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：202300130113	姓名：丁正旸	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。

实验过程与内容：
数据集地址：http://storage.amesholland.xyz/Pokemon.csv
数据集存在的部分问题：
•最后两行数据无意义，可直接删去

In [21]:

```
df
```

Out[21]:

	#	Name	Type 1	Type 2	Total	HP	Attack	D
0	1	Bulbasaur	Grass	Poison	318	45	49	
1	2	Ivysaur	Grass	Poison	405	60	62	
2	3	Venusaur	Grass	Poison	525	80	82	
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	
4	4	Charmander	Fire	NaN	309	39	52	
...	...	...	...	...	...	...	...	
805	721	Volcanion	Fire	Water	600	80	110	
806	undefined	undefined	undefined	undefined	undefined	undefined	undefined	unc
807	undefined	undefined	undefined	undefined	undefined	undefined	undefined	unc
808	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
809	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

810 rows × 13 columns

In [56]:

```
# 观察到最后4行无有效数据，直接删除（保留前805行）
df_clean = df.iloc[:-4].copy()  # 切片删除最后4行，copy避免警告
print("\n删除无意义行后数据集形状：", df_clean.shape)  # 应输出(806, 13)
```

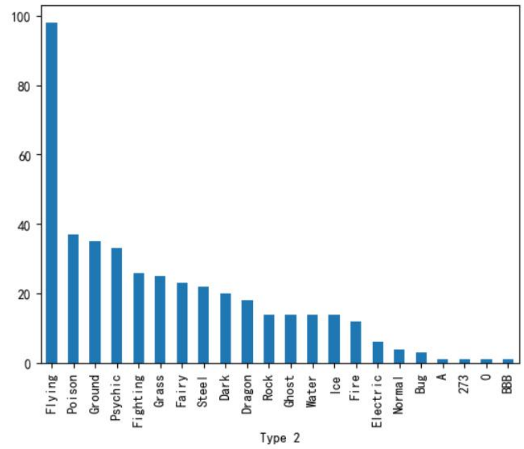
删除无意义行后数据集形状： (806, 13)

•type2 存在异常的数值取值，可清空

In [57]:

```python
# 查看Type2列的所有取值及计数
df_clean['Type 2'].value_counts().plot(kind='bar')
```

Out[57]:
<Axes: xlabel='Type 2'>

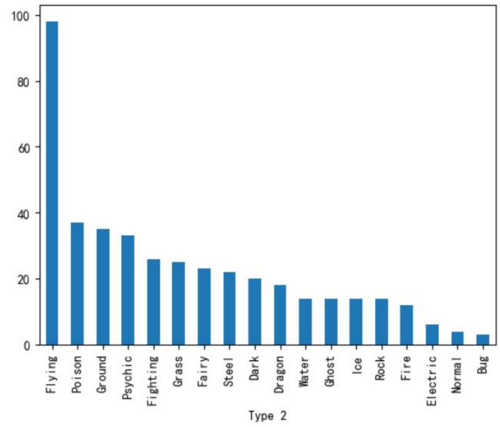


In [58]:

```python
# 发现A, 273, 0, BBB异常值，删除异常行
df_clean = df_clean[~df_clean['Type 2'].isin(['A', '273', '0', 'BBB'])]

# 重新查看Type2列的所有取值及计数
df_clean['Type 2'].value_counts().plot(kind='bar')
```

Out[58]:
<Axes: xlabel='Type 2'>



•数据集中存在重复值

In [59]:

```python
# 检查完全重复的行
df_clean[df_clean.duplicated()]
```

Out[59]:

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

In [60]:

```python
# 删除重复行
df_clean = df_clean.drop_duplicates()
print("\n删除重复值后数据集形状：", df_clean.shape)

# 验证无重复值
print(f"处理后重复行数量：{len(df_clean[df_clean.duplicated()])}")  # 应输出0
```

```
删除重复值后数据集形状： (797, 13)
处理后重复行数量：0
```

•Attack 属性存在过高的异常值

In [61]:

```
# 查看Attack列的描述性统计
print("Attack属性描述性统计：")
print(df_clean['Attack'].describe())
```

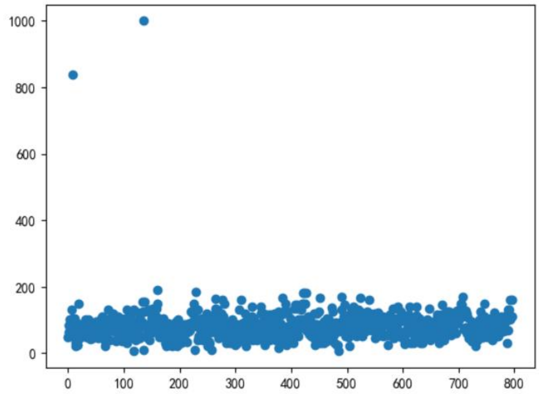
```
Attack属性描述性统计：
count      796
unique     117
top        100
freq        39
Name: Attack, dtype: object
```

In [62]:

```
# 1. 将 Attack 列转为数值类型（非数值值→NaN）
df_clean['Attack'] = pd.to_numeric(df_clean['Attack'], errors='coerce')
#查看分布
plt.scatter(range(0,df_clean.shape[0]),df_clean.iloc[:,6])
```

Out[62]:

```
&lt;matplotlib.collections.PathCollection at 0x2145c27cb90&gt;
```



In [65]:

```
# 2. 通过3σ原则识别异常值
#计算 3σ 异常值
attack_mean = df_clean['Attack'].mean()
attack_std = df_clean['Attack'].std()
upper_bound = attack_mean + 3 * attack_std
lower_bound = attack_mean - 3 * attack_std

#过滤异常值（保留在 [lower_bound, upper_bound] 范围内的数据）
df_clean = df_clean[(df_clean['Attack'] >= lower_bound) & (df_clean['Attack'] <= upper_bound

print(f"Attack 列均值：{attack_mean:.2f}")
print(f"Attack 列标准差：{attack_std:.2f}")
print(f"异常值边界：[{lower_bound:.2f}, {upper_bound:.2f}]")
print(f"清理后的数据量：{len(df_clean)} 行")
```

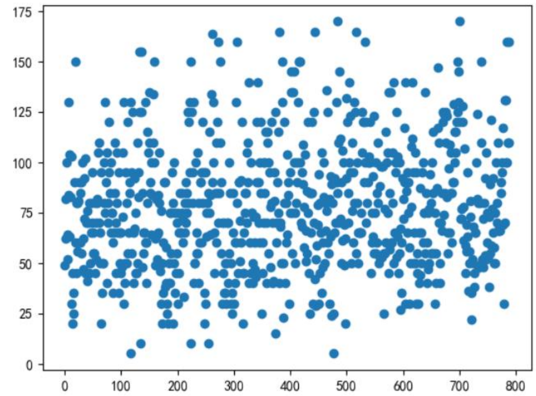
```
Attack 列均值：78.92
Attack 列标准差：32.43
异常值边界：[-18.37, 176.22]
清理后的数据量：789 行
```

In [66]:

```
#查看清理异常值后的分布
plt.scatter(range(0,df_clean.shape[0]),df_clean.iloc[:,6])
```

Out[66]:

```
&lt;matplotlib.collections.PathCollection at 0x2145ab91ee0&gt;
```



- 有两条数据的 generation 与 Legendary 属性被置换

In [67]:

```
# 查看Generation和Legendary列的取值类型和分布
print("Generation列取值类型：", df_clean['Generation'].dtype)
print("Legendary列取值类型：", df_clean['Legendary'].dtype)
print("\nGeneration列取值分布：")
print(df_clean['Generation'].value_counts())
print("\nLegendary列取值分布：")
print(df_clean['Legendary'].value_counts())
```

```
Generation列取值类型： object
Legendary列取值类型： object

Generation列取值分布：
Generation
5            165
1            158
3            156
4            121
2            105
6             81
FALSE          2
undefined      1
Name: count, dtype: int64

Legendary列取值分布：
Legendary
FALSE      722
TRUE        60
0            3
1            1
Poison       1
Ground       1
Name: count, dtype: int64
```

In [70]:

```
# 定义有效取值集合
valid_generation = {'1', '2', '3', '4', '5', '6'}  # Generation列有效数字（字符串形式）
valid_legendary = {'TRUE', 'FALSE', True, False}   # Legendary列有效布尔值（含字符串和布尔类

# 1. 筛选 Generation 列非数字的完整行
generation_invalid_full = df_clean[~df_clean['Generation'].isin(valid_generation)]

# 2. 筛选 Legendary 列非 True/False 的完整行
legendary_invalid_full = df_clean[~df_clean['Legendary'].isin(valid_legendary)]

# 3. 合并所有异常行（去重，避免同一行被重复显示）
all_invalid_full = pd.concat([generation_invalid_full, legendary_invalid_full]).drop_duplicat
```

In [71]:

```
display(all_invalid_full)
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	G
11	9	Blastoise	Water	NaN	530	79	83.0	100	85	105	78	
32	25	Pikachu	Electric	NaN	320	35	55.0	40	50	50	90	
771	695	Heliolisk	Electric	Normal	481	62	55.0	52	109	94	109	
45	38	Ninetales	Fire	NaN	505	73	76.0	75	81	100	100	
78	70	Weepinbell	Grass	Poison	390	65	90.0	50	85	45	55	
115	105	Marowak	Ground	NaN	425	60	80.0	110	50	80	45	
130	119	Seaking	Water	NaN	450	80	92.0	65	65	80	68	
533	475	GalladeMega Gallade	Psychic	Fighting	618	68	165.0	95	65	115	110	

In [73]:

```
#发现有两行的Generation与Legendary属性被置换，两行Legendary的属性被误填了其他属性的值
#1.互换行号为11和32的行的 Generation 和 Legendary 属性
# 保存原始值
gen_11 = df_clean.loc[11, 'Generation']
leg_11 = df_clean.loc[11, 'Legendary']
gen_32 = df_clean.loc[32, 'Generation']
leg_32 = df_clean.loc[32, 'Legendary']

# 互换值
df_clean.loc[11, 'Generation'] = leg_32
df_clean.loc[11, 'Legendary'] = gen_32
df_clean.loc[32, 'Generation'] = leg_11
df_clean.loc[32, 'Legendary'] = gen_11

# 2. 将索引为78、115的行的 Legendary 属性设为 undefined
df_clean.loc[78, 'Legendary'] = 'undefined'
df_clean.loc[115, 'Legendary'] = 'undefined'

# 验证结果
print("修改后索引=11、32、78、115 的行：")
display(df_clean.loc[[11,32,78,115], ['Generation', 'Legendary']])
```

修改后索引=11、32、78、115 的行：

	Generation	Legendary
11	0	FALSE
32	1	FALSE
78	1	undefined
115	1	undefined

结论与体会：

本次实验通过数据清洗、异常值处理及可视化分析，有效提升了数据质量。针对分类列（如Generation、Legendary）的格式不统一和非预期值问题，采用有效值筛选、类型转换等方法完成数据规整；对数值列（如 Attack）运用 3σ 原则识别并剔除异常值，确保了数据的可靠性。可视化结果清晰呈现了关键指标的分布特征，为后续分析提供了直观支撑。

数据预处理是数据分析的核心基础，严谨的异常值检测和类型校验能避免后续分析结果失真。同时，灵活运用 Pandas 进行数据筛选、转换，结合 Matplotlib 可视化辅助判断，可显著提升分析效率。未来需更注重数据源头的规范性，减少清洗成本，同时优化异常值处理策略，兼顾数据完整性与准确性。