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

<table>
<tr><td>学号：202300130003</td><td>姓名：肖皓天</td><td>班级：数据 23</td></tr>
<tr><td colspan="3">实验题目：实验 2. 数据质量实践</td></tr>
<tr><td colspan="2">实验学时：32</td><td>实验日期：2025/9/23</td></tr>
<tr><td colspan="3">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</td></tr>
</table>

流程描述：

1. 导入数据

```python
import pandas as pd
import numpy as np

path = r"D:\0_aaa大三上\大数据分析实践\Pokemon.csv"
try:
    df = pd.read_csv(path, encoding="latin1", sep=None, engine="python")
except FileNotFoundError:
    raise SystemExit(f"错误：文件路径不正确或文件不存在 -> {path}")

def find_active_column(dataframe, name_options):
    active_col = next((col for col in name_options if col in dataframe.columns), None)

    if active_col is None:
        raise KeyError(f"数据中未找到任何指定的列。尝试过的名称：{name_options}")
    return active_col

COLUMN_MAPPING_RULES = {
    "COL_T1": ["Type 1", "type1", "Type1"],
    "COL_T2": ["Type 2", "type2", "Type2"],
    "COL_ATK": ["Attack", "attack", "ATK"],
    "COL_GEN": ["Generation", "generation", "Gen"],
    "COL_LEG": ["Legendary", "legendary", "isLegendary"]
}

print("正在规范化列名...")
for var_name, candidates in COLUMN_MAPPING_RULES.items():
    try:
        resolved_name = find_active_column(df, candidates)
        globals()[var_name] = resolved_name
        print(f"  变量 '{var_name}' -> 被赋值为列名 '{resolved_name}'")
    except KeyError as e:
        print(f"警告：{e}")

print("\n数据维度:", df.shape)
print("数据预览:")
df.head()
```

数据维度：(810, 13)
数据预览：

	#	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

2. 最后两行数据无意义，可直接删去

```python
before = len(df)
if before >= 2:
    df = df.iloc[:-2].copy()
else:
    df = df.iloc[0:0].copy()
print(f"Rows: {before} -> {len(df)}")
```

```
Rows: 810 -> 808
```

3. 对数据进行统计，发现 type2 存在异常的数值取值，可清空

```python
def analyze_series_quality(input_series: pd.Series, known_categories: Set[str]) -> Dict[str, pd.Series]:

    series = input_series.fillna('[Missing]').astype(str).str.strip()
    series = series.replace('', '[Blank]')

    full_dist = series.value_counts()

    valid_set = known_categories.union({'[Missing]', '[Blank]'})
    anomalous_mask = ~series.isin(valid_set)
    anomalous_dist = series[anomalous_mask].value_counts()

    singleton_dist = full_dist[full_dist == 1]

    return {
        "full_distribution": full_dist,
        "anomalous_distribution": anomalous_dist,
        "singleton_distribution": singleton_dist
    }


valid_types = {
    'Bug','Dark','Dragon','Electric','Fairy','Fighting','Fire','Flying','Ghost',
    'Grass','Ground','Ice','Normal','Poison','Psychic','Rock','Steel','Water'
}


analysis_results = analyze_series_quality(df[COL_T2], valid_types)

def print_analysis_report(results: Dict[str, pd.Series], col_name: str):
    print(f"--- 分析报告: '{col_name}' ---")

    print("\n【全里统计】")
    print(results['full_distribution'])

    print("\n【异常值统计】")
    anomalies = results['anomalous_distribution']
    print(anomalies if not anomalies.empty else "（无异常值）")

    print("\n【孤立值统计】")
    singletons = results['singleton_distribution']
    print(singletons if not singletons.empty else "（无孤立值）")
    print("-"*(20 + len(col_name)))

print_analysis_report(analysis_results, COL_T2)
```

```
【全里统计】
Type 2
[Missing]    384
Flying        98
Poison        37
Ground        35
Psychic       33
Fighting      26
Grass         25
Fairy         23
Steel         22
Dark          20
Dragon        18
Rock          14
Ghost         14
Water         14
Ice           14
Fire          12
Electric       6
Normal         4
Bug            3
undefined      2
A              1
273            1
0              1
BBB            1
Name: count, dtype: int64

【异常值统计】
Type 2
undefined    2
0            1
273          1
A            1
BBB          1
Name: count, dtype: int64

【孤立值统计】
Type 2
A      1
273    1
0      1
BBB    1
Name: count, dtype: int64
--------------------------
```

```python
import numpy as np
import pandas as pd
import unicodedata

values_to_exclude = {'0', '273', 'a', 'bbb'}

def is_value_banned(original_value, ban_list):

    if pd.isna(original_value):
        return False

    normalized_str = unicodedata.normalize('NFKC', str(original_value))

    cleaned_str = normalized_str.strip()
    folded_str = cleaned_str.casefold()

    if folded_str and (folded_str in ban_list):
        return True

    return False

mask_to_nullify = df[COL_T2].apply(lambda x: is_value_banned(x, values_to_exclude))


num_hits = mask_to_nullify.sum()
print(f"检测到 {num_hits} 条需要处理的记录。")

if num_hits > 0:
    print("将被置空的原始值示例:")
    print(df.loc[mask_to_nullify, COL_T2].head(10).tolist())


df[COL_T2] = df[COL_T2].where(~mask_to_nullify, np.nan)

print("\n处理完成。df 中的相关值已被置为 NaN。")
```

```
检测到 4 条需要处理的记录。
将被置空的原始值示例:
['0', '273', 'A', 'BBB']

处理完成。df 中的相关值已被置为 NaN。
```

4. 数据集中存在重复值

```python
import pandas as pd
import numpy as np
from typing import Dict, Any, Tuple

def analyze_and_clean_duplicates(dataframe: pd.DataFrame) -> Tuple[pd.DataFrame, Dict[str, Any]]:

    grouped = dataframe.groupby(list(dataframe.columns))
    group_indices = grouped.groups

    duplicate_row_indices = []
    unique_duplicate_first_indices = []
    total_redundant_count = 0

    for group_key, indices in group_indices.items():
        if len(indices) > 1:
            duplicate_row_indices.extend(indices)
            unique_duplicate_first_indices.append(indices[0])
            total_redundant_count += (len(indices) - 1)

    report = {
        "redundant_count": total_redundant_count,
        "all_duplicates_df": dataframe.loc[duplicate_row_indices].sort_index(),
        "unique_duplicates_df": dataframe.loc[unique_duplicate_first_indices].sort_index()
    }


    indices_to_keep = [indices[0] for indices in group_indices.values()]
    cleaned_df = dataframe.loc[indices_to_keep].sort_index()

    return cleaned_df, report

df, analysis_report = analyze_and_clean_duplicates(df)

print(f"去重条数: {analysis_report['redundant_count']}; ")

print("\n>>> 重复的行（包含所有重复出现）:")
print(analysis_report['all_duplicates_df'])

print("\n>>> 重复的行（每组只保留一次）:")
print(analysis_report['unique_duplicates_df'])

print("\nDataFrame 已通过函数完成去重。")
```

```
去重条数：6；

>>> 重复的行（包含所有重复出现）：
             #       Name     Type 1     Type 2      Total         HP  \
14          11    Metapod        Bug        NaN        205         50
15          11    Metapod        Bug        NaN        205         50
21          17  Pidgeotto     Normal     Flying        349         63
23          17  Pidgeotto     Normal     Flying        349         63
184        168    Ariados        Bug     Poison        390         70
185        168    Ariados        Bug     Poison        390         70
186        168    Ariados        Bug     Poison        390         70
187        168    Ariados        Bug     Poison        390         70
806  undefined  undefined  undefined  undefined  undefined  undefined
807  undefined  undefined  undefined  undefined  undefined  undefined

        Attack    Defense    Sp. Atk    Sp. Def      Speed Generation  \
14          20         55         25         25         30          1
15          20         55         25         25         30          1
21          60         55         50         50         71          1
23          60         55         50         50         71          1
184         90         70         60         60         40          2
185         90         70         60         60         40          2
186         90         70         60         60         40          2
187         90         70         60         60         40          2
806  undefined  undefined  undefined  undefined  undefined  undefined
807  undefined  undefined  undefined  undefined  undefined  undefined

     Legendary
14       FALSE
15       FALSE
21       FALSE
23       FALSE
184      FALSE
185      FALSE
186      FALSE
187      FALSE
806  undefined
807  undefined


>>> 重复的行（每组只保留一次）：
             #       Name     Type 1     Type 2      Total         HP  \
14          11    Metapod        Bug        NaN        205         50
21          17  Pidgeotto     Normal     Flying        349         63
184        168    Ariados        Bug     Poison        390         70
806  undefined  undefined  undefined  undefined  undefined  undefined

        Attack    Defense    Sp. Atk    Sp. Def      Speed Generation  \
14          20         55         25         25         30          1
21          60         55         50         50         71          1
184         90         70         60         60         40          2
806  undefined  undefined  undefined  undefined  undefined  undefined

     Legendary
14       FALSE
21       FALSE
184      FALSE
806  undefined

DataFrame 已通过函数完成去重。
```

5. Attack 属性存在过高的异常值

```python
def parse_attack_value(value):

    if pd.isna(value):
        return np.nan

    s = str(value).strip()

    null_patterns = re.compile(r'^(NA|N/A|-|--|—|\s*)$', re.IGNORECASE)
    if null_patterns.match(s):
        return np.nan

    s_no_comma = s.replace(',', '')

    try:
        return float(s_no_comma)
    except (ValueError, TypeError):
        return np.nan

numeric_attack_series = df[COL_ATK].apply(parse_attack_value)

original_non_null = df[COL_ATK].notna().sum()
converted_non_null = numeric_attack_series.notna().sum()
parsing_failures = original_non_null - converted_non_null

print(f"[Analysis] 成功转换 {converted_non_null} 个有效数值，解析失败 {parsing_failures} 个。")

df[COL_ATK] = numeric_attack_series

if df[COL_ATK].notna().sum() == 0:
    print("[Warning] 列 '{COL_ATK}' 不包含任何有效数值，跳过异常值处理。")
else:
    Q1 = df[COL_ATK].quantile(0.25)
    Q3 = df[COL_ATK].quantile(0.75)
    IQR = Q3 - Q1
    upper_fence = Q3 + 1.5 * IQR

    outliers_count = (df[COL_ATK] > upper_fence).sum()

    df[COL_ATK].clip(upper=upper_fence, inplace=True)

    print(f"[Clean] 在列 '{COL_ATK}' 中截断了 {outliers_count} 个高端异常值。")
    print(f"  - 截断阈值 (Q3 + 1.5*IQR) = {upper_fence:.2f}")

    print("\n处理后列的描述性统计:")
    display(df[COL_ATK].describe())
```

```
[Analysis] 成功转换 800 个有效数值，解析失败 1 个。
[Clean] 在列 'Attack' 中截断了 9 个高端异常值。
  - 截断阈值 (Q3 + 1.5*IQR) = 167.50

处理后列的描述性统计:
count    800.000000
mean      79.110625
std       32.445670
min        5.000000
25%       55.000000
50%       75.000000
75%      100.000000
max      167.500000
Name: Attack, dtype: float64
```

6. 有两条数据的 generation 与 Legendary 属性被置换

```python
def check_for_swap(row, gen_col, leg_col):

    gen_val = row[gen_col]
    leg_val = row[leg_col]

    gen_is_bool_like = isinstance(gen_val, str) and gen_val.strip().lower() in ['true', 'false']

    leg_is_numeric_like = pd.to_numeric(leg_val, errors='coerce') is not np.nan

    return gen_is_bool_like and leg_is_numeric_like

swap_mask = df.apply(lambda row: check_for_swap(row, COL_GEN, COL_LEG), axis=1)

swap_count = swap_mask.sum()
print(f"通过行级应用函数检测到 {swap_count} 个可能被调换的记录。")


if swap_count > 0:
    print("正在执行数据调换...")
    rows_to_swap_idx = df.index[swap_mask]

    temp_storage = df.loc[rows_to_swap_idx, COL_GEN].copy()
    df.loc[rows_to_swap_idx, COL_GEN] = df.loc[rows_to_swap_idx, COL_LEG]
    df.loc[rows_to_swap_idx, COL_LEG] = temp_storage
    print("调换完成。")

df[COL_GEN] = pd.to_numeric(df[COL_GEN], errors='coerce').astype("Int64")

leg_series_lower = df[COL_LEG].astype(str).str.strip().str.lower()
conditions = [
    leg_series_lower == 'true',
    leg_series_lower == 'false'
]
choices = [True, False]
df[COL_LEG] = np.select(conditions, choices, default=None) # 不匹配的设为None(Pandas会转为NaN)

print("\n处理后数据预览:")
df[[COL_GEN, COL_LEG]].head(10)
```

通过行级应用函数检测到 2 个可能被调换的记录。
正在执行数据调换...
调换完成。

处理后数据预览：

	Generation	Legendary
0	1	False
1	1	False
2	1	False
3	1	False
4	1	False
5	1	False
6	1	False
7	1	False
8	1	False
9	1	False

结论分析与体会：

通过本次实验，我了解并实践了对已有数据进行预处理清洗的操作，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。