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：202300300198	姓名：朱宣玮	班级：数据 23
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：

Python 3.11

Jupyter Notebook

数据集：

Pokeman Dataset：721 Pokemon, including their number, name, first and second type, and basic stats: HP, Attack, Defense, Special Attack, Special Defense, and Speed

http://storage.amesholland.xyz/Pokemon.csv

#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generatior	Legendary
1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	FALSE
2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	FALSE
3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	FALSE
3	VenusaurM	Grass	Poison	625	80	100	123	122	120	80	1	FALSE
4	Charmand	Fire		309	39	52	43	60	50	65	1	FALSE
5	Charmelec	Fire		405	58	64	58	80	65	80	1	FALSE
6	Charizard	Fire	Flying	534	78	84	78	109	85	100	1	FALSE
6	CharizardM	Fire	Dragon	634	78	130	111	130	85	100	1	FALSE
6	CharizardM	Fire	Flying	634	78	104	78	159	115	100	1	FALSE
7	Squirtle	Water		314	44	840	65	50	64	43	1	FALSE
8	Wartortle	Water		405	59	63	80	65	80	58	1	FALSE
9	Blastoise	Water		530	79	83	100	85	105	78	FALSE	1
9	BlastoiseM	Water		630	79	103	120	135	115	78	1	FALSE
10	Caterpie	Bug		195	45	30	35	20	20	45	1	FALSE
11	Metapod	Bug		205	50	20	55	25	25	30	1	FALSE
11	Metapod	Bug		205	50	20	55	25	25	30	1	FALSE
12	Butterfree	Bug	Flying	395	60	45	50	90	80	70	1	FALSE
13	Weedle	Bug	Poison	195		35	30	20	20	50	1	FALSE
14	Kakuna	Bug	Poison	205	45	25	50	25	25	35	1	FALSE
15	Beedrill	Bug	Poison	395	65	90	40	45	80	75	1	FALSE
15	BeedrillMe	Bug	Poison	495	65	150	40	15	80	145	1	FALSE
17	Pidgeotto	Normal	Flying	349	63	60	55	50	50	71	1	FALSE
16	Pidgey	Normal	Flying	251	40	45	40	35	35	56	1	FALSE
17	Pidgeotto	Normal	Flying	349	63	60	55	50	50	71	1	FALSE
18	Pidgeot	Normal	Flying	479	83	80	75	70	70	101	1	FALSE
18	PidgeotMe	Normal	Flying	579	83	80	80	135	80	121	1	FALSE
19	Rattata	Normal		253	30	56	35	25	35	72	1	FALSE
20	Raticate	Normal		413	55	81	60	50	70	97	1	FALSE
21	Spearow	Normal	Flying	262	40	60	30	31	31	70	1	FALSE
22	Fearow	Normal	Flying	442	65	90	65	61	61	100	1	FALSE
23	Ekans	Poison		288	35	60	44	40	54	55	1	FALSE
24	Arbok	Poison		438	60	85	69	65	79	80	1	FALSE

实验步骤与结果：

1. 库的导入与数据的读入，使用 pandas 库处理数据，matplotlib 绘图

```
import pandas as pd
import matplotlib.pyplot as plt
```

[779] ✓ 0.0s Python

```
df = pd.read_csv("Pokemon.csv", encoding="latin1")
df.head()
```

[780] ✓ 0.0s Python

	#	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

```
df.info()
```

[755] ✓ 0.0s Python

```
<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
```

2. 查看最后 5 行，发现最后 4 行均为 underfined 或 NaN，无意义，删除

```
df.tail(5)
```

[783] ✓ 0.0s Python

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
805	721	Volcanion	Fire	Water	600	80	110	120	130	90	70	6	TRUE
806	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined
807	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined
808	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
809	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

```
# 最后4行无意义，删除
df = df.iloc[:-4, :]
```

[784] ✓ 0.0s Python

3. 查找每一列都是 NaN 的行并删除

```
# 删除全是NaN的行
df[df.isna().all(axis=1)]
```

[785] ✓ 0.0s Python

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
408	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
df = df.dropna(how='all')
```

[786] ✓ 0.0s Python

4. 查找 Total, HP, Attack, Defense, Sp. Atk, Sp. Def, Speed, Generation, Legendary 列中存在 undefined 的行，并删除

```
# 先枚举第4~12列的列名（假设从0开始计数）
cols_to_check = df.columns[4:13]  # 第4列索引3，第12列索引11

# 找出含有 'undefined' 的行
undefined_rows = df[df[cols_to_check].apply(lambda row: row.str.contains('undefined', na=False)).any(axis=1)]
print("Rows containing 'undefined':")
print(undefined_rows)

# 删除这些行
df = df.drop(index=undefined_rows.index)
```

[787] ✓ 0.0s Python

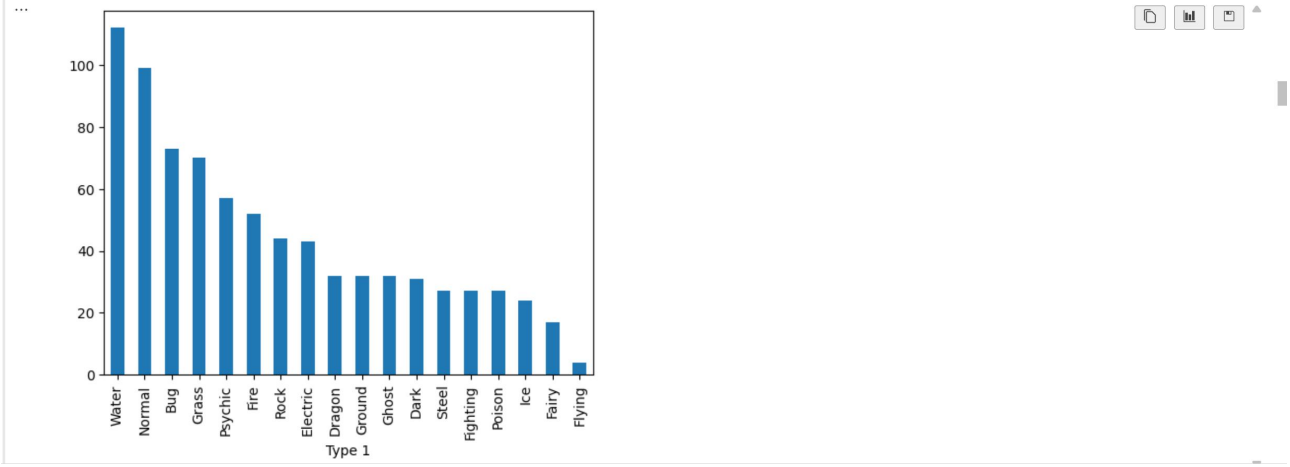
```
Rows containing 'undefined':
       #       Name    Type 1  Type 2 Total  HP Attack Defense Sp. Atk  \
771  695  Heliolisk  Electric  Normal   481  62     55      52     109   

     Sp. Def Speed Generation Legendary
771       94   109  undefined     FALSE
```

5. 统计 Type 1 列的值绘制条形图，均正常

```
df["Type 1"].value_counts().plot(kind='bar')
```

[788] ✓ 0.1s Python

```
<Axes: xlabel='Type 1'>
```

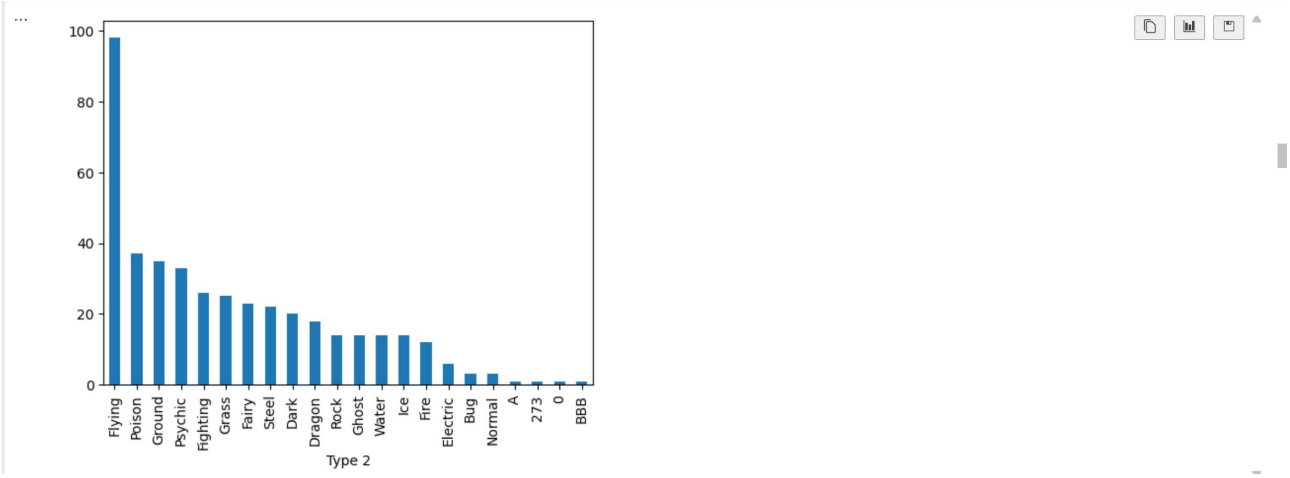


6. 统计 Type 2 列的值绘制条形图，发现异常值

```
df["Type 2"].value_counts().plot(kind='bar')
```

[789] ✓ 0.1s Python

```
<Axes: xlabel='Type 2'>
```



异常值为 A, 273, 0, BBB，删除这些值所在的行

```
# 删除异常取值所在的行
invalid_values = ['A', '273', '0', 'BBB']
invalid_rows = df[df["Type 2"].isin(invalid_values)]
print("Invalid rows:")
print(invalid_rows)

df = df.drop(index=invalid_rows.index)
```

[790] ✓ 0.0s Python

```
Invalid rows:
       #                Name    Type 1 Type 2 Total   HP Attack Defense  \
34    27           Sandshrew    Ground      0   300   50     75      85
39    32     Nidoranâ♀?Poison       NaN    273    46   57     40      40
117  107          Hitmonchan  Fighting      A   455   50    105      79
428  382  KyogrePrimal Kyogre     Water    BBB   770  100    150      90

    Sp. Atk Sp. Def Speed Generation Legendary
34       20      30    40          1     FALSE
39       40      50     1      FALSE       NaN
117      35     110    76          1     FALSE
428     180     160    90          3      TRUE
```

7. 删除重复的行

```
df[df.duplicated()]
```
[792] ✓ 0.0s Python

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
15	11	Metapod	Bug	NaN	205	50	20	55	25	25	30	1	FALSE
23	17	Pidgeotto	Normal	Flying	349	63	60	55	50	50	71	1	FALSE
185	168	Ariados	Bug	Poison	390	70	90	70	60	60	40	2	FALSE
186	168	Ariados	Bug	Poison	390	70	90	70	60	60	40	2	FALSE
187	168	Ariados	Bug	Poison	390	70	90	70	60	60	40	2	FALSE

```
df = df.drop_duplicates()
```
[793] ✓ 0.0s Python

8. 将 Total, HP, Attack, Defense, Sp. Atk, Sp. Def, Speed 列转换为数值类型，无法转换（原始数据集中缺失或格式不对）的设置为 NaN
这些属性在宝可梦中应为正整数，所以至少有 1 个值小于等于 0 或为 NaN 的行应删除。

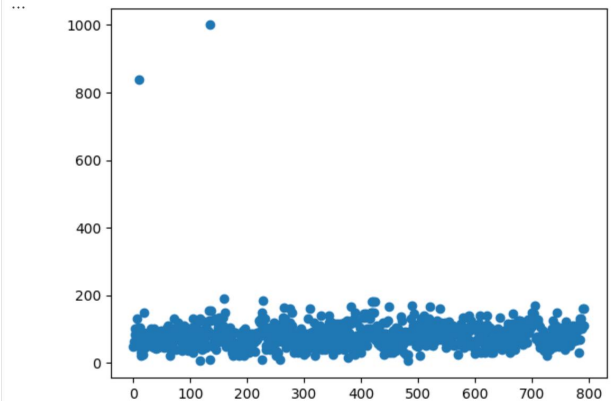
```
# 获取索引 4~10 的列名
cols_to_convert = df.columns[4:11]
print(cols_to_convert)
# 转换为数值，无法转换的设置为 NaN
for col in cols_to_convert:
    df[col] = pd.to_numeric(df[col], errors="coerce")

# 找出这些列中至少有一个值 <= 0 或为 NaN 的行
invalid_rows = df[(df[cols_to_convert] <= 0).any(axis=1) | df[cols_to_convert].isna().any(axis=1)]
print("Rows with values <= 0 or NaN in numeric stats:")
print(invalid_rows)

# 删除这些行
df = df.drop(index=invalid_rows.index)
```
[16] ✓ 0.0s Python

```
Index(['Total', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed'], dtype='object')
Rows with values <= 0 or NaN in numeric stats:
       #      Name Type 1  Type 2  Total    HP  Attack  Defense  Sp. Atk  \
17    13    Weedle    Bug  Poison    195   NaN    35.0       30     20.0
349  315   Roselia  Grass  Poison    400  50.0    60.0      -10    100.0
620  554  Darumaka   Fire     NaN    315  70.0    90.0       45     15.0

     Sp. Def  Speed Generation Legendary
17      20.0     50          1     FALSE
349     80.0     65          3     FALSE
620     45.0    -50          5     FALSE
```

9. Attack 属性存在过高的异常值

```
plt.scatter(range(df.shape[0]), df.iloc[:,6])
```
[17] ✓ 0.0s Python

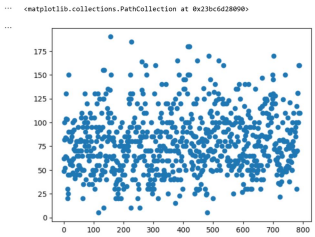
```
<matplotlib.collections.PathCollection at 0x23bc4858750>
```



删除 Attack 大于 400 的异常值

```
df = df[df["Attack"] <= 400]
```
[18] ✓ 0.0s Python

删除后再次绘图



10. 在宝可梦中，Total 的值应为 HP, Attack, Defense, Sp. Atk, Sp. Def, Speed 的值之和。数据集中大部分行也符合这个关系。
因此，我们筛选不符合 Total 为基础属性之和的行并删除。

```python
# 需要检查的列
cols_stats = ['HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed']

# 找出 Total 不等于 6 个基础属性之和的行
mismatch_rows = df[df['Total'] != df[cols_stats].sum(axis=1)]
print("Rows where Total does not match sum of stats:")
print(mismatch_rows)

# 删除这些行
df = df.drop(index=mismatch_rows.index)
```

[20] ✓ 0.0s Python

```
Rows where Total does not match sum of stats:
       #                   Name   Type 1  Type 2  Total     HP  Attack  \
38    31              Nidoqueen   Poison  Ground    505   90.0    92.0
48    41                  Zubat   Poison  Flying    845   40.0    45.0
62    55                Golduck    Water     NaN    500  120.0    82.0
362  326                Grumpig  Psychic     NaN    470   80.0    45.0
463  412                  Burmy      Bug     NaN    224   40.0    29.0
766  690                 Skrelp   Poison   Water    320   50.0    60.0
768  692              Clauncher    Water     NaN    330   50.1    53.0
786  710  PumpkabooAverage Size    Ghost   Grass    435   49.0    66.0
793  711    GourgeistSuper Size    Ghost   Grass    494  185.0   100.0
795  713                Avalugg      Ice     NaN    714   95.0   117.0

     Defense  Sp. Atk  Sp. Def  Speed Generation Legendary
38        87   750.00    85.00     76          1     FALSE
48        35    30.00    40.00     55          1     FALSE
62        78    95.00    80.00     85          1     FALSE
362       65    90.00   110.11     80          3     FALSE
463      450    29.00    45.00     36          4     FALSE
766       60    60.21    60.00     30          6     FALSE
768       62    58.00    63.00     44          6     FALSE
786       70    44.00    55.00     51          6     FALSE
793      122    58.00    75.00     54          6     FALSE
795      184    44.00    46.00     28          6     FALSE
```

观察到仅有 10 行不符合该关系，删除这些行。
11. 有两条数据的 Generation 与 Legendary 属性被置换
先统计两列的值

```python
df["Generation"].value_counts()
```

[71] ✓ 0.0s Python

```
Generation
5        164
3        157
1        155
4        120
2        106
6         76
FALSE      2
Name: count, dtype: int64
```

```python
df["Legendary"].value_counts()
```

[72] ✓ 0.0s Python

```
Legendary
FALSE     709
TRUE       64
0           3
1           1
Poison      1
Ground      1
Name: count, dtype: int64
```

Generation 的合法值为 1-6，Legendary 合法值为 TRUE 或者 FALSE
筛选 Generation 不在 1-6，且 Legendary 为数字的行，并交换两列的值

```python
# 找出置换行：Generation 不在 1~6，且 Legendary 是数字字符串
swapped_rows = df[ (~df["Generation"].isin([str(i) for i in range(1,7)])) &
                   (df["Legendary"].str.isdigit()) ]

print("Swapped rows:")
print(swapped_rows)

# 交换两列
df.loc[swapped_rows.index, ["Generation", "Legendary"]] = df.loc[swapped_rows.index, ["Legendary", "Generation"]].values
```

[21] ✓ 0.0s Python

```
Swapped rows:
     #       Name    Type 1 Type 2  Total    HP  Attack  Defense  Sp. Atk  \
11   9  Blastoise     Water    NaN    530  79.0    83.0      100     85.0
32  25    Pikachu  Electric    NaN    320  35.0    55.0       40     50.0

    Sp. Def  Speed Generation Legendary
11    105.0     78      FALSE         1
32     50.0     90      FALSE         0
```

交换之后，仍有一些行的 Generation 与 Legendary 是不合理的，找到并删除。

```
df["Generation"].value_counts()
```

[74] ✓ 0.0s Python

```
Generation
5    164
3    157
1    156
4    120
2    106
6     76
0      1
Name: count, dtype: int64
```

```
df["Legendary"].value_counts()
```

[75] ✓ 0.0s Python

```
Legendary
FALSE     711
TRUE       64
0           2
Poison      1
Ground      1
Name: count, dtype: int64
```

删除 Generation 不在 1 - 6 的行

```
# 找出 Generation 不在 '1'-'6' 的行
invalid_gen = df[~df["Generation"].isin([str(i) for i in range(1,7)])]
print("Rows with invalid Generation:")
print(invalid_gen)

# 删除这些行
df = df.drop(index=invalid_gen.index)
```

[76] ✓ 0.0s Python

```
Rows with invalid Generation:
      #     Name    Type 1 Type 2  Total    HP  Attack  Defense  Sp. Atk  \
32  25  Pikachu  Electric    NaN    320  35.0    55.0       40     50.0

    Sp. Def  Speed Generation Legendary
32     50.0     90          0     FALSE
```

删除 Legendary 不为 'TRUE' 或 'FALSE' 的行

```
# 找出 Legendary 不为 'TRUE' 或 'FALSE' 的行
invalid_legendary = df[~df["Legendary"].isin(['TRUE','FALSE'])]
print("Rows with invalid Legendary:")
print(invalid_legendary)

# 删除这些行
df = df.drop(index=invalid_legendary.index)
```

[77] ✓ 0.0s Python

```
Rows with invalid Legendary:
       #                 Name   Type 1    Type 2  Total    HP  Attack  \
45    38            Ninetales     Fire       NaN    505  73.0    76.0
78    70           Weepinbell    Grass    Poison    390  65.0    90.0
115  105              Marowak   Ground       NaN    425  60.0    80.0
130  119              Seaking    Water       NaN    450  80.0    92.0
533  475  GalladeMega Gallade  Psychic  Fighting    618  68.0   165.0

     Defense  Sp. Atk  Sp. Def  Speed Generation Legendary
45        75     81.0    100.0    100          1         0
78        50     85.0     45.0     55          1    Poison
115      110     50.0     80.0     45          1    Ground
130       65     65.0     80.0     68          1         0
533       95     65.0    115.0    110          4       NaN
```

12. 查看清洗后的数据

```
df.info()
df.describe(include="all")
```

[79] ✓ 0.0s Python

```
<class 'pandas.core.frame.DataFrame'>
Index: 774 entries, 0 to 805
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   #           774 non-null    object
 1   Name        774 non-null    object
 2   Type 1      774 non-null    object
 3   Type 2      404 non-null    object
 4   Total       774 non-null    int64
 5   HP          774 non-null    float64
 6   Attack      774 non-null    float64
 7   Defense     774 non-null    int64
 8   Sp. Atk     774 non-null    float64
 9   Sp. Def     774 non-null    float64
 10  Speed       774 non-null    int64
 11  Generation  774 non-null    object
 12  Legendary   774 non-null    object
dtypes: float64(4), int64(3), object(6)
memory usage: 84.7+ KB
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
count	774	774	774	404	774.000000	774.000000	774.000000	774.000000	774.00000	774.000000	774.000000	774	774
unique	699	774	18	18	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6	2
top	479	Bulbasaur	Water	Flying	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5	FALSE
freq	6	1	107	96	NaN	NaN	NaN	NaN	NaN	NaN	NaN	164	710
mean	NaN	NaN	NaN	NaN	435.974160	69.483204	79.042636	73.908269	73.21447	71.937984	68.387597	NaN	NaN
std	NaN	NaN	NaN	NaN	119.608998	25.708619	32.465796	31.145622	32.58295	27.724411	29.188279	NaN	NaN
min	NaN	NaN	NaN	NaN	180.000000	1.000000	5.000000	5.000000	10.00000	20.000000	5.000000	NaN	NaN
25%	NaN	NaN	NaN	NaN	330.000000	50.000000	55.000000	50.000000	50.00000	50.000000	45.000000	NaN	NaN
50%	NaN	NaN	NaN	NaN	450.000000	65.000000	75.000000	70.000000	65.00000	70.000000	65.000000	NaN	NaN
75%	NaN	NaN	NaN	NaN	515.000000	80.000000	100.000000	90.000000	95.00000	90.000000	90.000000	NaN	NaN
max	NaN	NaN	NaN	NaN	780.000000	255.000000	190.000000	230.000000	194.00000	230.000000	180.000000	NaN	NaN

共 774 行

结论分析与体会：

结论分析

本次实验通过 Pandas 库对宝可梦数据集进行了系统的数据清洗与异常值处理。

主要包括删除无意义行、处理缺失值、剔除异常属性值及重复数据，规范了数值类型并确保属性间逻辑关系一致（如 Total 与基础属性之和）。

体会

掌握了 dropna()、isin()、to_numeric()、apply() 等函数的使用，并理解了针对不同异常情况的处理方法。

异常值和类型错误若不处理，会影响分析结果的准确性与可解释性。

通过数据预处理可以提升数据质量，为后续分析与可视化提供基础。