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：202300130166	姓名：朱亚宁	班级：23 数据
实验题目：数据质量实践		
实验学时：2	实验日期：2025.10.10	
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。		

实验步骤：

一、导入数据

In [3]:
```
import pandas as pd
from pandas import DataFrame
import numpy as np

primitive_data=pd.read_csv("Pokemon.csv",encoding="latin_1")
primitive_data
```

Out[3]:

	#	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
5	5	Charmeleon	Fire	NaN	405	58	64	58	80	65	80	1	FALSE
6	6	Charizard	Fire	Flying	534	78	84	78	109	85	100	1	FALSE
7	6	CharizardMega Charizard X	Fire	Dragon	634	78	130	111	130	85	100	1	FALSE
8	6	CharizardMega Charizard Y	Fire	Flying	634	78	104	78	159	115	100	1	FALSE
9	7	Squirtle	Water	NaN	314	44	840	65	50	64	43	1	FALSE
10	8	Wartortle	Water	NaN	405	59	63	80	65	80	58	1	FALSE
11	9	Blastoise	Water	NaN	530	79	83	100	85	105	78	FALSE	1
12	9	BlastoiseMega Blastoise	Water	NaN	630	79	103	120	135	115	78	1	FALSE
13	10	Caterpie	Bug	NaN	195	45	30	35	20	20	45	1	FALSE

二、删除无效行

In [4]:
```
#删除无效行
primitive_data_1 = primitive_data.iloc[:-4]
primitive_data_1
```

Out[4]:

	#	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
5	5	Charmeleon	Fire	NaN	405	58	64	58	80	65	80	1	FALSE
6	6	Charizard	Fire	Flying	534	78	84	78	109	85	100	1	FALSE
7	6	CharizardMega Charizard X	Fire	Dragon	634	78	130	111	130	85	100	1	FALSE
8	6	CharizardMega Charizard Y	Fire	Flying	634	78	104	78	159	115	100	1	FALSE
9	7	Squirtle	Water	NaN	314	44	840	65	50	64	43	1	FALSE
10	8	Wartortle	Water	NaN	405	59	63	80	65	80	58	1	FALSE
11	9	Blastoise	Water	NaN	530	79	83	100	85	105	78	FALSE	1

三、删除重复行

In [5]:
```
#列出重复行
primitive_data_1[primitive_data_1.duplicated()]
```

...

In [6]:
```
#删除重复行
primitive_data_1 = primitive_data_1.drop_duplicates()
primitive_data_1
```

...

四、对 NAN 值进行处理

1、第一列为 NAN 的行删除

```
In [5]: nan_rows_first_column = primitive_data_1[primitive_data_1.iloc[:, 0].isnull()]

        nan_rows_first_column
```

Out[5]:

	#	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

```
In [6]: primitive_data_1 = primitive_data_1.drop([408])
        primitive_data_1
```

2、第三、四列为 NAN 的填充“Unknown”

```
In [7]: nan_rows_first_column = primitive_data_1[primitive_data_1.iloc[:,2].isnull()]

        nan_rows_first_column
```

Out[7]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
39	32	Nidoranâ□?Poison	NaN	273	46	57	40	40	40	50	1	FALSE	NaN

```
In [8]: primitive_data_1.iloc[:,2] = primitive_data_1.iloc[:,2].fillna('Unknown')

In [9]: nan_rows_first_column = primitive_data_1[primitive_data_1.iloc[:,3].isnull()]

        nan_rows_first_column
```

...

```
In [10]: primitive_data_1.iloc[:,3] = primitive_data_1.iloc[:,3].fillna('Unknown')
```

3、第六列为 NAN 的填充平均值，由于第六列数据类型为 object，不利于计算平均值，所以先转化为 float 类型，填充平均值之后再转化为 int 类型。

```
In [11]: nan_rows_first_column = primitive_data_1[primitive_data_1.iloc[:,5].isnull()]

         nan_rows_first_column
```

Out[11]:

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
17	13	Weedle	Bug	Poison	195	NaN	35	30	20	20	50	1	FALSE

```
In [12]: primitive_data_1.iloc[:,5]
```

...

```
In [13]: #转化为float类型
         primitive_data_1.iloc[:, 5] = primitive_data_1.iloc[:, 5].astype(float)

In [14]: #填充平均值
         primitive_data_1.iloc[:, 5] = primitive_data_1.iloc[:, 5].fillna(primitive_data_1.iloc[:, 5].mean())

In [15]: #确保填充完成
         nan_rows_first_column = primitive_data_1[primitive_data_1.iloc[:,5].isnull()]

         nan_rows_first_column
```

...

```
In [16]: #转化为int型
         primitive_data_1.iloc[:, 5] = primitive_data_1.iloc[:, 5].astype(int)
```

五、各列的数据类型均为 object，根据数据特征转化为相应的类型，比如 int，float 等等。

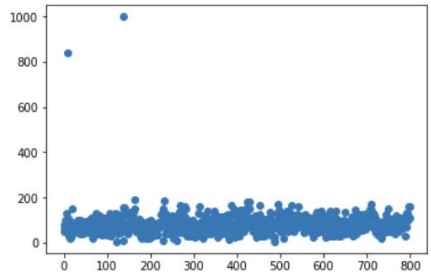
```
In [18]: primitive_data_1.iloc[:, 0] = primitive_data_1.iloc[:, 0].astype(int)

In [19]: primitive_data_1.iloc[:,4] = primitive_data_1.iloc[:, 4].astype(int)
         primitive_data_1.iloc[:, 6] = primitive_data_1.iloc[:, 6].astype(float)
         primitive_data_1.iloc[:, 7] = primitive_data_1.iloc[:, 7].astype(int)
         primitive_data_1.iloc[:, 8] = primitive_data_1.iloc[:, 8].astype(float)
         primitive_data_1.iloc[:,9] = primitive_data_1.iloc[:, 9].astype(float)
         primitive_data_1.iloc[:, 10] = primitive_data_1.iloc[:,10].astype(int)
```

六、清除异常值

1、对于第七列

（1）寻找异常值

In [20]:
```
import matplotlib.pyplot as plt

# 绘制散点图，清理过的数据
plt.scatter(range(0, primitive_data_1.shape[0]), primitive_data_1.iloc[:,6])


# 显示图形
plt.show()
```



In [21]:
```
#找出异常值
import numpy as np

Q1 = primitive_data_1.iloc[:,6].quantile(0.25)
Q3 = primitive_data_1.iloc[:,6].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = primitive_data_1[(primitive_data_1.iloc[:,6] < lower_bound) | (primitive_data_1.iloc[:,6] > upper_bound)]
print("异常值：")
print(outliers)
```

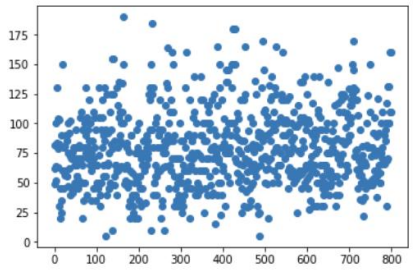
（2）异常值替换为平均值

In [22]:
```
#平均值替换
column_mean = primitive_data_1.iloc[:, 6][(primitive_data_1.index != 9) & (primitive_data_1.index != 140)].mean()

primitive_data_1.iloc[9, 6] = column_mean
```

In [23]:
```
#平均值替换
column_mean = primitive_data_1.iloc[:, 6][(primitive_data_1.index != 9) & (primitive_data_1.index != 140)].mean()
print(column_mean)
primitive_data_1.iloc[138, 6] = column_mean
primitive_data_1.iloc[138,6]
```

...

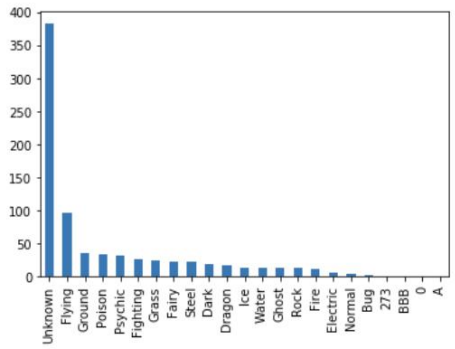
In [24]:
```
import matplotlib.pyplot as plt

# 绘制散点图，清理过的数据
plt.scatter(range(0, primitive_data_1.shape[0]), primitive_data_1.iloc[:,6])


# 显示图形
plt.show()
```



2、对于“Type 2”列，可视化找出异常值，并进行“Unknown”填充

In [25]:
```
#清空异常值
%matplotlib inline
primitive_data_1["Type 2"].value_counts().plot(kind='bar')
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26139186208>
```



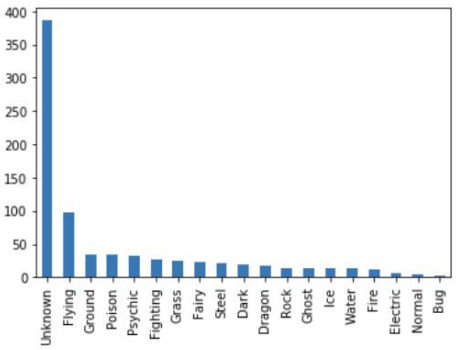
In [26]:
```
primitive_data_1.loc[primitive_data_1["Type 2"] == "273", primitive_data_1.columns[3]] = "Unknown"
```

In [27]:
```
primitive_data_1.loc[primitive_data_1["Type 2"] == "A", primitive_data_1.columns[3]] = "Unknown"
primitive_data_1.loc[primitive_data_1["Type 2"] == "BBB", primitive_data_1.columns[3]] = "Unknown"
primitive_data_1.loc[primitive_data_1["Type 2"] == "0", primitive_data_1.columns[3]] = "Unknown"
```

结果：

In [28]:
```
#清空异常值
%matplotlib inline
primitive_data_1["Type 2"].value_counts().plot(kind='bar')
```

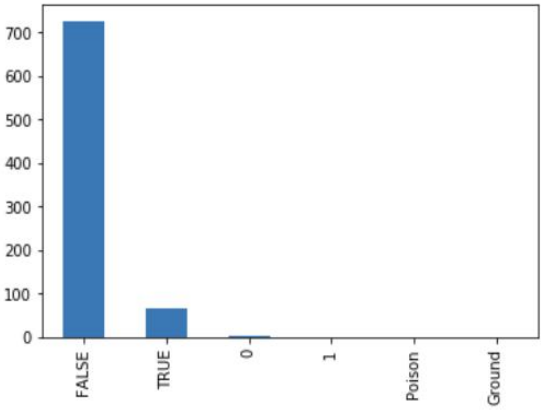
Out[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26139498278>
```



3、对于最后两列

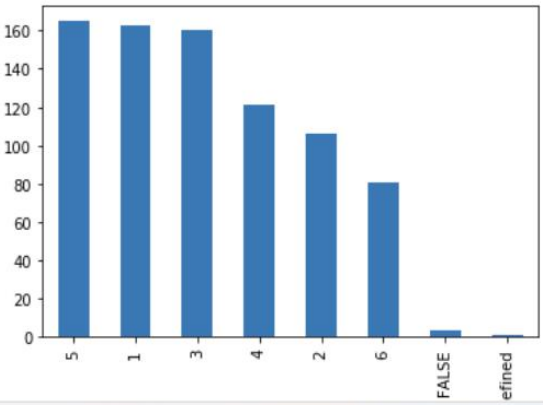
```
In [29]: #清空异常值
         %matplotlib inline
         primitive_data_1["Legendary"].value_counts().plot(kind='bar')
```

```
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x2613a9741d0>
```




```
In [30]: #清空异常值
         %matplotlib inline
         primitive_data_1["Generation"].value_counts().plot(kind='bar')
```

```
Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x261390e6048>
```



通过观察可以得出，存在这两列数据互换的情况。同时 Legendary 列为 0 的情况可能对应“FALSE”，用“FALSE”进行替换。“Legendary”列出现的异常值，替换为“FALSE”，因为从图标可以看到，FALSE 类较多，填充为 FALSE 对整体产生的影响较小。

```
In [31]: primitive_data_1.loc[primitive_data_1["Legendary"] == "0", primitive_data_1.columns[12]] = "FALSE"
```

```
In [32]: # 数字类型0/1和布尔类型的情况
         out2 = primitive_data_1[
             (primitive_data_1["Legendary"].isin(["0", "1"])) &
             (primitive_data_1["Generation"].isin(["TRUE", "FALSE"]))
         ]
         out2
```

```
Out[32]:
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
11	9	Blastoise	Water	Unknown	530	79	83.0	100	85.0	105.0	78	FALSE	1

```
In [33]: primitive_data_1.loc[primitive_data_1["Legendary"] == "1", primitive_data_1.columns[12]] = "FALSE"
         primitive_data_1.loc[primitive_data_1["Legendary"] == "1", primitive_data_1.columns[11]] = "1"
```

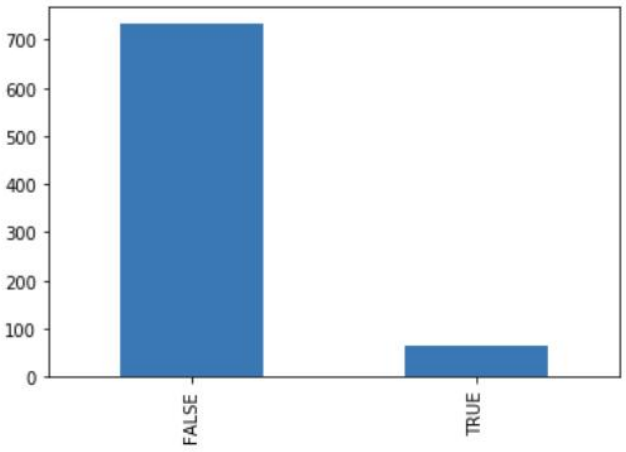
```
In [34]: primitive_data_1.loc[primitive_data_1["Legendary"] == "Poison", primitive_data_1.columns[12]] = "FALSE"
         primitive_data_1.loc[primitive_data_1["Legendary"] == "Ground", primitive_data_1.columns[12]] = "FALSE"
```

针对 Generation 列出现的 undefined，替换为“1”，考虑到对整体的比例影响较小。

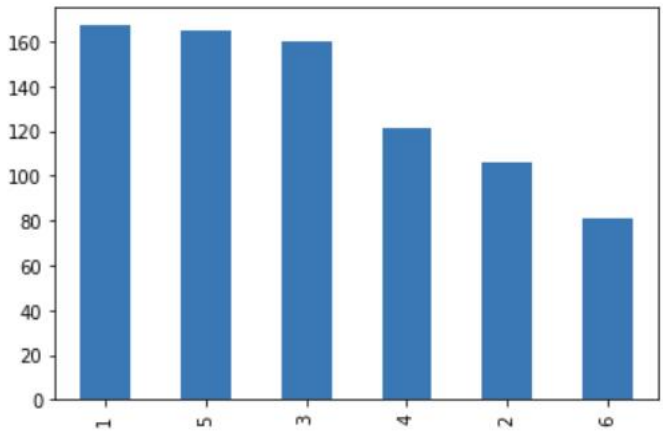
```
In [36]: primitive_data_1.loc[primitive_data_1["Generation"] == "undefined", primitive_data_1.columns[11]] = "1"
```

结果：

```
In [35]: #清空异常值
         %matplotlib inline
         primitive_data_1["Legendary"].value_counts().plot(kind='bar')
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x2613a933cc0>
```




```
In [44]: #清空异常值
         %matplotlib inline
         primitive_data_1["Generation"].value_counts().plot(kind='bar')
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x2613910ef28>
```



在进行完上述所有操作后，将最后两列的数据类型由 object 分别转化为 bool 和 int。

```
In [42]: primitive_data_1.iloc[:,11] = primitive_data_1.iloc[:, 11].astype(int)
         primitive_data_1.iloc[:,12] = primitive_data_1.iloc[:, 12].astype(bool)
```

结论分析与体会：

对于脏数据，发现数据集中存在属性值录入错误、格式不统一的情况，如部分宝可梦的身高、体重单位混乱，技能名称拼写错误等。通过编写正则表达式、建立映射表等方式，对错误数据进行修正，规范了数据格式，使数据更加标准化。

缺失数据处理方面，采用了不同策略。对于数值型数据，如宝可梦的 HP、攻击、防御等属性，若缺失值数量较少，使用均值、中位数进行填充；若缺失值较多，则结合宝可梦的类型、世代等特征，通过建立回归模型预测填充。对于分类数据，如特性、属性等，采用众数填充或根据已有数据的逻辑关系进行推断填充。经过处理，缺失数据对分析结果的影响大幅降低。

在异常值处理上，利用箱线图、散点图等可视化工具，识别出部分数值型属性的异常值，如个别宝可梦的某项能力值远高于正常范围。对于这些异常值，根据实际情况进行处理，若属于错误录入则修正，若反映特殊情况则保留并注明。

经过完整的数据预处理流程，数据集质量显著提升，为后续的数据分析，如宝可梦属性分布、不同世代宝可梦能力变化趋势等研究提供了可靠的数据基础。同时也验证了数据预处理清洗流程的科学性和有效性，其各个环节紧密相连，缺一不可。