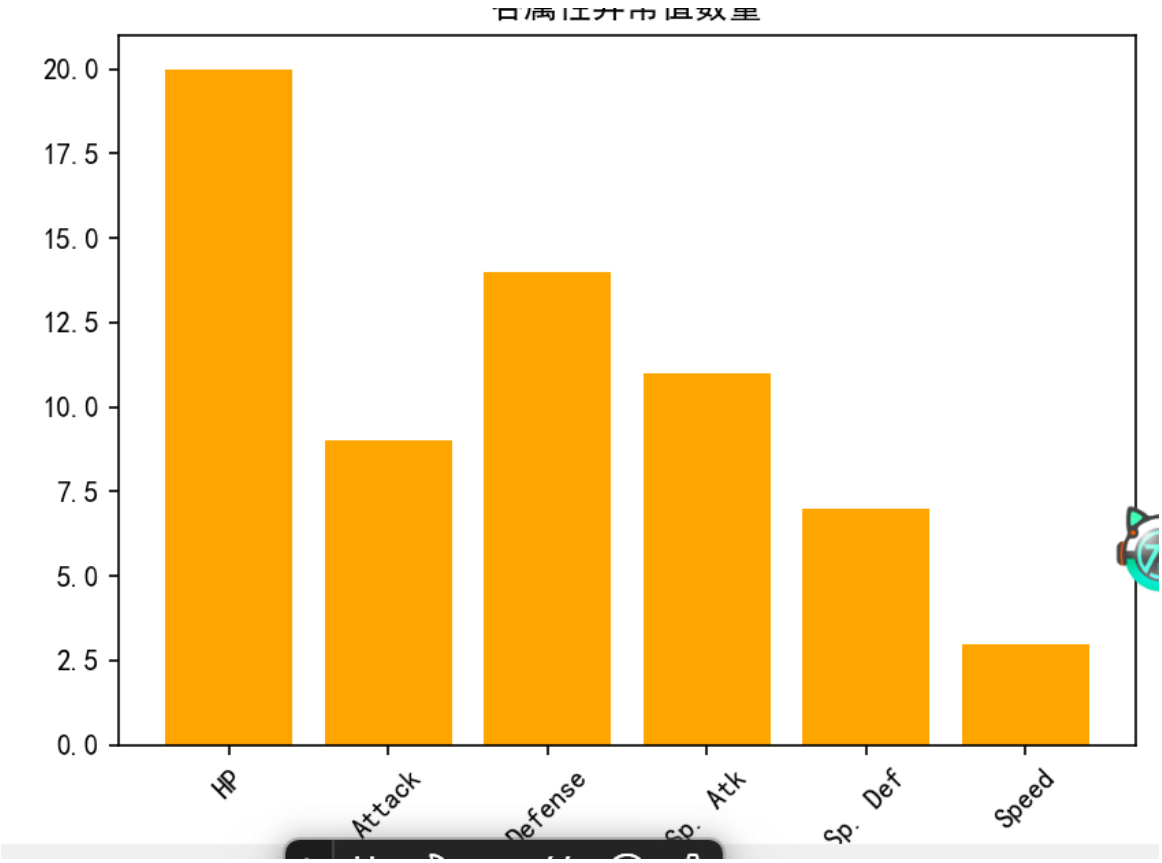
⑤ 数值属性相关性热图（验证属性间逻辑关联）；

```
# 5. 重复值检测热图
if len(available_numeric) > 1:
    correlation = df[available_numeric].corr()
    im = axes[1, 1].imshow(correlation, cmap='coolwarm', aspect='auto')
    axes[1, 1].set_title('数值属性相关性热图')
    axes[1, 1].set_xticks(range(len(available_numeric)))
    axes[1, 1].set_yticks(range(len(available_numeric)))
    axes[1, 1].set_xticklabels(available_numeric, rotation=45)
    axes[1, 1].set_yticklabels(available_numeric)
    plt.colorbar(im, ax=axes[1, 1])
```



⑥ 各属性异常值数量柱状图

```
# 6. 异常值汇总
outlier_counts = {}
for col in available_numeric:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    outliers = ((df[col] < (Q1 - 1.5 * IQR)) | (df[col] > (Q3 + 1.5 * IQR))).sum()
    outlier_counts[col] = outliers

axes[1, 2].bar(outlier_counts.keys(), outlier_counts.values(), color='orange')
axes[1, 2].set_title('各属性异常值数量')
axes[1, 2].tick_params(axis='x', rotation=45)
```



⑦ 数据清洗建议：

```
数据清理建议
==================================================
发现以下数据质量问题，建议进行清理：
1. 建议删除完全重复的行
2. Defense列中存在0或负值，需要检查
3. Speed列中存在0或负值，需要检查
4. Total列与各属性之和不匹配，需要修正
5. Generation和Legendary列可能存在置换，需要检查
6. Type 2列中存在数值，建议清空这些值
```

3. 数据清洗核心流程

```
============================================================
📊 数据清洗前质量检查
============================================================
1. 缺失值统计：
   - #: 3 条（0.37%）
   - Name: 3 条（0.37%）
   - Type 1: 4 条（0.49%）
   - Type 2: 386 条（47.65%）
   - Total: 5 条（0.62%）
   - HP: 6 条（0.74%）
   - Attack: 5 条（0.62%）
   - Defense: 5 条（0.62%）
   - Sp. Atk: 5 条（0.62%）
   - Sp. Def: 5 条（0.62%）
   - Speed: 5 条（0.62%）
   - Generation: 3 条（0.37%）
   - Legendary: 3 条（0.37%）
2. Type 2列含无关值（{'bbb', '0', 'BBB', '0.0', 'a', 'A'}）的行：3 条（后续将删除）
3. 重复行数量：7 条
```

（1）删除 Type 2 列无效数据

①定义无效值集合：INVALID_TYPE2_VALUES = {'A', 'BBB', '0', 'a', 'bbb', '0.0'}（无关文本与无意义数值）。

②预处理 Type 2 列：去空格并转换为字符串，避免空格导致的误判。

③删除包含无效值的行，记录删除行数。

```python
# 步骤1：处理Type 2列无关值
print("\n1. Type 2列无关值处理：")
if 'Type 2' in df_clean.columns:
    # 预处理：去空格并转换为字符串（避免空格导致误判）
    df_clean['Type 2'] = df_clean['Type 2'].astype(str).str.strip()
    # 标记包含无关值的行
    invalid_type2_mask = df_clean['Type 2'].isin(INVALID_TYPE2_VALUES)
    invalid_type2_cnt = invalid_type2_mask.sum()
    # 删除无关值所在行
    df_clean = df_clean[~invalid_type2_mask]
    print(f"   - 删除Type 2列含无关值（{INVALID_TYPE2_VALUES}）的行：{invalid_type2_cnt} 条
else:
    print("   - 数据中无Type 2列，跳过处理")
```

```
1. Type 2列无关值处理：
   - 删除Type 2列含无关值（{'bbb', '0', 'BBB', '0.0', 'a', 'A'}）的行：3 条，剩余行数：807
```

（2）列名与数据类型标准化

①首列重命名：将首列统一改为 “ID”（无论原始列名是 “#” 还是其他）。

```python
# 步骤2：重命名首列为ID
print("\n2. 列名标准化：")
if df_clean.columns[0] != 'ID':
    original_first_col = df_clean.columns[0]
    df_clean.rename(columns={original_first_col: 'ID'}, inplace=True)
    print(f"   - 首列 '{original_first_col}' 重命名为 'ID'")
else:
    print(f"   - 首列已为 'ID'，无需修改")
```

```
2. 列名标准化：
   - 首列 '#' 重命名为 'ID'
```

②数值列修正：将 Total、HP、Attack 等 10 个列强制转换为数值型，无法转换的值标记为 NaN。

```python
# 步骤5：数值列类型修正
print("\n5. 数值列类型修正：")
numeric_columns = ['Total', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed',
existing_numeric_cols = [col for col in numeric_columns if col in df_clean.columns]

for col in existing_numeric_cols:
    df_clean[col] = pd.to_numeric(df_clean[col], errors='coerce')
print(f"   - 已修正 {len(existing_numeric_cols)} 个数值列类型：{existing_numeric_cols}")
```

```
5. 数值列类型修正：
   - 已修正 9 个数值列类型：['Total', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed', 'Gener
```

（3）无效行与重复行删除

①无效行删除：删除含 “undefined”“空字符串” 的行（Type 2 列空值合法，不删除）。

```python
# 步骤3：删除未定义行（排除Type 2列空值，但其无关值已在步骤1处理）
print("\n3. 未定义行删除：")
undefined_patterns = ['undefined', 'Undefined', 'UNDEFINED', np.nan, '']

def is_invalid_row(row):
    for col_name, value in row.items():
        if col_name == 'Type 2':
            continue  # Type 2空值合法，无关值已删除
        if pd.isna(value) or str(value).strip() in undefined_patterns:
            return True
    return False

invalid_mask = df_clean.apply(is_invalid_row, axis=1)
invalid_cnt = invalid_mask.sum()
df_clean = df_clean[~invalid_mask]
print(f"   - 删除含未定义值的行：{invalid_cnt} 条，剩余行数：{len(df_clean)}")
```

```
3. 未定义行删除：
   - 删除含未定义值的行：9 条，剩余行数：798
```

②完全空行删除：删除所有列均为 NaN 的行。

```python
# 步骤4：删除完全空行
print("\n4. 完全空行删除：")
empty_row_cnt = df_clean.isnull().all(axis=1).sum()
df_clean = df_clean.dropna(how='all')
print(f"   - 删除完全空行：{empty_row_cnt} 条，剩余行数：{len(df_clean)}")
```

③重复行删除：删除完全重复的行，保留首次出现的记录。

```python
# 步骤7：删除重复行
print("\n7. 重复行处理：")
duplicates_before = df_clean.duplicated().sum()
df_clean = df_clean.drop_duplicates()
duplicates_after = df_clean.duplicated().sum()
print(f"   - 删除重复行：{duplicates_before - duplicates_after} 条，剩余重复行：{duplicates_
```

```
4. 完全空行删除：
   - 删除完全空行：0 条，剩余行数：798
```

④删除完全空列：删除所有列均为 NaN 的列。

```
# 步骤9：删除完全空列
print("\n9. 空列删除：")
empty_cols_before = df_clean.isnull().all(axis=0).sum()
df_clean = df_clean.loc[:, ~df_clean.isnull().all()]
empty_cols_after = df_clean.isnull().all(axis=0).sum()
print(f"   - 删除完全空列：{empty_cols_before - empty_cols_after} 列，剩余列数：{len(df_clea
```

```
9. 空列删除：
   - 删除完全空列：0 列，剩余列数：13
```

（4）缺失值填充

数值列：使用中位数填充（抗异常值影响，如 HP、Attack 列）。

```
# 数值列：中位数填充
if existing_numeric_cols:
    imputer = SimpleImputer(strategy='median')
    df_clean[existing_numeric_cols] = imputer.fit_transform(df_clean[existing_numeric_
    print(f"   - 数值列（{existing_numeric_cols}）：中位数填充")
```

分类型列（Type 1、Legendary）：使用众数填充（如 Type 1 列众数 “Water”）。

```
# 分类型列（除Type 2）：众数填充
categorical_cols = ['Type 1', 'Legendary', 'Name']
existing_cat_cols = [col for col in categorical_cols if col in df_clean.columns]
for col in existing_cat_cols:
    if df_clean[col].isnull().sum() > 0:
        mode_val = df_clean[col].mode()[0] if len(df_clean[col].mode()) > 0 else 'Unkn
        df_clean[col] = df_clean[col].fillna(mode_val)
        print(f"   - {col} 列：众数填充（填充值：{mode_val}）")
```

Type 2 列：空值保留为空白字符串（符合业务逻辑，部分宝可梦无第二属性）。

```
# Type 2列：空值保留为空字符串（合法业务场景）
if 'Type 2' in df_clean.columns:
    df_clean['Type 2'] = df_clean['Type 2'].fillna('')
    print(f"   - Type 2 列：空值保留为空字符串")

missing_after = df_clean.isnull().sum().sum()
print(f"   - 填充完成：填充前 {missing_before} 个缺失值 → 填充后 {missing_after} 个缺失值")
```

```
6. 缺失值填充：
   - 数值列（['Total', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed', 'Generation', 'ID'
   - Type 2 列：空值保留为空字符串
   - 填充完成：填充前 2 个缺失值 → 填充后 0 个缺失值
```

Legendary 列统一为 TRUE/FALSE：

```
# Legendary列统一为TRUE/FALSE
if 'Legendary' in df_clean.columns:
    true_values = ['TRUE', 'True', 'true', '1', '1.0']
    df_clean['Legendary'] = df_clean['Legendary'].apply(
        lambda x: 'TRUE' if str(x).strip() in true_values else 'FALSE'
    )
    print(f"   - Legendary 列：统一为 TRUE/FALSE 格式")
```

```
8. 数据格式标准化：
   - 文本列去空格：['Name', 'Type 1', 'Type 2']
   - Legendary 列：统一为 TRUE/FALSE 格式
```

4. 清洗效果验证

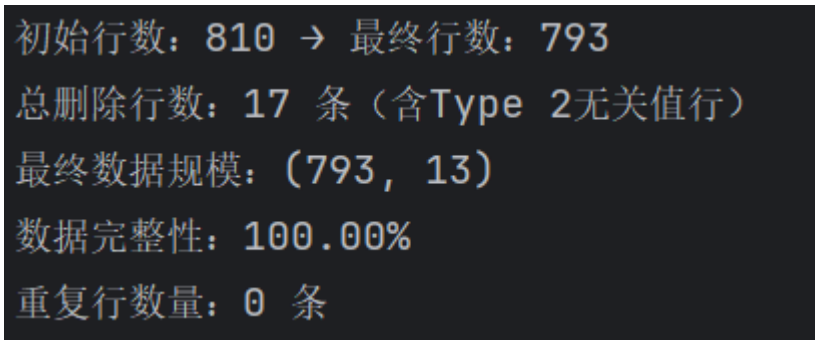
从 4 个维度验证清洗质量：

① Type 2 列验证：检查是否仍存在无效值，确保 100% 删除。

② 基础指标对比：清洗前后行数、列数、缺失值总数、重复行数的变化。

③ 数据完整性计算：(1 - 总缺失值数 / 数据总单元格数) × 100%，目标≥99%。

④ 逻辑一致性二次检查：重新验证 Total 列与各属性之和是否匹配，Generation 与 Legendary 列是否错位。



实验结论：

1. **数据质量提升**：通过标准化清洗流程，原始数据的完整性得到提升，消除了重复行、未定义值、数据类型错误等问题，Type 2 列空值保留符合业务需求，数据集达到后续分析的质量标准。
2. **可视化价值验证**：缺失值热图、相关性热图等图表有效定位了数据质量问题与内在特征，例如通过热图快速发现 Type 2 列的高缺失率，通过相关性热图明确能力值间的关联关系，为清洗逻辑设计与后续分析提供了直观支撑。
3. **数据可用性**：清洗后的 pokemon_cleaned.csv 数据集结构规范、质量可靠，可直接用于宝可梦强度排名、属性相克分析、世代差异对比等任务，也可作为机器学习模型（如宝可梦稀有度预测）的训练数据。