C

2025 年 1 月 28 日

1 数据预处理

1.1 导入数据

[42]:

```
# 导入相关 package
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt
import chardet
```

[43]:

```
import os

# 设置环境变量 LOKY_MAX_CPU_COUNT
os.environ["LOKY_MAX_CPU_COUNT"] = "8"  # 使用 CPU 核心数
```

[44]:

```
# 定义一个函数，自动检测文件编码并读取文件
def read_csv_with_detected_encoding(file_path):
    # 检测文件编码
    with open(file_path, 'rb') as f:
        result = chardet.detect(f.read())
        encoding = result['encoding']
        print(f"检测到文件 {file_path} 的编码格式为: {encoding}")
    # 使用检测到的编码读取文件
    return pd.read_csv(file_path, encoding=encoding)

# 读取 data_dictionary.csv 文件
csv_content =␣
 ↪read_csv_with_detected_encoding('2025_Problem_C_Data\\data_dictionary.csv')
```

```python
print("data_dictionary.csv 数据预览：")
print(csv_content.head())

# 读取 summerOly_medal_counts.csv 文件
medal_counts = read_csv_with_detected_encoding('2025_Problem_C_Data\\summerOly_medal_counts.csv')
print("\nsummerOly_medal_counts.csv 数据预览：")
print(medal_counts.head())

# 读取 summerOly_hosts.csv 文件
olympic_hosts = read_csv_with_detected_encoding('2025_Problem_C_Data\\summerOly_hosts.csv')
print("\nsummerOly_hosts.csv 数据预览：")
print(olympic_hosts.head())

# 读取 summerOly_programs.csv 文件
olympic_programs = read_csv_with_detected_encoding('2025_Problem_C_Data\\summerOly_programs.csv')
print("\nsummerOly_programs.csv 数据预览：")
print(olympic_programs.head())

# 读取 summerOly_athletes.csv 文件
olympic_athletes = read_csv_with_detected_encoding('2025_Problem_C_Data\\summerOly_athletes.csv')
print("\nsummerOly_athletes.csv 数据预览：")
print(olympic_athletes.head())
```

```
检测到文件 2025_Problem_C_Data\data_dictionary.csv 的编码格式为：Windows-1252
data_dictionary.csv 数据预览：
  summerOly_medal_counts.csv                                     Unnamed: 1  \
0                  variables                                    explanation
1                       Rank      Rank of country based on total medals won
2                        NOC  Name of country as recorded for that Olympics
```

```
3                       Gold       Number of Gold medals the country earned
4                     Silver     Number of Silver medals the country earned

      Unnamed: 2
0        example
1           1, 2
2  China, France
3        0, 1, 2
4        0, 1, 2
检测到文件 2025_Problem_C_Data\summerOly_medal_counts.csv 的编码格式为：utf-8

summerOly_medal_counts.csv 数据预览:
   Rank            NOC  Gold  Silver  Bronze  Total  Year
0     1  United States    11       7       2     20  1896
1     2         Greece    10      18      19     47  1896
2     3        Germany     6       5       2     13  1896
3     4         France     5       4       2     11  1896
4     5  Great Britain     2       3       2      7  1896
检测到文件 2025_Problem_C_Data\summerOly_hosts.csv 的编码格式为：UTF-8-SIG

summerOly_hosts.csv 数据预览:
   Year                       Host
0  1896             Athens, Greece
1  1900              Paris, France
2  1904   St. Louis, United States
3  1908     London, United Kingdom
4  1912          Stockholm, Sweden
检测到文件 2025_Problem_C_Data\summerOly_programs.csv 的编码格式为：Windows-1252

summerOly_programs.csv 数据预览:
      Sport         Discipline Code Sports Governing Body 1896 1900 1904  \
0  Aquatics  Artistic Swimming  SWA        World Aquatics    0    0    0
1  Aquatics             Diving  DIV        World Aquatics    0    0    2
2  Aquatics  Marathon Swimming  OWS        World Aquatics    0    0    0
3  Aquatics           Swimming  SWM        World Aquatics    4    7    9
4  Aquatics         Water Polo  WPO        World Aquatics    0    1    1
```

```
   1906*  1908 1912  …  1988 1992  1996  2000  2004  2008  2012  2016  2020  \
0      0     0    0  …     2    2   1.0   2.0   2.0   2.0   2.0   2.0   2.0   
1      1     2    4  …     4    4   4.0   8.0   8.0   8.0   8.0   8.0   8.0   
2      0     0    0  …     0    0   0.0   0.0   0.0   2.0   2.0   2.0   2.0   
3      4     6    9  …    31   31  32.0  32.0  32.0  32.0  32.0  32.0  35.0   
4      0     1    1  …     1    1   1.0   2.0   2.0   2.0   2.0   2.0   2.0   

   2024  
0   2.0  
1   8.0  
2   2.0  
3  35.0  
4   2.0  

[5 rows x 35 columns]
检测到文件 2025_Problem_C_Data\summerOly_athletes.csv 的编码格式为：utf-8

summerOly_athletes.csv 数据预览:
                    Name Sex            Team  NOC  Year         City  \
0              A Dijiang   M           China  CHN  1992    Barcelona   
1               A Lamusi   M           China  CHN  2012       London   
2            Gunnar Aaby   M         Denmark  DEN  1920    Antwerpen   
3            Edgar Aabye   M  Denmark/Sweden  DEN  1900        Paris   
4  Cornelia (-strannood)   F     Netherlands  NED  1932  Los Angeles   

        Sport                           Event     Medal  
0  Basketball     Basketball Men's Basketball  No medal  
1        Judo    Judo Men's Extra-Lightweight  No medal  
2    Football         Football Men's Football  No medal  
3  Tug-Of-War     Tug-Of-War Men's Tug-Of-War      Gold  
4   Athletics  Athletics Women's 100 metres  No medal  
```

1.2 数据清洗

1.2.1 缺失值检查

[45]:

```python
# 1. 缺失值检查
def check_missing_values(file_path):
    """
    检查 CSV 文件中的缺失值。

    参数:
        file_path (str): CSV 文件的路径。

    返回:
        None, 但会打印缺失值的相关信息。
    """
    try:
        # 尝试读取 CSV 文件
        data = pd.read_csv(file_path, encoding='utf-8')
    except UnicodeDecodeError:
        data = pd.read_csv(file_path, encoding='ISO-8859-1')

    print(file_path)

    # 检查每列的缺失值数量
    missing_values_per_column = data.isnull().sum()
    print("每列的缺失值数量: ")
    print(missing_values_per_column)

    # 检查整个数据框的总缺失值数量
    total_missing_values = missing_values_per_column.sum()
    print("整个数据框的总缺失值数量: ", total_missing_values)

    # 检查是否有任何缺失值
    has_missing_values = data.isnull().values.any()
    print("数据框中是否存在缺失值: ", has_missing_values)
    print("\n")
    # 如果有缺失值，输出包含缺失值的行
    if has_missing_values:
        print("\n包含缺失值的行：")
        print(data[data.isnull().any(axis=1)])

content_name = ['2025_Problem_C_Data\\summerOly_medal_counts.csv',␣
 ↪'2025_Problem_C_Data\\summerOly_hosts.csv',␣
 ↪'2025_Problem_C_Data\\summerOly_programs.csv',␣
 ↪'2025_Problem_C_Data\\summerOly_athletes.csv']
for i in content_name:
    check_missing_values(i)
```

```
2025_Problem_C_Data\summerOly_medal_counts.csv
每列的缺失值数量:
Rank      0
NOC       0
Gold      0
Silver    0
Bronze    0
Total     0
Year      0
dtype: int64
整个数据框的总缺失值数量： 0
数据框中是否存在缺失值： False



2025_Problem_C_Data\summerOly_hosts.csv
每列的缺失值数量:
Year    0
Host    0
dtype: int64
整个数据框的总缺失值数量： 0
数据框中是否存在缺失值： False



2025_Problem_C_Data\summerOly_programs.csv
每列的缺失值数量:
```

```
Sport                    0
Discipline               2
Code                     0
Sports Governing Body    0
1896                     0
1900                     0
1904                     0
1906*                    0
1908                     0
1912                     0
1920                     0
1924                     0
1928                     2
1932                     2
1936                     2
1948                     2
1952                     2
1956                     2
1960                     2
1964                     2
1968                     2
1972                     2
1976                     2
1980                     2
1984                     2
1988                     3
1992                     2
1996                     2
2000                     2
2004                     2
2008                     2
2012                     2
2016                     2
2020                     2
2024                     2
dtype: int64
整个数据框的总缺失值数量： 49
```

```
数据框中是否存在缺失值： True



包含缺失值的行：
                Sport     Discipline Code Sports Governing Body 1896 1900  \
12      Basque Pelota  Basque Pelota  PEL                  FIPV    0    1
44  Modern Pentathlon            NaN  MPN                  UIPM    0    0
65  Water Motorsports            NaN  PBT                   UIM    0
69            Skating         Figure  FSK                   ISU    0    0
70         Ice Hockey     Ice Hockey  IHO                  IIHF    0    0

   1904  1906*  1908 1912  …  1988 1992 1996 2000 2004 2008  2012 2016  \
12    0      0     0    0  …   NaN       0.0  0.0  0.0  0.0   0.0  0.0
44    0      0     0    1  …     2    2  1.0  2.0  2.0  2.0   2.0  2.0
65    0      0     3    0  …     0    0  0.0  0.0  0.0  0.0   0.0  0.0
69    0      0     4    0  …   NaN  NaN  NaN  NaN  NaN  NaN   NaN  NaN
70    0      0     0    0  …   NaN  NaN  NaN  NaN  NaN  NaN   NaN  NaN

    2020 2024
12   0.0  0.0
44   2.0  2.0
65   0.0  0.0
69   NaN  NaN
70   NaN  NaN

[5 rows x 35 columns]
2025_Problem_C_Data\summerOly_athletes.csv
每列的缺失值数量：
Name      0
Sex       0
Team      0
NOC       0
Year      0
City      0
Sport     0
Event     0
```

```
Medal    0
dtype: int64
整个数据框的总缺失值数量： 0
数据框中是否存在缺失值： False
```

1.2.2 补全 summerOly_programs.csv 中的缺失值

[46]:

```python
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
import re
import os

# 确保保存结果的目录存在
os.makedirs('Generated', exist_ok=True)

data = olympic_programs.copy()

# 3. 检查缺失值
#print(" 每列的缺失值数量： ")
#print(data.isnull().sum())

# 4. 填充 Discipline 列的缺失值
data['Discipline'] = data['Discipline'].fillna(data['Sport'])

# 5. 准备年份列的数据
years = [col for col in data.columns if col.isdigit() or col.endswith('*')]

# 6. 将数据从宽格式转换为长格式
data_long = data.melt(id_vars=['Sport', 'Discipline', 'Code', 'Sports Governing␣
  ↪Body'],
                      value_vars=years,
                      var_name='Year',
                      value_name='Events')

# 7. 将年份列转换为数值
data_long['Year'] = data_long['Year'].str.replace('*', '').astype(int)

# 8. 清理 Events 列中的非数值字符
def clean_events(value):
    if isinstance(value, str):
        # 移除非数值字符
        cleaned_value = re.sub(r'[^0-9]', '', value)
        return float(cleaned_value) if cleaned_value.isdigit() else np.nan
    return value

data_long['Events'] = data_long['Events'].apply(clean_events)

# 9. 将 1924 年以及之后的 Skating 和 Ice Hockey 项目的赛事数目填为 0
mask = (data_long['Year'] >= 1924) & (data_long['Sport'].isin(['Skating', 'Ice␣
 ↪Hockey']))
data_long.loc[mask, 'Events'] = 0

# 10. 分组处理，按运动种类单独训练模型
for sport, group in data_long.groupby('Sport'):
    # 分离已知数据和缺失数据
    known_data = group.dropna(subset=['Events'])
    missing_data = group[group['Events'].isna()]

    if not known_data.empty and not missing_data.empty:
        # 准备训练数据
        X_known = known_data[['Year']]
        y_known = known_data['Events']

        # 检查已知数据的数量
        if len(y_known) < 5:
            print(f"警告：运动种类 '{sport}' 的已知数据太少，使用 KNN 或线性回归填
充。")
```

```
            # 尝试使用线性回归
            if len(y_known) >= 3:  # 至少需要 3 个点来拟合线性回归
                model = LinearRegression()
                model.fit(X_known, y_known)
                predicted_events = model.predict(missing_data[['Year']])
            else:  # 使用 KNN, K=1
                model = KNeighborsRegressor(n_neighbors=1)
                model.fit(X_known, y_known)
                predicted_events = model.predict(missing_data[['Year']])

            # 将预测值四舍五入为整数
            predicted_events = np.round(predicted_events).astype(int)

            # 将预测值转换为 Pandas Series, 并确保索引对齐
            predicted_series = pd.Series(predicted_events, index=missing_data.index)

            # 填充缺失值
            data_long.loc[data_long['Sport'] == sport, 'Events'] = data_long.loc[data_long['Sport'] == sport, 'Events'].fillna(predicted_series)
        else:
            # 训练随机森林模型
            model = RandomForestRegressor(n_estimators=100, random_state=42)
            model.fit(X_known, y_known)

            # 预测缺失数据
            X_missing = missing_data[['Year']]
            predicted_events = model.predict(X_missing)

            # 将预测值四舍五入为整数
            predicted_events = np.round(predicted_events).astype(int)

            # 将预测值转换为 Pandas Series, 并确保索引对齐
            predicted_series = pd.Series(predicted_events, index=missing_data.index)
            # 填充缺失值
            data_long.loc[data_long['Sport'] == sport, 'Events'] = data_long.
 ↪loc[data_long['Sport'] == sport, 'Events'].fillna(predicted_series)

            # 记录日志
            print(f"运动种类 '{sport}' 的模型训练完成，预测了␣
 ↪{len(predicted_events)} 个缺失值。")
    else:
        print(f"运动种类 '{sport}' 没有缺失数据或没有足够的已知数据。")

# 11. 将数据重新转换为宽格式
data_filled = data_long.pivot_table(index=['Sport', 'Discipline', 'Code',␣
 ↪'Sports Governing Body'],
                                    columns='Year',
                                    values='Events',
                                    aggfunc='first').reset_index()

# 12. 输出结果
print("\n填充后的数据：")
print(data_filled.head())

# 13. 保存结果到新的 CSV 文件
output_path = 'Generated\\summerOly_programs_filled.csv'
data_filled.to_csv(output_path, index=False, encoding='utf-8')  # 确保保存时使用
正确的编码
print(f"填充后的数据已保存到 {output_path}")
```

```
运动种类 'Aquatics' 没有缺失数据或没有足够的已知数据。
运动种类 'Archery' 没有缺失数据或没有足够的已知数据。
运动种类 'Athletics' 没有缺失数据或没有足够的已知数据。
运动种类 'Badminton' 的模型训练完成，预测了 2 个缺失值。
运动种类 'Baseball and Softball' 的模型训练完成，预测了 8 个缺失值。
运动种类 'Basketball' 的模型训练完成，预测了 2 个缺失值。
运动种类 'Basque Pelota' 的模型训练完成，预测了 4 个缺失值。
运动种类 'Boxing' 没有缺失数据或没有足够的已知数据。
运动种类 'Breaking' 没有缺失数据或没有足够的已知数据。
```

```
运动种类 'Canoeing' 的模型训练完成，预测了 1 个缺失值。
运动种类 'Cricket' 没有缺失数据或没有足够的已知数据。
运动种类 'Croquet' 没有缺失数据或没有足够的已知数据。
运动种类 'Cycling' 没有缺失数据或没有足够的已知数据。
运动种类 'Equestrian' 没有缺失数据或没有足够的已知数据。
运动种类 'Fencing' 没有缺失数据或没有足够的已知数据。
运动种类 'Field hockey' 没有缺失数据或没有足够的已知数据。
运动种类 'Flag football' 没有缺失数据或没有足够的已知数据。
运动种类 'Football' 没有缺失数据或没有足够的已知数据。
运动种类 'Golf' 没有缺失数据或没有足够的已知数据。
运动种类 'Gymnastics' 没有缺失数据或没有足够的已知数据。
运动种类 'Handball' 的模型训练完成，预测了 1 个缺失值。
运动种类 'Ice Hockey' 没有缺失数据或没有足够的已知数据。
运动种类 'Jeu de Paume' 没有缺失数据或没有足够的已知数据。
运动种类 'Judo' 没有缺失数据或没有足够的已知数据。
运动种类 'Karate' 没有缺失数据或没有足够的已知数据。
运动种类 'Lacrosse' 的模型训练完成，预测了 3 个缺失值。
运动种类 'Modern Pentathlon' 没有缺失数据或没有足够的已知数据。
运动种类 'Polo' 没有缺失数据或没有足够的已知数据。
运动种类 'Rackets' 没有缺失数据或没有足够的已知数据。
运动种类 'Roque' 没有缺失数据或没有足够的已知数据。
运动种类 'Rowing' 没有缺失数据或没有足够的已知数据。
运动种类 'Rugby' 没有缺失数据或没有足够的已知数据。
运动种类 'Sailing' 没有缺失数据或没有足够的已知数据。
运动种类 'Shooting' 没有缺失数据或没有足够的已知数据。
运动种类 'Skateboarding' 没有缺失数据或没有足够的已知数据。
运动种类 'Skating' 没有缺失数据或没有足够的已知数据。
运动种类 'Sport Climbing' 没有缺失数据或没有足够的已知数据。
运动种类 'Squash' 没有缺失数据或没有足够的已知数据。
运动种类 'Surfing' 没有缺失数据或没有足够的已知数据。
运动种类 'Table Tennis' 没有缺失数据或没有足够的已知数据。
运动种类 'Taekwondo' 的模型训练完成，预测了 2 个缺失值。
运动种类 'Tennis' 的模型训练完成，预测了 2 个缺失值。
运动种类 'Total disciplines' 没有缺失数据或没有足够的已知数据。
运动种类 'Total events' 没有缺失数据或没有足够的已知数据。
运动种类 'Total sports' 没有缺失数据或没有足够的已知数据。
运动种类 'Triathlon' 没有缺失数据或没有足够的已知数据。
```

```
运动种类 'Tug of War' 没有缺失数据或没有足够的已知数据。
运动种类 'Volleyball' 的模型训练完成，预测了 1 个缺失值。
运动种类 'Water Motorsports' 的模型训练完成，预测了 1 个缺失值。
运动种类 'Weightlifting' 没有缺失数据或没有足够的已知数据。
运动种类 'Wrestling' 没有缺失数据或没有足够的已知数据。

填充后的数据:
Year     Sport          Discipline Code Sports Governing Body  1896  1900  \
0     Aquatics  Artistic Swimming  SWA         World Aquatics   0.0   0.0
1     Aquatics             Diving  DIV         World Aquatics   0.0   0.0
2     Aquatics  Marathon Swimming  OWS         World Aquatics   0.0   0.0
3     Aquatics           Swimming  SWM         World Aquatics   4.0   7.0
4     Aquatics         Water Polo  WPO         World Aquatics   0.0   1.0

Year  1904  1906  1908  1912  …  1988  1992  1996  2000  2004  2008  2012  \
0      0.0   0.0   0.0   0.0  …   2.0   2.0   1.0   2.0   2.0   2.0   2.0
1      2.0   1.0   2.0   4.0  …   4.0   4.0   4.0   8.0   8.0   8.0   8.0
2      0.0   0.0   0.0   0.0  …   0.0   0.0   0.0   0.0   0.0   2.0   2.0
3      9.0   4.0   6.0   9.0  …  31.0  31.0  32.0  32.0  32.0  32.0  32.0
4      1.0   0.0   1.0   1.0  …   1.0   1.0   1.0   2.0   2.0   2.0   2.0

Year  2016  2020  2024
0      2.0   2.0   2.0
1      8.0   8.0   8.0
2      2.0   2.0   2.0
3     32.0  35.0  35.0
4      2.0   2.0   2.0

[5 rows x 35 columns]
填充后的数据已保存到 Generated\summerOly_programs_filled.csv
```

1.2.3 Medal_counts 数据清洗

[47]:

```
# 2. 数据清洗
# 确保数据的格式正确
data = medal_counts[['Year', 'NOC', 'Gold', 'Silver', 'Bronze', 'Total']]
```

```
# 3. 创建年份和国家的索引
years = data['Year'].unique()
noc = data['NOC'].unique()

# 4. 定义一个函数来生成表格
def generate_table(data, column_name):
    # 创建一个空的 DataFrame，以年份为列，国家为行
    table = pd.DataFrame(index=noc, columns=years)

    # 填充数据
    for index, row in data.iterrows():
        year = row['Year']
        country = row['NOC']
        value = row[column_name]
        table.at[country, year] = value

    # 推断数据类型并填充缺失值为 0
    table = table.infer_objects(copy=False).fillna(0).astype(int)

    return table

# 5. 生成金牌、银牌、铜牌和总数的表格
gold_table = generate_table(data, 'Gold')
silver_table = generate_table(data, 'Silver')
bronze_table = generate_table(data, 'Bronze')
total_table = generate_table(data, 'Total')

# 6. 保存到新的 CSV 文件
gold_table.to_csv('Generated\\summerOly_gold_summary.csv')
silver_table.to_csv('Generated\\summerOly_silver_summary.csv')
bronze_table.to_csv('Generated\\summerOly_bronze_summary.csv')
total_table.to_csv('Generated\\summerOly_total_summary.csv')
```

[48]:
```
# 7. 输出结果
print("金牌表格：")
print(gold_table)
```

```
金牌表格:
                      1896  1900  1904  1908  1912  1920  1924  1928  1932  \
United States           11    19    76    23    26    41    45    22     0
Greece                  10     0     1     0     1     0     0     0     0
Germany                  6     4     4     3     5     0     0    10     0
France                   5    27     0     5     7     9    13     6     0
Great Britain            2    15     1    56    10    14     9     3     0
…                    …   …   …   …   …   …   …   …   …
Saint Lucia              0     0     0     0     0     0     0     0     0
Dominica                 0     0     0     0     0     0     0     0     0
Albania                  0     0     0     0     0     0     0     0     0
Cabo Verde               0     0     0     0     0     0     0     0     0
Refugee Olympic Team     0     0     0     0     0     0     0     0     0

                      1936  …  1988  1992  1996  2000  2004  2008  2012  \
United States           24  …    36    37    44    37    36    36    48
Greece                   0  …     0     2     4     4     6     0     0
Germany                 38  …     0    33    20    13    13    16    11
France                   7  …     6     8    15    13    11     7    11
Great Britain            4  …     5     5     1    11     9    19    29
…                    …  …   …   …   …   …   …   …   …
Saint Lucia              0  …     0     0     0     0     0     0     0
Dominica                 0  …     0     0     0     0     0     0     0
Albania                  0  …     0     0     0     0     0     0     0
Cabo Verde               0  …     0     0     0     0     0     0     0
Refugee Olympic Team     0  …     0     0     0     0     0     0     0

                      2016  2020  2024
United States           46    39    40
Greece                   3     2     1
Germany                 17    10    12
France                  10    10    16
Great Britain           27    22    14
…                    …   …   …
Saint Lucia              0     0     1
Dominica                 0     0     1
Albania                  0     0     0
```

```
Cabo Verde               0     0     0
Refugee Olympic Team     0     0     0

[210 rows x 30 columns]
```

[49]:
```python
print("\n银牌表格: ")
print(silver_table)
```

```
银牌表格:
                      1896  1900  1904  1908  1912  1920  1924  1928  1932  \
United States            7    14    78    12    19    27    27    18     0
Greece                  18     0     0     3     0     1     0     0     0
Germany                  5     3     5     5    13     0     0     7     0
France                   4    39     1     5     4    19    15    10     0
Great Britain            3     7     1    51    15    15    13    10     0
…                     …   …   …   …   …   …   …   …   …
Saint Lucia              0     0     0     0     0     0     0     0     0
Dominica                 0     0     0     0     0     0     0     0     0
Albania                  0     0     0     0     0     0     0     0     0
Cabo Verde               0     0     0     0     0     0     0     0     0
Refugee Olympic Team     0     0     0     0     0     0     0     0     0

                      1936  …  1988  1992  1996  2000  2004  2008  2012  \
United States           21  …    31    34    32    24    39    39    26
Greece                   0  …     0     0     4     6     6     2     0
Germany                 31  …     0    21    18    17    16    11    20
France                   6  …     4     5     7    14     9    16    11
Great Britain            7  …    10     3     8    10     9    13    18
…                     …  …   …   …   …   …   …   …   …
Saint Lucia              0  …     0     0     0     0     0     0     0
Dominica                 0  …     0     0     0     0     0     0     0
Albania                  0  …     0     0     0     0     0     0     0
Cabo Verde               0  …     0     0     0     0     0     0     0
Refugee Olympic Team     0  …     0     0     0     0     0     0     0

                      2016  2020  2024
```

```
United States         37   41   44
Greece                 1    1    1
Germany               10   11   13
France                18   12   26
Great Britain         23   20   22
…                    …    …    …
Saint Lucia            0    0    1
Dominica               0    0    0
Albania                0    0    0
Cabo Verde             0    0    0
Refugee Olympic Team   0    0    0

[210 rows x 30 columns]
```

[50]:

```python
print("\n铜牌表格： ")
print(bronze_table)
```

```
铜牌表格：
                      1896  1900  1904  1908  1912  1920  1924  1928  1932  \
United States            2    15    77    12    19    27    27    16     0
Greece                  19     0     1     1     1     0     0     0     0
Germany                  2     2     6     5     7     0     0    14     0
France                   2    37     0     9     3    13    10     5     0
Great Britain            2     9     0    39    16    13    12     7     0
…                    …    …    …    …    …    …    …    …    …
Saint Lucia              0     0     0     0     0     0     0     0     0
Dominica                 0     0     0     0     0     0     0     0     0
Albania                  0     0     0     0     0     0     0     0     0
Cabo Verde               0     0     0     0     0     0     0     0     0
Refugee Olympic Team     0     0     0     0     0     0     0     0     0

                      1936  …  1988  1992  1996  2000  2004  2008  2012  \
United States           12  …    27    37    25    32    26    37    30
Greece                   0  …     1     0     0     3     4     1     2
Germany                 32  …     0    28    27    26    20    14    13
France                   6  …     6    16    15    11    13    20    13
```

```
Great Britain            3  …    9   12    6    7   12   19   18
…                     …   …   …   …   …   …   …   …   …
Saint Lucia              0  …    0    0    0    0    0    0    0
Dominica                 0  …    0    0    0    0    0    0    0
Albania                  0  …    0    0    0    0    0    0    0
Cabo Verde               0  …    0    0    0    0    0    0    0
Refugee Olympic Team     0  …    0    0    0    0    0    0    0

                      2016  2020  2024
United States           38    33    42
Greece                   2     1     6
Germany                 15    16     8
France                  14    11    22
Great Britain           17    22    29
…                     …   …   …
Saint Lucia              0     0     0
Dominica                 0     0     0
Albania                  0     0     2
Cabo Verde               0     0     1
Refugee Olympic Team     0     0     1

[210 rows x 30 columns]
```

[51]:

```
print("\n总数表格: ")
print(total_table)
```

```
总数表格:
                      1896  1900  1904  1908  1912  1920  1924  1928  1932  \
United States           20    48   231    47    64    95    99    56     0
Greece                  47     0     2     4     2     1     0     0     0
Germany                 13     9    15    13    25     0     0    31     0
France                  11   103     1    19    14    41    38    21     0
Great Britain            7    31     2   146    41    42    34    20     0
…                     …   …   …   …   …   …   …   …   …
Saint Lucia              0     0     0     0     0     0     0     0     0
Dominica                 0     0     0     0     0     0     0     0     0
```

```
Albania                   0     0     0     0     0     0     0     0     0
Cabo Verde                0     0     0     0     0     0     0     0     0
Refugee Olympic Team      0     0     0     0     0     0     0     0     0

                      1936  …  1988  1992  1996  2000  2004  2008  2012  \
United States           57  …    94   108   101    93   101   112   104
Greece                   0  …     1     2     8    13    16     3     2
Germany                101  …     0    82    65    56    49    41    44
France                  19  …    16    29    37    38    33    43    35
Great Britain           14  …    24    20    15    28    30    51    65
…                    …  …   …   …   …   …   …   …   …
Saint Lucia              0  …     0     0     0     0     0     0     0
Dominica                 0  …     0     0     0     0     0     0     0
Albania                  0  …     0     0     0     0     0     0     0
Cabo Verde               0  …     0     0     0     0     0     0     0
Refugee Olympic Team     0  …     0     0     0     0     0     0     0

                      2016  2020  2024
United States          121   113   126
Greece                   6     4     8
Germany                 42    37    33
France                  42    33    64
Great Britain           67    64    65
…                    …   …   …
Saint Lucia              0     0     2
Dominica                 0     0     1
Albania                  0     0     2
Cabo Verde               0     0     1
Refugee Olympic Team     0     0     1

[210 rows x 30 columns]
```

1.2.4 清理 summerOly_medal_counts.csv 异常值

[52]:

```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer

# 读取 CSV 文件
file_path = '2025_Problem_C_Data\\summerOly_medal_counts.csv'
data = pd.read_csv(file_path)

# 定义一个函数，用于剔除非英文字符
def remove_non_english_chars(text):
    if pd.isnull(text):
        return text
    return re.sub(r'[^a-zA-Z]', '', text)

# 对 NOC 列进行数据检查和处理
data['NOC'] = data['NOC'].apply(remove_non_english_chars)

# 提取实际的奥运会年份
olympic_years = data['Year'].unique()
olympic_years = np.sort(olympic_years)  # 按年份排序
print("实际的奥运会年份: ", olympic_years)

# 将数据按年份和国家分组
data['Year'] = data['Year'].astype(int)
data['NOC'] = data['NOC'].astype(str)
data = data[['Year', 'NOC', 'Gold', 'Silver', 'Bronze', 'Total']]

# 获取所有国家
countries = data['NOC'].unique()

# 创建一个完整的年份和国家组合的 DataFrame
all_combinations = pd.MultiIndex.from_product([olympic_years, countries],␣
 ↪names=['Year', 'NOC']).to_frame(index=False)

# 合并数据，填充缺失值为 NaN（暂时不填充为 0）
```

```
complete_data = pd.merge(all_combinations, data, on=['Year', 'NOC'], how='left')

# 计算每个国家首次参加奥运会的时间
first_participation = complete_data[complete_data['Total'] > 0].
 ↪groupby('NOC')['Year'].min().reset_index()
first_participation.columns = ['NOC', 'First_Participation']
complete_data = pd.merge(complete_data, first_participation, on='NOC',␣
 ↪how='left')

# 将每个国家在首次参加之前的所有年份的奖牌数填充为 0
complete_data.loc[complete_data['Year'] < complete_data['First_Participation'],␣
 ↪['Gold', 'Silver', 'Bronze', 'Total']] = 0

# 将首次参加时间列删除，因为它已经不再需要
complete_data.drop(columns=['First_Participation'], inplace=True)

# 定义一个函数来处理每个奖牌类型
def knn_impute(column_name):
    # 提取需要处理的列
    grouped = complete_data[['Year', 'NOC', column_name]].groupby('NOC')
    # 将分组结果转换为多个 DataFrame
    grouped_dfs = [group for noc, group in grouped]
    for df in grouped_dfs:
        current_noc = df['NOC'].iloc[0]  # 由于每个分组的 'NOC' 是相同的，可以直接
取第一个值
        #print(f" 当前组的 NOC: {current_noc}")

        # 初始化 KNNImputer
        imputer = KNNImputer(n_neighbors=3, weights='distance')  # n_neighbors␣
 ↪是邻居数量，weights 可以选择 'uniform' 或 'distance'

        # 选择需要填充的列
        try:
            df_filled = imputer.fit_transform(df[['Year', column_name]])
        except ValueError as e:
            print(f"Error processing {current_noc} for {column_name}: {e}")
            continue

        # 将结果转换回 DataFrame
        df_filled = pd.DataFrame(df_filled, columns=['Year', column_name])
        #print(df_filled)

        df_filled['Year'] = df_filled['Year'].astype(int)
        df_filled[column_name] = df_filled[column_name].round().astype(int)

        #print(df_filled)

        # 合并回原始数据
        #complete_data['NOC'==current_noc, column_name] = df_filled[column_name]
        for year in df_filled['Year']:
            index = df_filled[df_filled['Year'] == year].index[0]
            original_value = complete_data.loc[(complete_data['NOC'] ==␣
↪current_noc) & (complete_data['Year'] == year), column_name]

            if original_value.isna().any():
                # 如果存在 NaN 值，进行填充
                complete_data.loc[(complete_data['NOC'] == current_noc) &␣
↪(complete_data['Year'] == year), column_name] = df_filled[column_name].
↪iloc[index]
            else:
                # 获取原始值和填充值
                original_value = original_value.values[0]  # 获取具体的数值
                imputed_value = df_filled[column_name].iloc[index]

                # 检查分母是否为零
                if imputed_value != 0:
                    if abs(original_value - imputed_value) / imputed_value > 0.
↪2:
                        print(f"Large difference detected for {current_noc} in␣
↪{year}: original={original_value}, imputed={imputed_value}")
                        complete_data.loc[(complete_data['NOC'] == current_noc)␣
↪& (complete_data['Year'] == year), column_name] = imputed_value
```

```python
                #else:
                    #print(f"Imputed value is zero for {current_noc} in {year}, skipping division.")

def adjust_outliers(column_name):
    print(f"Adjusting outliers for {column_name}")
    for current_noc in countries:
        country_data = complete_data[complete_data['NOC'] == current_noc][['Year', column_name]].sort_values(by='Year')
        for i in range(1, len(country_data) - 1):
            current_year = country_data.iloc[i]['Year']
            current_value = country_data.iloc[i][column_name]
            prev_value = country_data.iloc[i - 1][column_name]
            next_value = country_data.iloc[i + 1][column_name]

            if prev_value == 0 or next_value == 0:
                continue

            # 计算左右年份的平均值
            avg_value = (prev_value + next_value) / 2

            # 检查当前值是否偏离平均值超过 50%
            if ((abs(current_value - avg_value) > avg_value) and current_value >= avg_value) or ((abs(current_value - avg_value) > current_value) and current_value <= avg_value):
                # 替换为三个值的平均值
                new_value = (current_value + prev_value + next_value) / 3
                new_value = round(new_value)
                #print(f"Outlier detected for {current_noc} in {current_year}: original={current_value}, adjusted={new_value}")
                complete_data.loc[(complete_data['NOC'] == current_noc) & (complete_data['Year'] == current_year), column_name] = new_value

medal_list = ['Total', 'Gold', 'Silver', 'Bronze']

# 进行 KNN 补全
```

```python
for medal in medal_list:
    knn_impute(medal)

# 调整异常值
for medal in medal_list:
    adjust_outliers(medal)

# 保存处理后的数据为 CSV 文件
output_file = 'Generated\\summerOly_medal_counts_imputed.csv'
complete_data.to_csv(output_file, index=False)
print(f"处理后的数据已保存到 {output_file}")
```

```
实际的奥运会年份： [1896 1900 1904 1908 1912 1920 1924 1928 1932 1936 1948 1952␣
 ↪1956 1960
 1964 1968 1972 1976 1980 1984 1988 1992 1996 2000 2004 2008 2012 2016
 2020 2024]
Adjusting outliers for Total
Adjusting outliers for Gold
Adjusting outliers for Silver
Adjusting outliers for Bronze
处理后的数据已保存到 Generated\summerOly_medal_counts_imputed.csv
```

1.2.5 处理国家变更与如今不存在的国家

[53]:

```python
import pandas as pd

# 读取 CSV 文件
file_path = 'Generated\\summerOly_medal_counts_imputed.csv'
df = pd.read_csv(file_path)

# 定义国家名称映射关系
country_mapping = {
    'WestGermany': 'Germany',
    'EastGermany': 'Germany',
    'UnitedTeamofGermany': 'Germany',
    'RussianEmpire': 'Russia',
    'SovietUnion': 'Russia',
    'Czechoslovakia': 'CzechRepublic',
    'Yugoslavia': 'Serbia',
    'Bohemia': 'CzechRepublic',
    'Formosa': 'Taiwan',
    'Mixedteam': 'Mixedteam'
}

# 更新国家名称
df['NOC'] = df['NOC'].replace(country_mapping)

# 去除如今不存在的国家
current_countries = [
    'UnitedStates', 'Greece', 'Germany', 'France', 'GreatBritain', 'Hungary',␣
 ↪'Austria', 'Australia', 'Denmark', 'Switzerland',
    'Mixedteam', 'Belgium', 'Italy', 'Cuba', 'Canada', 'Spain', 'Luxembourg',␣
 ↪'Norway', 'Netherlands', 'India', 'Sweden',
    'Australasia', 'Finland', 'SouthAfrica', 'Estonia', 'Brazil', 'Japan',␣
 ↪'CzechRepublic', 'NewZealand', 'Yugoslavia',
    'Argentina', 'Uruguay', 'Poland', 'Haiti', 'Portugal', 'Romania', 'Egypt',␣
 ↪'Ireland', 'Chile', 'Philippines', 'Mexico',
    'Latvia', 'Turkey', 'Jamaica', 'Peru', 'Ceylon', 'TrinidadandTobago',␣
 ↪'Panama', 'SouthKorea', 'Iran', 'PuertoRico',
    'Lebanon', 'Bulgaria', 'Venezuela', 'Iceland', 'Pakistan', 'Bahamas',␣
 ↪'Ethiopia', 'Ghana', 'Morocco', 'Singapore',
    'BritishWestIndies', 'Iraq', 'Tunisia', 'Kenya', 'Nigeria', 'Mongolia',␣
 ↪'Uganda', 'Cameroon', 'Taiwan', 'NorthKorea',
    'Colombia', 'Niger', 'Bermuda', 'Thailand', 'Zimbabwe', 'Tanzania',␣
 ↪'Guyana', 'China', 'IvoryCoast', 'Syria', 'Algeria',
    'ChineseTaipei', 'DominicanRepublic', 'Zambia', 'Suriname', 'CostaRica',␣
 ↪'Indonesia', 'NetherlandsAntilles', 'Senegal',
    'VirginIslands', 'Djibouti', 'UnifiedTeam', 'Lithuania', 'Namibia',␣
 ↪'Croatia', 'IndependentOlympicParticipants', 'Israel',
    'Slovenia', 'Malaysia', 'Qatar', 'Russia', 'Ukraine', 'CzechRepublic',␣
 ↪'Kazakhstan', 'Belarus', 'FRYugoslavia', 'Slovakia',
    'Armenia', 'Burundi', 'Ecuador', 'HongKong', 'Moldova', 'Uzbekistan',␣
  ↪'Azerbaijan', 'Tonga', 'Georgia', 'Mozambique',
    'SaudiArabia', 'SriLanka', 'Vietnam', 'Barbados', 'Kuwait', 'Kyrgyzstan',␣
  ↪'Macedonia', 'UnitedArabEmirates',
    'SerbiaandMontenegro', 'Paraguay', 'Eritrea', 'Serbia', 'Tajikistan',␣
  ↪'Samoa', 'Sudan', 'Afghanistan', 'Mauritius', 'Togo',
    'Bahrain', 'Grenada', 'Botswana', 'Cyprus', 'Gabon', 'Guatemala',␣
  ↪'Montenegro', 'IndependentOlympicAthletes', 'Fiji',
    'Jordan', 'Kosovo', 'ROC', 'SanMarino', 'NorthMacedonia', 'Turkmenistan',␣
  ↪'BurkinaFaso', 'SaintLucia', 'Dominica',
    'Albania', 'CaboVerde', 'RefugeeOlympicTeam'
]

# 保留当前存在的国家
df = df[df['NOC'].isin(current_countries)]

# 国家合并取均值
df_grouped = df.groupby(['Year', 'NOC']).mean().apply(np.floor).reset_index()

# 保存数据
df_grouped.to_csv('Generated\\summerOly_medal_counts_processed.csv')

# 查看处理后的数据
print(df_grouped.head(4))
```

```
   Year          NOC  Gold  Silver  Bronze  Total
0  1896  Afghanistan   0.0     0.0     0.0    0.0
1  1896      Albania   0.0     0.0     0.0    0.0
2  1896      Algeria   0.0     0.0     0.0    0.0
3  1896    Argentina   0.0     0.0     0.0    0.0
```

1.2.6 清理 athletes.csv 并转换格式为宽

[54]:

```
# 读取 summerOly_athletes.csv 文件
data = olympic_athletes.copy()

# 转换为长格式，将年份放到列的抬头位置
```

```python
pivot_df = data.pivot_table(index=['Name', 'Sex', 'Team', 'NOC', 'City', 'Sport', 'Event'],
                                  columns='Year',
                                  values='Medal',
                                  aggfunc='first').reset_index()

# 填充缺失值为 0
pivot_df = pivot_df.fillna(0)

# 输出结果
print("转换为宽格式后的数据：")
print(pivot_df.head())

# 保存为新的 CSV 文件
output_path = 'Generated\\summerOly_athletes_wide_format.csv'
pivot_df.to_csv(output_path, index=False, encoding='utf-8')
print(f"宽格式数据已保存到 {output_path}")
```

```
转换为宽格式后的数据：
Year           Name Sex          Team  NOC   City          Sport  \
0       (jr) Larocca   M     Argentina  ARG  Paris     Equestrian
1      . Chadalavada   F         India  IND  Tokyo        Fencing
2             . Deni   M     Indonesia  INA  Tokyo  Weightlifting
3                671   F         China  CHN  Paris       Breaking
4           A Alayed   F  Saudi Arabia  KSA  Paris       Swimming

Year                     Event 1896 1900 1904  … 1988 1992 1996 2000 2004  \
0           Jumping Individual    0    0    0  …    0    0    0    0    0
1     Women's Sabre Individual    0    0    0  …    0    0    0    0    0
2                   Men's 67kg    0    0    0  …    0    0    0    0    0
3                      B-Girls    0    0    0  …    0    0    0    0    0
4       Women's 200m Freestyle    0    0    0  …    0    0    0    0    0

Year 2008 2012 2016      2020      2024
0       0    0    0         0  No medal
1       0    0    0  No medal         0
2       0    0    0  No medal         0
```

```
3        0    0    0         0    Bronze
4        0    0    0         0  No medal

[5 rows x 38 columns]
宽格式数据已保存到 Generated\summerOly_athletes_wide_format.csv
```

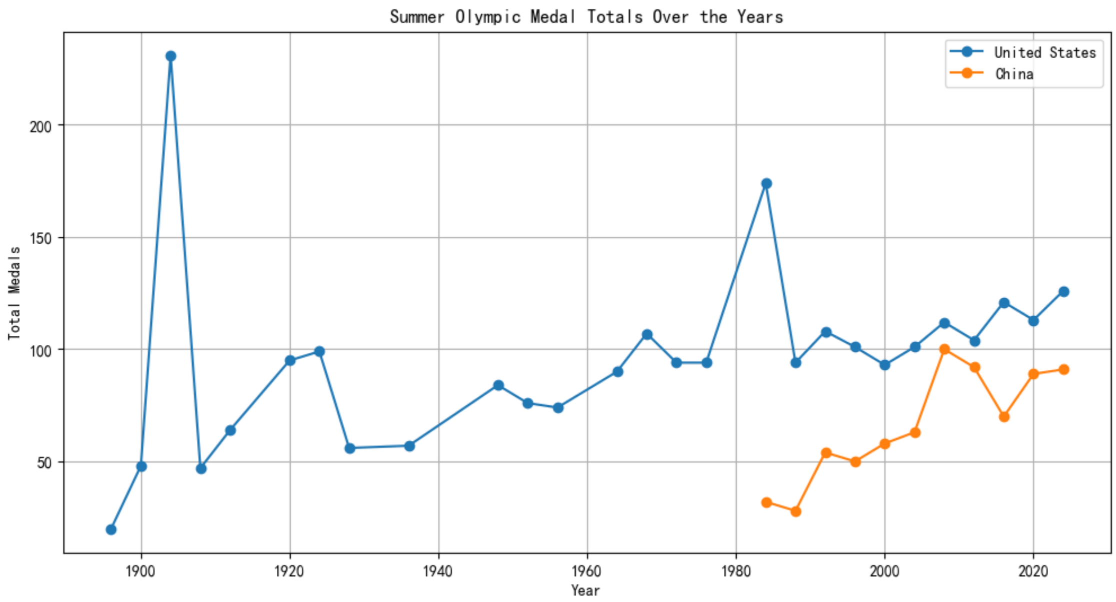
2 分析数据

2.1 国家级特征

[55]:

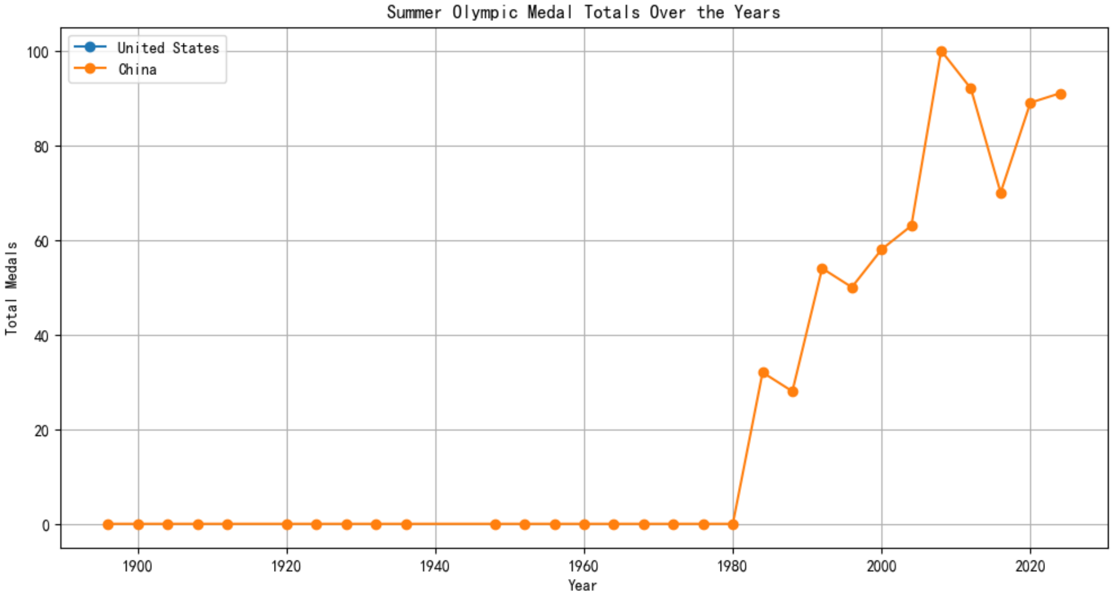
```python
import pandas as pd
import matplotlib.pyplot as plt

# 读取 CSV 文件
data = medal_counts.copy()

# 数据预处理
# 由于数据格式较为复杂，需要先将其转换为更易于处理的格式
# 提取年份和各个国家的奖牌总数
# 假设我们关注的是美国（United States）和中国的（China）奖牌总数
us_data = data[data['NOC'] == 'United States'][['Year', 'Total']].
  ↪rename(columns={'Total': 'US_Total'})
china_data = data[data['NOC'] == 'China'][['Year', 'Total']].
  ↪rename(columns={'Total': 'China_Total'})

# 合并数据
merged_data = pd.merge(us_data, china_data, on='Year', how='outer').
  ↪sort_values(by='Year')

# 绘制折线图
plt.figure(figsize=(12, 6))
plt.plot(merged_data['Year'], merged_data['US_Total'], label='United States',␣
  ↪marker='o')
plt.plot(merged_data['Year'], merged_data['China_Total'], label='China',␣
  ↪marker='o')
```

```
# 添加标题和图例
plt.title('Summer Olympic Medal Totals Over the Years')
plt.xlabel('Year')
plt.ylabel('Total Medals')
plt.legend()

# 显示网格
plt.grid(True)

# 显示图表
plt.show()
```



[56]:

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

# 读取 CSV 文件
data = pd.read_csv('Generated\\summerOly_medal_counts_imputed.csv')

# 数据预处理
# 由于数据格式较为复杂，需要先将其转换为更易于处理的格式
# 提取年份和各个国家的奖牌总数
# 假设我们关注的是美国（United States）和中国的（China）奖牌总数
us_data = data[data['NOC'] == 'United States'][['Year', 'Total']].
  ↪rename(columns={'Total': 'US_Total'})
china_data = data[data['NOC'] == 'China'][['Year', 'Total']].
  ↪rename(columns={'Total': 'China_Total'})

# 合并数据
merged_data = pd.merge(us_data, china_data, on='Year', how='outer').
  ↪sort_values(by='Year')

# 绘制折线图
plt.figure(figsize=(12, 6))
plt.plot(merged_data['Year'], merged_data['US_Total'], label='United States',␣
  ↪marker='o')
plt.plot(merged_data['Year'], merged_data['China_Total'], label='China',␣
  ↪marker='o')

# 添加标题和图例
plt.title('Summer Olympic Medal Totals Over the Years')
plt.xlabel('Year')
plt.ylabel('Total Medals')
plt.legend()

# 显示网格
plt.grid(True)

# 显示图表
plt.show()
```



2.2 项目级特征

[171]:

```
import pandas as pd

# 读取 CSV 文件
df = pd.read_csv('Generated\\summerOly_programs_filled.csv')

# 获取所有年份列
years = [col for col in df.columns if col.isdigit()]

# 初始化一个空的 DataFrame 来存储结果
result = pd.DataFrame(columns=['Year', 'Amount'])

# 遍历每个年份，计算总项目数
for year in years:
    total_events = df[year].sum()
    new_row = pd.DataFrame({'Year': [int(year)], 'Amount': [int(total_events)]})
    result = pd.concat([result, new_row], ignore_index=True)

# 显示结果
```

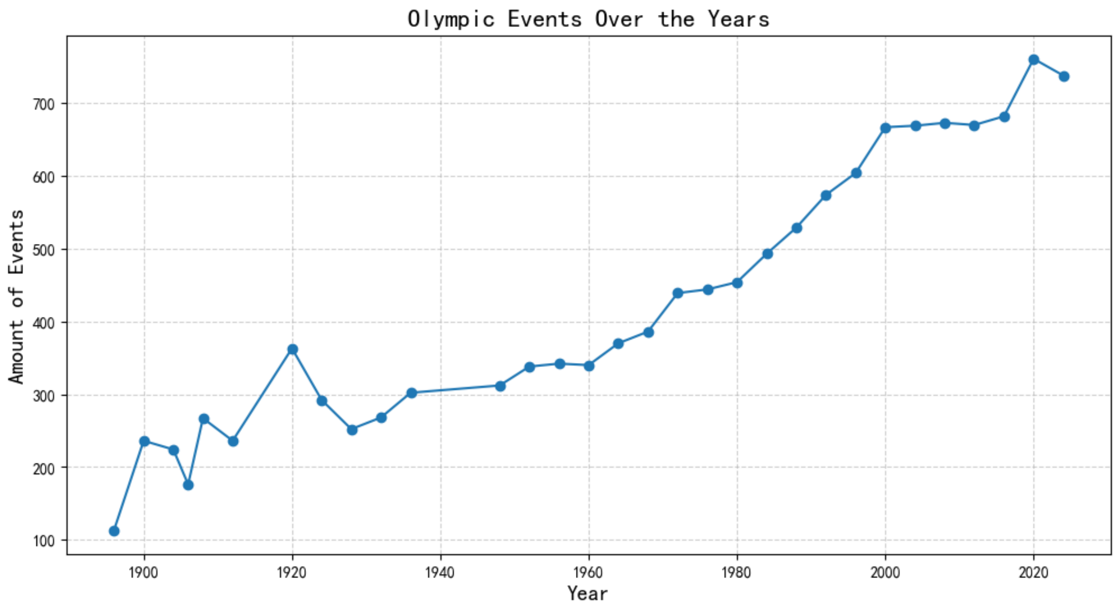
```python
print(result.head())

result.to_csv('Generated\\Project_amount.csv')
```

```
   Year  Amount
0  1896     113
1  1900     236
2  1904     224
3  1906     176
4  1908     267
```

2.2.1 可视化

[172]:

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

# 读取 CSV 文件
df = pd.read_csv('Generated\\Project_amount.csv')

# 绘制折线图
plt.figure(figsize=(12, 6))  # 设置图形大小
plt.plot(df['Year'], df['Amount'], marker='o', linestyle='-')  # 绘制折线图，添加标记点

# 添加标题和标签
plt.title('Olympic Events Over the Years', fontsize=16)  # 添加标题
plt.xlabel('Year', fontsize=14)  # 添加 x 轴标签
plt.ylabel('Amount of Events', fontsize=14)  # 添加 y 轴标签

# 添加网格线
plt.grid(True, linestyle='--', alpha=0.6)

# 显示图形
plt.show()
```

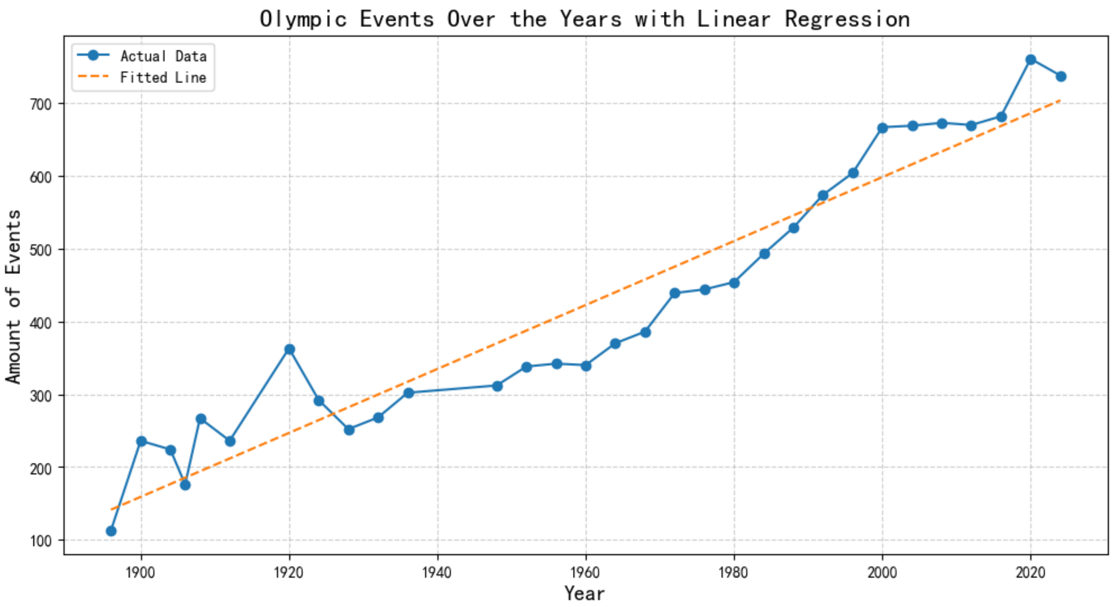


2.2.2 线性拟合

[177]:

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt

# 读取 CSV 文件
df = pd.read_csv('Generated\\Project_amount.csv')

# 准备数据
X = df['Year'].values.reshape(-1, 1)  # 将年份作为自变量
y = df['Amount'].values  # 将项目数作为因变量

# 创建线性回归模型
model = LinearRegression()

# 拟合模型
model.fit(X, y)
```

```
# 预测 2028 年的总项目数
year_2028 = np.array([2028]).reshape(-1, 1)
predicted_amount_2028 = model.predict(year_2028)

# 计算决定系数 R^2
y_pred = model.predict(X)
r2 = r2_score(y, y_pred)

# 绘制折线图和拟合线
plt.figure(figsize=(12, 6))
plt.plot(X, y, marker='o', linestyle='-', label='Actual Data')  # 绘制实际数据
plt.plot(X, y_pred, linestyle='--', label='Fitted Line')  # 绘制拟合线

# 添加标题和标签
plt.title('Olympic Events Over the Years with Linear Regression', fontsize=16)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Amount of Events', fontsize=14)

# 添加网格线
plt.grid(True, linestyle='--', alpha=0.6)

# 添加图例
plt.legend()

# 显示图形
plt.show()

# 打印预测结果和拟合度
print(f"预测 2028 年的总项目数: {round(predicted_amount_2028[0])}")
print(f"决定系数 R^2: {r2:.4f}")
```



```
预测 2028 年的总项目数: 721
决定系数 R^2: 0.9177
```

2.3 运动员级特征

2.3.1 预处理

```
[57]: # 读取 summerOly_athletes.csv 文件
data = olympic_athletes.copy()

# 提取必要的列
athlete_years = olympic_athletes[['Name', 'Sex', 'NOC', 'Team', 'Year',␣
 ↪'Sport', 'Event']].drop_duplicates()
# 合并 Name, Sex, NOC 列
athlete_years['Feature'] = athlete_years['Name'] + ', ' + athlete_years['Sex'] ␣
 ↪+ ', ' + athlete_years['NOC']

# 删除原始的 Name, Sex, NOC 列
#athlete_years = athlete_years.drop(columns=['Name', 'Sex', 'NOC'])

# 对每个运动员进行排序
```

```
athlete_years = athlete_years.sort_values(by=['Feature','Year'])
athlete_years.to_csv('Generated\\athlete_yesrs.csv', index=False,␣
 ↪encoding='utf-8')
```

2.3.2 添加唯一特征值

[58]:

```
import os

# 设置环境变量 LOKY_MAX_CPU_COUNT
os.environ["LOKY_MAX_CPU_COUNT"] = "8"  # 使用 CPU 核心数
```

[59]:

```
import pandas as pd
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# 读取 CSV 文件
file_path = 'Generated\\athlete_yesrs.csv'  # 替换为你的文件路径
data = pd.read_csv(file_path)

# 显示原始数据的前几行
print("原始数据的前几行: ")
print(data.head())

# 设置时间阈值
time_threshold_small = 12
time_threshold_large = 44

# 按 Feature 分组
grouped = data.groupby('Feature')

# 用于存储处理后的数据
processed_data = []

# 遍历每个分组
for feature, group in grouped:
    # 按 Year 排序
    group = group.sort_values(by='Year')
```

```
# 初始化变量
unique_feature_count = 0
last_year = None

# 遍历分组中的每条记录
for index, row in group.iterrows():
    current_year = row['Year']

    # 判断是否为同一个运动员
    if last_year is not None:
        year_diff = current_year - last_year
        if year_diff > time_threshold_large:
            # 如果时间跨度大于 44 年，直接认为是不同运动员
            unique_feature_count += 1
        elif year_diff > time_threshold_small:
            # 如果时间跨度在 12 到 44 年之间，进行聚类分析
            features_cluster = group[['Year', 'Sport', 'Event']].apply(lambda x: x.factorize()[0])
            features_cluster = StandardScaler().fit_transform(features_cluster)

            # 使用 DBSCAN 聚类
            dbscan = DBSCAN(eps=0.5, min_samples=2)
            group['Cluster'] = dbscan.fit_predict(features_cluster)

            # 为每个聚类生成唯一标识
            for cluster in group['Cluster'].unique():
                cluster_group = group[group['Cluster'] == cluster]
                for _, cluster_row in cluster_group.iterrows():
                    new_feature = f"{feature}_{cluster}"
                    processed_data.append({
                        'Name': cluster_row['Name'],
                        'Sex': cluster_row['Sex'],
                        'Team': cluster_row['Team'],
                        'NOC': cluster_row['NOC'],
                            'Year': cluster_row['Year'],
                            'Sport': cluster_row['Sport'],
                            'Event': cluster_row['Event'],
                            'Feature': new_feature
                        })
                        unique_feature_count += 1
                break  # 已经处理完当前分组，跳出循环

        # 如果时间跨度在阈值内，认为是同一个运动员
        new_feature = f"{feature}_{unique_feature_count}"
        processed_data.append({
            'Name': row['Name'],
            'Sex': row['Sex'],
            'Team': row['Team'],
            'NOC': row['NOC'],
            'Year': row['Year'],
            'Sport': row['Sport'],
            'Event': row['Event'],
            'Feature': new_feature
        })

        # 更新变量
        last_year = current_year

# 将处理后的数据转换为 DataFrame
processed_df = pd.DataFrame(processed_data)

# 显示处理后的数据
print("\n处理后的数据：")
print(processed_df[['Feature', 'Sport', 'Event', 'Year']].head())

# 保存处理后的数据到新的 CSV 文件
output_file_path = 'Generated\\athlete_years_processed.csv'
processed_df.to_csv(output_file_path, index=False)
print(f"\n处理后的数据已保存到 {output_file_path}")
```

原始数据的前几行：

```
             Name Sex  NOC          Team  Year         Sport  \
0    (jr) Larocca   M  ARG     Argentina  2024    Equestrian
1   . Chadalavada   F  IND         India  2020       Fencing
2          . Deni   M  INA     Indonesia  2020  Weightlifting
3             671   F  CHN         China  2024      Breaking
4        A Alayed   F  KSA  Saudi Arabia  2024      Swimming

                     Event             Feature
0        Jumping Individual   (jr) Larocca, M, ARG
1  Women's Sabre Individual  . Chadalavada, F, IND
2                 Men's 67kg         . Deni, M, INA
3                   B-Girls            671, F, CHN
4    Women's 200m Freestyle       A Alayed, F, KSA

处理后的数据:
                   Feature         Sport                     Event  Year
0   (jr) Larocca, M, ARG_0    Equestrian        Jumping Individual  2024
1  . Chadalavada, F, IND_0       Fencing  Women's Sabre Individual  2020
2         . Deni, M, INA_0  Weightlifting                Men's 67kg  2020
3            671, F, CHN_0      Breaking                   B-Girls  2024
4       A Alayed, F, KSA_0      Swimming    Women's 200m Freestyle  2024

处理后的数据已保存到 Generated\athlete_years_processed.csv
```

2.3.3 统计连续参加奥运会的年数与对应人数

[60]:

```python
# 读取 CSV 文件
file_path = 'Generated\\athlete_years_processed.csv'  # 替换为你的文件路径
athlete_years = pd.read_csv(file_path)
```

[61]:

```python
# 计算连续参加的届数
def count_consecutive_years(group):
    years = group['Year'].sort_values().values
    consecutive_year = []
    current_count = 1
    for i in range(1, len(years)):
        if years[i] - years[i - 1] <= 6 :
            if years[i] - years[i - 1] >= 3:
                current_count += 1
        else:
            if current_count > 10:
                print(group)
            consecutive_year.append(current_count)
            current_count = 1
    consecutive_year.append(current_count)
    return pd.Series(consecutive_year)

# 应用函数计算每个运动员的连续届数
consecutive_years = athlete_years.groupby('Feature').apply(count_consecutive_years, include_groups=False).explode().reset_index()
consecutive_years.columns = ['Feature', 'level_0',  'Consecutive_Years']  # 修正列名
consecutive_years = consecutive_years.drop(columns=['level_0'])  # 删除不必要的列

# 统计每个连续届数的人数
consecutive_years_count = consecutive_years['Consecutive_Years'].value_counts().reset_index()
consecutive_years_count.columns = ['Consecutive_Years', 'Count']

# 输出结果
print("连续参加奥运会的届数与对应人次: ")
print(consecutive_years_count)

# 保存为新的 CSV 文件
output_path = 'Generated\\consecutive_years_count.csv'
consecutive_years_count.to_csv(output_path, index=False, encoding='utf-8')
print(f"统计结果已保存到 {output_path}")
```

```
连续参加奥运会的届数与对应人次:
   Consecutive_Years   Count
0                  1  108202
1                  2   23470
```

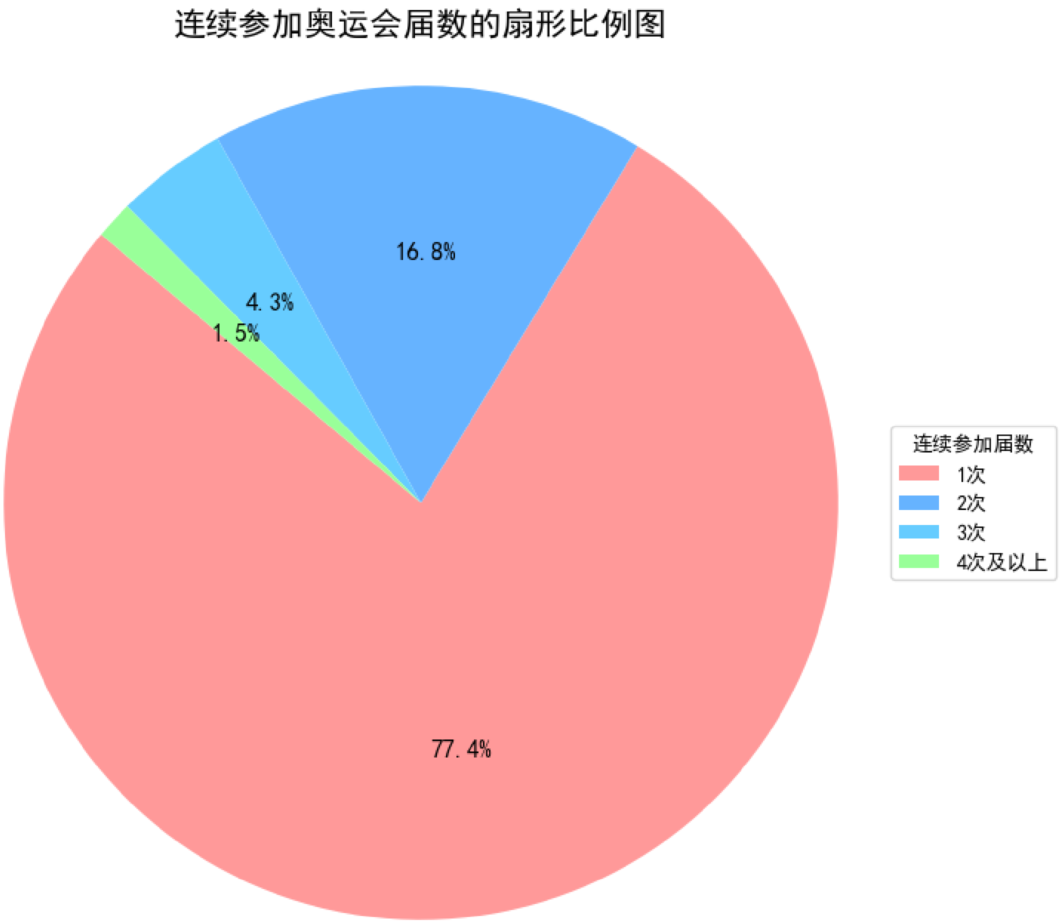
```
2                   3   6036
3                   4   1575
4                   5    372
5                   6     79
6                   7     18
7                   8      4
8                   9      1
统计结果已保存到 Generated\consecutive_years_count.csv
```

数据可视化

[62]:

```python
# 导入必要的库
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

# 设置支持中文的字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体字体
plt.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 读取数据
data = pd.read_csv("Generated/consecutive_years_count.csv")

# 定义大致届数区间
bins = [0, 2, 3, 4, 14]
labels = ['1次', '2次', '3次', '4次及以上']

# 将数据分组到区间
data['Group'] = pd.cut(data['Consecutive_Years'], bins=bins, labels=labels, right=False)

# 计算每个区间的总人次，显式设置 observed=True
grouped_data = data.groupby('Group', observed=True)['Count'].sum().reset_index()

# 准备绘图数据
labels = grouped_data['Group']
sizes = grouped_data['Count']
colors = ['#ff9999', '#66b3ff', '#66ccff', '#99ff99']  # 颜色列表
```

```python
# 绘制饼图
plt.figure(figsize=(8, 8))
wedges, texts, autotexts = plt.pie(sizes, colors=colors, autopct='%1.1f%%', startangle=140, textprops={'fontsize': 12})

# 添加图例（色块 + 标签），放置在右侧
plt.legend(wedges, labels, title="连续参加届数", loc="center left", bbox_to_anchor=(1, 0, 0.5, 1))

plt.title('连续参加奥运会届数的扇形比例图', fontsize=16)
plt.axis('equal')  # 确保饼图是圆形
plt.show()
```



[63]:

```
# 保存组别与对应比例
group_percentages = []
for label, autotext in zip(labels, autotexts):
    # 获取百分比文本并去掉百分号，转换为浮点数
    percentage = float(autotext.get_text().strip('%'))
    group_percentages.append((label, percentage))

# 打印结果
print("组别与对应比例：")
for group, percentage in group_percentages:
    print(f"{group}: {percentage:.1f}%")
```

```
组别与对应比例：
1 次: 77.4%
2 次: 16.8%
3 次: 4.3%
4 次及以上: 1.5%
```

2.3.4 统计运动员参加奥运会的时间跨度

[64]:

```
# 读取 CSV 文件
file_path = 'Generated\\athlete_years_processed.csv'  # 替换为你的文件路径
athlete_years = pd.read_csv(file_path)
```

[65]:

```
# 计算每个运动员的第一次和最后一次参赛年份
def calculate_year_gap(group):
    years = group['Year'].values
    min_n = 2032
    max_n = 1896
    for i in years:
        if i < min_n:
            min_n = i
        if i > max_n:
            max_n = i
    if len(years) > 0:
        if max_n - min_n + 1 > 60:
            #print(group)
            return 1
        return max_n - min_n + 1
    else:
        return 0

# 应用函数计算每个运动员的间隔年数
athlete_gaps = athlete_years.groupby('Feature').apply(calculate_year_gap,␣
 ↪include_groups=False).reset_index()
athlete_gaps.columns = ['Feature', 'Year_Gap']

# 统计每个间隔年数的人数
gap_counts = athlete_gaps['Year_Gap'].value_counts().reset_index()
gap_counts.columns = ['Year_Gap', 'Count']

# 按 Year_Gap 排序
gap_counts = gap_counts.sort_values(by='Year_Gap')

# 输出结果
print("运动员第一次参加奥运会和最后一次参加奥运会之间的间隔年数：")
print(gap_counts)

# 保存为新的 CSV 文件
output_path = 'Generated\\athlete_year_gaps.csv'
gap_counts.to_csv(output_path, index=False, encoding='utf-8')
print(f"统计结果已保存到 {output_path}")
```

```
运动员第一次参加奥运会和最后一次参加奥运会之间的间隔年数：
    Year_Gap  Count
0          1  99249
8          3     98
1          5  21869
10         7     77
2          9   8014
18        11     12
3         13   2887
15        15     16
```

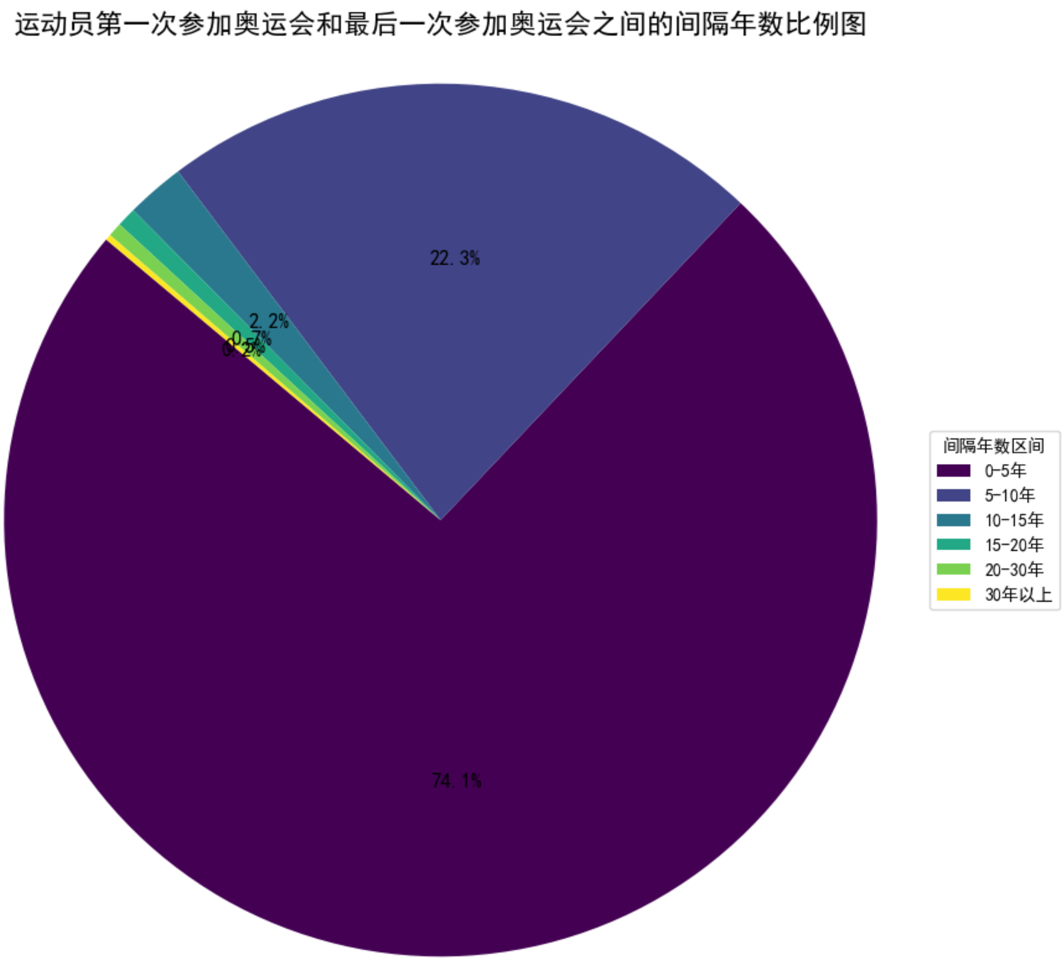
```
4           17    942
19          19      8
5           21    382
20          23      3
6           25    178
21          27      1
7           29    134
22          31      1
9           33     90
11          37     57
12          41     49
24          43      1
13          45     44
14          49     21
17          53     12
16          57     13
23          59      1
统计结果已保存到 Generated\athlete_year_gaps.csv
```

数据可视化

[66]:

```
# 导入必要的库
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# 设置支持中文的字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体字体
plt.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 读取数据
data = pd.read_csv("Generated\\athlete_year_gaps.csv")

# 定义大致间隔年数区间
bins = [0, 5, 10, 15, 20, 30, 120]  # 区间划分：0-5年，5-10年，10-15年，15-20年，20-30年，30年以上
labels = ['0-5年', '5-10年', '10-15年', '15-20年', '20-30年', '30年以上']
```

```python
# 将数据分组到区间
data['Group'] = pd.cut(data['Year_Gap'], bins=bins, labels=labels, right=False)

# 计算每个区间的总人次
grouped_data = data.groupby('Group', observed=True)['Count'].sum().reset_index()

# 准备绘图数据
labels = grouped_data['Group']
sizes = grouped_data['Count']
colors = plt.cm.viridis(np.linspace(0, 1, len(labels)))  # 使用颜色映射生成颜色列表

# 绘制饼图
plt.figure(figsize=(10, 10))
wedges, texts, autotexts = plt.pie(sizes, colors=colors, autopct='%1.1f%%', startangle=140, textprops={'fontsize': 12})

# 添加图例（色块 + 标签），放置在右侧
plt.legend(wedges, labels, title="间隔年数区间", loc="center left", bbox_to_anchor=(1, 0, 0.5, 1))

plt.title('运动员第一次参加奥运会和最后一次参加奥运会之间的间隔年数比例图', fontsize=16)
plt.axis('equal')  # 确保饼图是圆形
plt.show()
```


运动员第一次参加奥运会和最后一次参加奥运会之间的间隔年数比例图

’　”　根据扇形图，对于运动员连续参加比赛，只考虑连续参加 2-3 届的运动员的连续性影响，其余影响可以忽略不计。’　’　’

参加时间跨度为 **0-15** 年的运动员中连续参加的比例

[67]:
```python
# 合并时间跨度和连续届数数据
athlete_gaps.to_csv('Generated\\athlete_gaps.csv')
consecutive_years.to_csv('Generated\\consecutive_years.csv')
merged_data = athlete_gaps.merge(consecutive_years, on='Feature')

# 筛选出时间跨度为 1-15 年的运动员
filtered_data = merged_data[(merged_data['Year_Gap'] >= 1) & (merged_data['Year_Gap'] <= 15)]
```

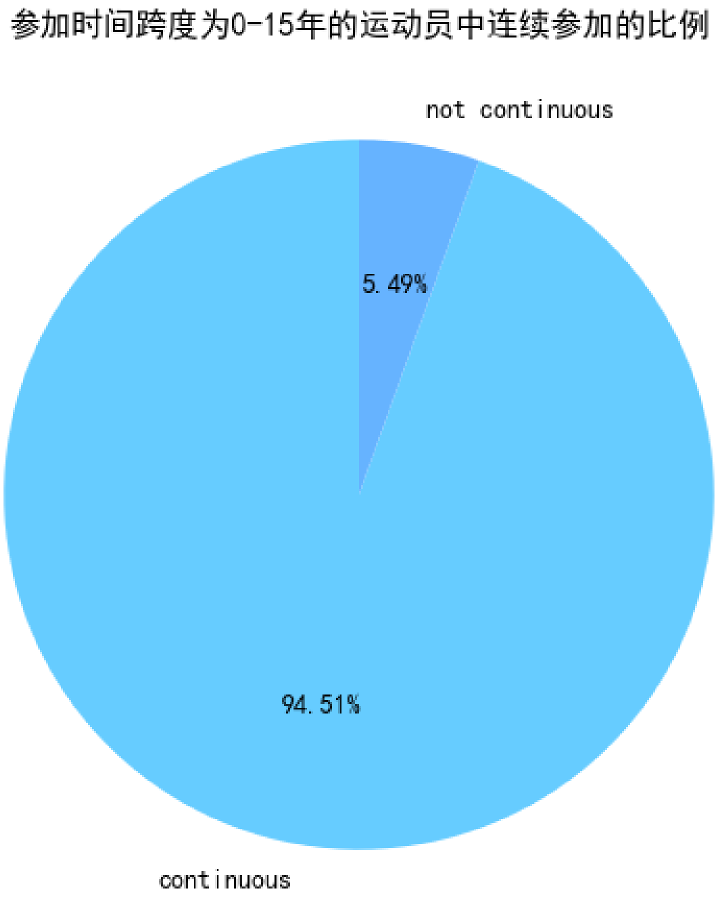
```python

# 统计连续参加的比例
total_count = filtered_data.shape[0]
consecutive_count = filtered_data[filtered_data['Year_Gap'] <=␣
 ↪filtered_data['Consecutive_Years']*4].shape[0]
consecutive_ratio = consecutive_count / total_count if total_count > 0 else 0

# 输出结果
print(f"时间跨度为 1-15 年的运动员中，连续参加的比例为：{consecutive_ratio:.2%}")

# 保存结果到 CSV 文件
output_path = 'Generated\\consecutive_ratio.csv'
filtered_data.to_csv(output_path, index=False, encoding='utf-8')
print(f"统计结果已保存到 {output_path}")
```

```
时间跨度为 1-15 年的运动员中，连续参加的比例为：94.51%
统计结果已保存到 Generated\consecutive_ratio.csv
```

数据可视化

[68]:

```python
# 导入必要的库
import matplotlib.pyplot as plt

# 设置支持中文的字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体字体
plt.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题

# 数据
percentages = [consecutive_ratio*100, (1-consecutive_ratio)*100]  # 一个百分数和
剩余部分
labels = ['continuous', 'not continuous']  # 标签
colors = ['#66ccff', '#66b3ff']  # 颜色

# 绘制饼图
plt.figure(figsize=(6, 6))  # 设置图形大小
plt.pie(percentages, labels=labels, colors=colors, autopct='%1.2f%%',␣
 ↪startangle=90)
# autopct='%1.2f%%' 表示在每个扇形上显示百分比，格式为 2 位小数
# startangle=90 表示从 90 度（即正上方）开始绘制

# 添加标题
plt.title('参加时间跨度为 0-15 年的运动员中连续参加的比例')

# 显示图形
plt.show()
```



结论

- ‘我们可以发现，参加奥运会时间跨度 0-15 年中绝大部分运动员都是连续参加的’
- ‘而且我们前面发现，绝大部分的运动员的时间跨度在 0-15 年之间，连续参加届数在 1-3 届’

- ‘而且我们知道，0-15 之间只能连续参加 1-3 次奥运会’
- ‘我们因此可以得出结论，绝大部分奥运会运动员连续参加了 1-3 次奥运会’
- ‘所以我们可以得出结论，考虑运动员连续参加比赛对奖牌的影响只需要考虑连续参加 2-3 次的情况’

[69]:
```
first_percentage = group_percentages[1][1]/
  ↪(group_percentages[0][1]+group_percentages[1][1]+group_percentages[2][1]+group_percentages[
print(f'一个参加了一次奥运会的运动员参加下一次奥运会的可能为{first_percentage : .
  ↪2f}' + '%')
second_percentage = group_percentages[2][1]/
  ↪(group_percentages[1][1]+group_percentages[2][1]+group_percentages[3][1])*100
print(f'一个参加了两次奥运会的运动员参加下一次奥运会的可能为{second_percentage : .
  ↪2f}' + '%')
third_percentage = group_percentages[3][1]/
  ↪(group_percentages[2][1]+group_percentages[3][1])*100
print(f'一个参加了三次奥运会的运动员参加下一次奥运会的可能为{third_percentage : .
  ↪2f}' + '%')
athlete_join_willing = {1 : first_percentage, 2 : second_percentage, 3 :␣
  ↪third_percentage}
```

```
一个参加了一次奥运会的运动员参加下一次奥运会的可能为 16.80%
一个参加了两次奥运会的运动员参加下一次奥运会的可能为 19.03%
一个参加了三次奥运会的运动员参加下一次奥运会的可能为 25.86%
```

3 构建模型

3.1 XGBoost

3.1.1 直接预测（超参数优化）

[70]:
```
import pandas as pd
import numpy as np
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder


#scikit-learn==1.5.2
```

```
# 加载数据
data = pd.read_csv('Generated\\summerOly_medal_counts_processed.csv')

# 数据预处理
# 将国家代码转换为数值标签
label_encoder = LabelEncoder()
data['NOC'] = label_encoder.fit_transform(data['NOC'])

# 处理缺失值
data = data.fillna(0)

# 创建特征：前一届奥运会的奖牌总数、金牌数、银牌数、铜牌数
data['Prev_Total'] = data.groupby('NOC')['Total'].shift(1)
data['Prev_Gold'] = data.groupby('NOC')['Gold'].shift(1)
data['Prev_Silver'] = data.groupby('NOC')['Silver'].shift(1)
data['Prev_Bronze'] = data.groupby('NOC')['Bronze'].shift(1)

# 填充缺失值
data['Prev_Total'] = data['Prev_Total'].fillna(0)
data['Prev_Gold'] = data['Prev_Gold'].fillna(0)
data['Prev_Silver'] = data['Prev_Silver'].fillna(0)
data['Prev_Bronze'] = data['Prev_Bronze'].fillna(0)

# 选择特征和目标变量
features = data[['Year', 'NOC', 'Prev_Total', 'Prev_Gold', 'Prev_Silver',␣
 ↪'Prev_Bronze']]
target_total = data['Total']
target_gold = data['Gold']
target_silver = data['Silver']
target_bronze = data['Bronze']

# 划分训练集和测试集
X_train_total, X_test_total, y_train_total, y_test_total =␣
 ↪train_test_split(features, target_total, test_size=0.2, random_state=42)
```

```
X_train_gold, X_test_gold, y_train_gold, y_test_gold =␣
 ↪train_test_split(features, target_gold, test_size=0.2, random_state=42)
X_train_silver, X_test_silver, y_train_silver, y_test_silver =␣
 ↪train_test_split(features, target_silver, test_size=0.2, random_state=42)
X_train_bronze, X_test_bronze, y_train_bronze, y_test_bronze =␣
 ↪train_test_split(features, target_bronze, test_size=0.2, random_state=42)

X_test = pd.DataFrame({'Total' : [X_test_total],
          'Gold' : [X_test_gold],
          'Silver' : [X_test_silver],
          'Bronze' : [X_test_bronze],
          })
y_test = pd.DataFrame({'Total' : [y_test_total],
          'Gold' : [y_test_gold],
          'Silver' : [y_test_silver],
          'Bronze' : [y_test_bronze],
          })

# 启用 GPU 加速
params = {
    #'tree_method' : "hist",
    #'device' : "cuda",
    #'predictor': 'gpu_predictor',  # 使用 GPU 进行预测
    'objective': 'reg:squarederror',
    'random_state': '42'
}


# 定义 XGBoost 模型
model_total = XGBRegressor(**params)
model_gold = XGBRegressor(**params)
model_silver = XGBRegressor(**params)
model_bronze = XGBRegressor(**params)

# 超参数优化
param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.7, 0.8, 0.9]
}

# 使用 GridSearchCV 进行超参数优化
grid_search_total = GridSearchCV(estimator=model_total, param_grid=param_grid,␣
 ↪cv=3, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_gold = GridSearchCV(estimator=model_gold, param_grid=param_grid,␣
 ↪cv=3, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_silver = GridSearchCV(estimator=model_silver,␣
 ↪param_grid=param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_bronze = GridSearchCV(estimator=model_bronze,␣
 ↪param_grid=param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1)

# 训练模型
grid_search_total.fit(X_train_total, y_train_total)
grid_search_gold.fit(X_train_gold, y_train_gold)
grid_search_silver.fit(X_train_silver, y_train_silver)
grid_search_bronze.fit(X_train_bronze, y_train_bronze)

# 获取最佳模型
best_model_total = grid_search_total.best_estimator_
best_model_gold = grid_search_gold.best_estimator_
best_model_silver = grid_search_silver.best_estimator_
best_model_bronze = grid_search_bronze.best_estimator_

# 评估模型
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f'MSE: {mse}')
    return y_pred

print("Total Medals Model Evaluation:")
```

```
evaluate_model(best_model_total, X_test_total, y_test_total)

print("Gold Medals Model Evaluation:")
evaluate_model(best_model_gold, X_test_gold, y_test_gold)

print("Silver Medals Model Evaluation:")
evaluate_model(best_model_silver, X_test_silver, y_test_silver)

print("Bronze Medals Model Evaluation:")
evaluate_model(best_model_bronze, X_test_bronze, y_test_bronze)

# 定义超参数网格
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [50, 100, 150],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9]
}

# 为每个国家单独训练模型
country_models = {}
for country in data['NOC'].unique():
    country_data = data[data['NOC'] == country]
    if len(country_data) > 1:  # 确保每个国家至少有两条记录
        country_data = country_data.replace([np.inf, -np.inf], np.nan)  # 替换无
穷值为 NaN
        country_data = country_data.ffill()  # 前向填充
        country_data = country_data.bfill()  # 后向填充
        country_features = country_data[['Year', 'NOC', 'Prev_Total',␣
 ↪'Prev_Gold', 'Prev_Silver', 'Prev_Bronze']]
        country_target_total = country_data['Total']
        country_target_gold = country_data['Gold']
        country_target_silver = country_data['Silver']
        country_target_bronze = country_data['Bronze']
        country_model_total = XGBRegressor(**params)
        country_model_gold = XGBRegressor(**params)
        country_model_silver = XGBRegressor(**params)
        country_model_bronze = XGBRegressor(**params)

        # 使用 GridSearchCV 进行超参数优化
        grid_search_country_model_total =␣
↪GridSearchCV(estimator=country_model_total, param_grid=param_grid, cv=2,␣
↪scoring='neg_mean_squared_error', n_jobs=-1, verbose=0)
        grid_search_country_model_gold =␣
↪GridSearchCV(estimator=country_model_gold, param_grid=param_grid, cv=2,␣
↪scoring='neg_mean_squared_error', n_jobs=-1, verbose=0)
        grid_search_country_model_silver =␣
↪GridSearchCV(estimator=country_model_silver, param_grid=param_grid, cv=2,␣
↪scoring='neg_mean_squared_error', n_jobs=-1, verbose=0)
        grid_search_country_model_bronze =␣
↪GridSearchCV(estimator=country_model_bronze, param_grid=param_grid, cv=2,␣
↪scoring='neg_mean_squared_error', n_jobs=-1, verbose=0)

        grid_search_country_model_total.fit(country_features,␣
↪country_target_total)
        grid_search_country_model_gold.fit(country_features,␣
↪country_target_gold)
        grid_search_country_model_silver.fit(country_features,␣
↪country_target_silver)
        grid_search_country_model_bronze.fit(country_features,␣
↪country_target_bronze)

        # 获取最佳模型
        best_country_model_total = grid_search_country_model_total.
↪best_estimator_
        best_country_model_gold = grid_search_country_model_gold.best_estimator_
        best_country_model_silver = grid_search_country_model_silver.
↪best_estimator_
```

```
        best_country_model_bronze = grid_search_country_model_bronze.
 ↪best_estimator_

        # 评估性能
        print("Total Medals Model Evaluation:")
        evaluate_model(best_country_model_total, X_test_total, y_test_total)
        print("Gold Medals Model Evaluation:")
        evaluate_model(best_country_model_gold, X_test_gold, y_test_gold)
        print("Silver Medals Model Evaluation:")
        evaluate_model(best_country_model_silver, X_test_silver, y_test_silver)
        print("Bronze Medals Model Evaluation:")
        evaluate_model(best_country_model_bronze, X_test_bronze, y_test_bronze)

        country_models[country] = {
            'total': best_country_model_total,
            'gold': best_country_model_gold,
            'silver': best_country_model_silver,
            'bronze': best_country_model_bronze
        }

# 预测 2028 年奥运会的奖牌数
next_year = 2028
predictions = []

for country in data['NOC'].unique():
    country_data = data[data['NOC'] == country]
    if len(country_data) > 1:
        prev_total = country_data['Total'].iloc[-1]
        prev_gold = country_data['Gold'].iloc[-1]
        prev_silver = country_data['Silver'].iloc[-1]
        prev_bronze = country_data['Bronze'].iloc[-1]

        next_data = pd.DataFrame({
            'Year': [next_year],
            'NOC': [country],
            'Prev_Total': [prev_total],
            'Prev_Gold': [prev_gold],
            'Prev_Silver': [prev_silver],
            'Prev_Bronze': [prev_bronze]
        })

        # 使用单独模型预测
        total_pred = country_models[country]['total'].predict(next_data)
        gold_pred = country_models[country]['gold'].predict(next_data)
        silver_pred = country_models[country]['silver'].predict(next_data)
        bronze_pred = country_models[country]['bronze'].predict(next_data)

        # 使用整体模型预测
        total_pred_global = best_model_total.predict(next_data)
        gold_pred_global = best_model_gold.predict(next_data)
        silver_pred_global = best_model_silver.predict(next_data)
        bronze_pred_global = best_model_bronze.predict(next_data)

        # 根据数据量分配权重
        data_count = len(country_data)
        weight = min(data_count / 10, 1)  # 数据量越多，权重越高，但不超过 1
        total_pred_combined = weight * total_pred + (1 - weight) *␣
↪total_pred_global
        gold_pred_combined = weight * gold_pred + (1 - weight) *␣
↪gold_pred_global
        silver_pred_combined = weight * silver_pred + (1 - weight) *␣
↪silver_pred_global
        bronze_pred_combined = weight * bronze_pred + (1 - weight) *␣
↪bronze_pred_global

        # 对预测结果取整
        total_pred_combined = round(total_pred_combined[0])
        gold_pred_combined = round(gold_pred_combined[0])
        silver_pred_combined = round(silver_pred_combined[0])
        bronze_pred_combined = round(bronze_pred_combined[0])

        predictions.append({
            'NOC': country,
            'Total_Predicted': total_pred_combined,
            'Gold_Predicted': gold_pred_combined,
            'Silver_Predicted': silver_pred_combined,
            'Bronze_Predicted': bronze_pred_combined
        })

# 将预测结果转换为 DataFrame
predictions_df = pd.DataFrame(predictions)

# 将 NOC 标签转换回国家代码
predictions_df['NOC'] = label_encoder.inverse_transform(predictions_df['NOC'])

# 输出预测结果
print(predictions_df)

# 保存预测结果到 CSV 文件
predictions_df.to_csv('Result\\2028_olympics_medal_predictions.csv',␣
  ↪index=False)
```

```
Total Medals Model Evaluation:
MSE: 14.183948320207696
Gold Medals Model Evaluation:
MSE: 2.9414593946059715
Silver Medals Model Evaluation:
MSE: 1.8287626291987034
Bronze Medals Model Evaluation:
MSE: 2.185780634244944
```

```
c:\Users\Ziqi\Documents\Python\2025-MCM-C\new_env\Lib\site-
packages\numpy\ma\core.py:2892: RuntimeWarning: invalid value encountered in
cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

```
Total Medals Model Evaluation:
MSE: 140.75342068015084
Gold Medals Model Evaluation:
MSE: 18.508602150537634
```

```
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 15.818427348076781
Total Medals Model Evaluation:
MSE: 142.85282353556897
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.57628960602166
Total Medals Model Evaluation:
MSE: 122.58495074211991
Gold Medals Model Evaluation:
MSE: 17.402119733106872
Silver Medals Model Evaluation:
MSE: 14.068511301831862
Bronze Medals Model Evaluation:
MSE: 12.350547132239438
Total Medals Model Evaluation:
MSE: 117.61289573977655
Gold Medals Model Evaluation:
MSE: 15.07693219210729
Silver Medals Model Evaluation:
MSE: 13.564828598561522
Bronze Medals Model Evaluation:
MSE: 14.238776584657261
Total Medals Model Evaluation:
MSE: 129.11978398983476
Gold Medals Model Evaluation:
MSE: 18.434324148292763
Silver Medals Model Evaluation:
MSE: 14.228132612839607
Bronze Medals Model Evaluation:
MSE: 14.471448380309658
Total Medals Model Evaluation:
```

```
MSE: 124.98686516532699
Gold Medals Model Evaluation:
MSE: 16.651968965013406
Silver Medals Model Evaluation:
MSE: 13.778019188979943
Bronze Medals Model Evaluation:
MSE: 14.139838236770917
Total Medals Model Evaluation:
MSE: 358.80440175013155
Gold Medals Model Evaluation:
MSE: 20.563497436255997
Silver Medals Model Evaluation:
MSE: 14.728424680409017
Bronze Medals Model Evaluation:
MSE: 62.151548453599624
Total Medals Model Evaluation:
MSE: 121.52328745335026
Gold Medals Model Evaluation:
MSE: 17.08826741405428
Silver Medals Model Evaluation:
MSE: 8.456780684764317
Bronze Medals Model Evaluation:
MSE: 13.800333944030273
Total Medals Model Evaluation:
MSE: 121.28168772713842
Gold Medals Model Evaluation:
MSE: 17.188011597117296
Silver Medals Model Evaluation:
MSE: 12.243500658956734
Bronze Medals Model Evaluation:
MSE: 13.01291421240365
Total Medals Model Evaluation:
MSE: 135.42778365851981
Gold Medals Model Evaluation:
MSE: 16.491949999900445
Silver Medals Model Evaluation:
MSE: 15.715501421589924
```

```
Bronze Medals Model Evaluation:
MSE: 15.425882914340793
Total Medals Model Evaluation:
MSE: 136.0125892999167
Gold Medals Model Evaluation:
MSE: 17.489983416107265
Silver Medals Model Evaluation:
MSE: 14.761046310429055
Bronze Medals Model Evaluation:
MSE: 16.594110127403372
Total Medals Model Evaluation:
MSE: 141.87124104542065
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.243400642547297
Total Medals Model Evaluation:
MSE: 120.71016681843432
Gold Medals Model Evaluation:
MSE: 15.787307093399031
Silver Medals Model Evaluation:
MSE: 14.19017143798728
Bronze Medals Model Evaluation:
MSE: 13.257024386223838
Total Medals Model Evaluation:
MSE: 147.57950644741916
Gold Medals Model Evaluation:
MSE: 18.25579859115117
Silver Medals Model Evaluation:
MSE: 18.632453573025572
Bronze Medals Model Evaluation:
MSE: 14.94270892045927
Total Medals Model Evaluation:
MSE: 139.6394026306558
Gold Medals Model Evaluation:
```

```
MSE: 17.767295402724574
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.460522670790425
Total Medals Model Evaluation:
MSE: 139.35187215005126
Gold Medals Model Evaluation:
MSE: 17.88929933233542
Silver Medals Model Evaluation:
MSE: 14.974081675585804
Bronze Medals Model Evaluation:
MSE: 16.854725073108664
Total Medals Model Evaluation:
MSE: 101.07380992388109
Gold Medals Model Evaluation:
MSE: 15.216573986366383
Silver Medals Model Evaluation:
MSE: 10.614158888550097
Bronze Medals Model Evaluation:
MSE: 17.021541084900498
Total Medals Model Evaluation:
MSE: 137.91288893522758
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 15.049108389021708
Total Medals Model Evaluation:
MSE: 61.34089209055521
Gold Medals Model Evaluation:
MSE: 10.549011817980551
Silver Medals Model Evaluation:
MSE: 7.605056533916365
Bronze Medals Model Evaluation:
MSE: 7.2707636253885966
```

```
Total Medals Model Evaluation:
MSE: 142.02179006561525
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.237062975312316
Total Medals Model Evaluation:
MSE: 139.3351760873514
Gold Medals Model Evaluation:
MSE: 18.035994408118114
Silver Medals Model Evaluation:
MSE: 14.73899526655078
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 143.3969044824103
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.724709920071003
Total Medals Model Evaluation:
MSE: 139.34559437757136
Gold Medals Model Evaluation:
MSE: 17.214406951374524
Silver Medals Model Evaluation:
MSE: 15.212162587307901
Bronze Medals Model Evaluation:
MSE: 16.490335383368734
Total Medals Model Evaluation:
MSE: 137.43044956221405
Gold Medals Model Evaluation:
MSE: 14.082064906186895
Silver Medals Model Evaluation:
```

```
MSE: 18.99416929276979
Bronze Medals Model Evaluation:
MSE: 13.800402129408212
Total Medals Model Evaluation:
MSE: 142.49664552351146
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.023011600995122
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 133.4642191773885
Gold Medals Model Evaluation:
MSE: 17.355501671116315
Silver Medals Model Evaluation:
MSE: 13.540156929741585
Bronze Medals Model Evaluation:
MSE: 15.426952193809083
Total Medals Model Evaluation:
MSE: 1059.2390573782006
Gold Medals Model Evaluation:
MSE: 217.9136816957673
Silver Medals Model Evaluation:
MSE: 65.1023815429746
Bronze Medals Model Evaluation:
MSE: 61.27728020539144
Total Medals Model Evaluation:
MSE: 120.7995675082242
Gold Medals Model Evaluation:
MSE: 15.957535261561022
Silver Medals Model Evaluation:
MSE: 14.024219968829057
Bronze Medals Model Evaluation:
MSE: 12.974846857613437
Total Medals Model Evaluation:
MSE: 114.4462319617079
```

```
Gold Medals Model Evaluation:
MSE: 14.065858206775063
Silver Medals Model Evaluation:
MSE: 12.038611154448432
Bronze Medals Model Evaluation:
MSE: 12.862802915832578
Total Medals Model Evaluation:
MSE: 134.83039375503853
Gold Medals Model Evaluation:
MSE: 18.518730006912765
Silver Medals Model Evaluation:
MSE: 15.35066484663198
Bronze Medals Model Evaluation:
MSE: 14.726264824957381
Total Medals Model Evaluation:
MSE: 123.04308202795725
Gold Medals Model Evaluation:
MSE: 16.378520776721903
Silver Medals Model Evaluation:
MSE: 13.313374606708491
Bronze Medals Model Evaluation:
MSE: 14.652005754387616
Total Medals Model Evaluation:
MSE: 92.45289568036877
Gold Medals Model Evaluation:
MSE: 17.45292206229233
Silver Medals Model Evaluation:
MSE: 9.54926623257859
Bronze Medals Model Evaluation:
MSE: 9.575333970581806
Total Medals Model Evaluation:
MSE: 141.5976571749477
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.763930751060396
Bronze Medals Model Evaluation:
```

```
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 124.00918192995411
Gold Medals Model Evaluation:
MSE: 15.70584489021724
Silver Medals Model Evaluation:
MSE: 13.685500024925961
Bronze Medals Model Evaluation:
MSE: 14.075050089775988
Total Medals Model Evaluation:
MSE: 135.07712626389417
Gold Medals Model Evaluation:
MSE: 16.407976076624227
Silver Medals Model Evaluation:
MSE: 14.712657467624638
Bronze Medals Model Evaluation:
MSE: 18.195535976693513
Total Medals Model Evaluation:
MSE: 141.4524764745244
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.06321729731559
Total Medals Model Evaluation:
MSE: 143.3969044824103
Gold Medals Model Evaluation:
MSE: 18.35809521683373
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 132.9917902552506
Gold Medals Model Evaluation:
MSE: 17.333133489644688
```

```
Silver Medals Model Evaluation:
MSE: 14.774083079468037
Bronze Medals Model Evaluation:
MSE: 14.888124326409718
Total Medals Model Evaluation:
MSE: 131.57558257428238
Gold Medals Model Evaluation:
MSE: 17.577953646470387
Silver Medals Model Evaluation:
MSE: 13.83418224110092
Bronze Medals Model Evaluation:
MSE: 15.704402378295871
Total Medals Model Evaluation:
MSE: 125.64458241245921
Gold Medals Model Evaluation:
MSE: 17.677077047051263
Silver Medals Model Evaluation:
MSE: 13.565503849149154
Bronze Medals Model Evaluation:
MSE: 14.639010014660908
Total Medals Model Evaluation:
MSE: 142.6711241751874
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.53910115819738
Total Medals Model Evaluation:
MSE: 119.74260543786875
Gold Medals Model Evaluation:
MSE: 16.40745144714485
Silver Medals Model Evaluation:
MSE: 13.848489533202711
Bronze Medals Model Evaluation:
MSE: 12.920595633059174
Total Medals Model Evaluation:
```

```
MSE: 119.64131817384899
Gold Medals Model Evaluation:
MSE: 15.789496425350672
Silver Medals Model Evaluation:
MSE: 13.436402389671295
Bronze Medals Model Evaluation:
MSE: 12.054728090609537
Total Medals Model Evaluation:
MSE: 132.08755741712528
Gold Medals Model Evaluation:
MSE: 17.16391235187512
Silver Medals Model Evaluation:
MSE: 14.09656418567718
Bronze Medals Model Evaluation:
MSE: 15.291661971328502
Total Medals Model Evaluation:
MSE: 141.42012674221036
Gold Medals Model Evaluation:
MSE: 18.33465279574127
Silver Medals Model Evaluation:
MSE: 14.846908790952323
Bronze Medals Model Evaluation:
MSE: 16.789748792221168
Total Medals Model Evaluation:
MSE: 71.46401099844464
Gold Medals Model Evaluation:
MSE: 10.874740656055133
Silver Medals Model Evaluation:
MSE: 8.503563416648833
Bronze Medals Model Evaluation:
MSE: 8.952931957009552
Total Medals Model Evaluation:
MSE: 430.3321074157528
Gold Medals Model Evaluation:
MSE: 51.211189027312564
Silver Medals Model Evaluation:
MSE: 60.169209903551945
```

```
Bronze Medals Model Evaluation:
MSE: 76.01822924563578
Total Medals Model Evaluation:
MSE: 141.9789980744976
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.847146113828398
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 128.99034889723183
Gold Medals Model Evaluation:
MSE: 16.59454281295756
Silver Medals Model Evaluation:
MSE: 13.569258233283135
Bronze Medals Model Evaluation:
MSE: 14.641360235746816
Total Medals Model Evaluation:
MSE: 958.9618946758767
Gold Medals Model Evaluation:
MSE: 89.13399482046448
Silver Medals Model Evaluation:
MSE: 101.97040236207555
Bronze Medals Model Evaluation:
MSE: 88.12731449605982
Total Medals Model Evaluation:
MSE: 137.77627507790044
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.505414889948891
Bronze Medals Model Evaluation:
MSE: 14.940825786098362
Total Medals Model Evaluation:
MSE: 423.5669933946924
Gold Medals Model Evaluation:
```

```
MSE: 36.31717644068732
Silver Medals Model Evaluation:
MSE: 47.29206765147535
Bronze Medals Model Evaluation:
MSE: 48.93329161520254
Total Medals Model Evaluation:
MSE: 145.86302352903726
Gold Medals Model Evaluation:
MSE: 19.99434790818509
Silver Medals Model Evaluation:
MSE: 23.392654973091872
Bronze Medals Model Evaluation:
MSE: 17.401896702261702
Total Medals Model Evaluation:
MSE: 139.87656568615063
Gold Medals Model Evaluation:
MSE: 18.441185924955366
Silver Medals Model Evaluation:
MSE: 13.53636457150377
Bronze Medals Model Evaluation:
MSE: 16.126925245552552
Total Medals Model Evaluation:
MSE: 139.00450093060985
Gold Medals Model Evaluation:
MSE: 18.257099882233568
Silver Medals Model Evaluation:
MSE: 15.39002164701618
Bronze Medals Model Evaluation:
MSE: 16.571470586552312
Total Medals Model Evaluation:
MSE: 142.2275439007511
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.306406148992405
```

```
Total Medals Model Evaluation:
MSE: 137.46119030179804
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.143672283592648
Bronze Medals Model Evaluation:
MSE: 16.943613958528637
Total Medals Model Evaluation:
MSE: 132.78970479872174
Gold Medals Model Evaluation:
MSE: 17.4729361979997
Silver Medals Model Evaluation:
MSE: 15.245944176253085
Bronze Medals Model Evaluation:
MSE: 15.606736971660961
Total Medals Model Evaluation:
MSE: 189.24244991334984
Gold Medals Model Evaluation:
MSE: 37.85571514365352
Silver Medals Model Evaluation:
MSE: 27.82685424008854
Bronze Medals Model Evaluation:
MSE: 31.511467693917236
Total Medals Model Evaluation:
MSE: 137.7564704547477
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.615002026414722
Bronze Medals Model Evaluation:
MSE: 15.55369268783078
Total Medals Model Evaluation:
MSE: 141.46316516955667
Gold Medals Model Evaluation:
MSE: 17.520532117597618
Silver Medals Model Evaluation:
```

```
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 15.779644445974679
Total Medals Model Evaluation:
MSE: 137.7791441335767
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 13.761335055679432
Bronze Medals Model Evaluation:
MSE: 15.26846197606054
Total Medals Model Evaluation:
MSE: 128.71381913191783
Gold Medals Model Evaluation:
MSE: 17.69731813118513
Silver Medals Model Evaluation:
MSE: 14.284207697126444
Bronze Medals Model Evaluation:
MSE: 13.743768023631464
Total Medals Model Evaluation:
MSE: 128.79591029344485
Gold Medals Model Evaluation:
MSE: 16.960412414942144
Silver Medals Model Evaluation:
MSE: 14.181781246488878
Bronze Medals Model Evaluation:
MSE: 14.09178375897888
Total Medals Model Evaluation:
MSE: 114.8451674781899
Gold Medals Model Evaluation:
MSE: 15.416523229377445
Silver Medals Model Evaluation:
MSE: 13.84757605170232
Bronze Medals Model Evaluation:
MSE: 13.857348757217634
Total Medals Model Evaluation:
MSE: 142.97759343948675
```

```
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.562434910371756
Total Medals Model Evaluation:
MSE: 125.65896596947744
Gold Medals Model Evaluation:
MSE: 16.561858866459158
Silver Medals Model Evaluation:
MSE: 13.791116233875021
Bronze Medals Model Evaluation:
MSE: 12.911142548854563
Total Medals Model Evaluation:
MSE: 131.91750636022712
Gold Medals Model Evaluation:
MSE: 17.5655350180797
Silver Medals Model Evaluation:
MSE: 13.81521282450842
Bronze Medals Model Evaluation:
MSE: 15.049765848257437
Total Medals Model Evaluation:
MSE: 206.45664606820688
Gold Medals Model Evaluation:
MSE: 43.468497500318506
Silver Medals Model Evaluation:
MSE: 20.675656069336327
Bronze Medals Model Evaluation:
MSE: 17.542365157074435
Total Medals Model Evaluation:
MSE: 136.13675874993626
Gold Medals Model Evaluation:
MSE: 17.779087314064153
Silver Medals Model Evaluation:
MSE: 14.921817966987119
Bronze Medals Model Evaluation:
```

```
MSE: 15.62986508350761
Total Medals Model Evaluation:
MSE: 111.17327072317192
Gold Medals Model Evaluation:
MSE: 14.261646986460654
Silver Medals Model Evaluation:
MSE: 11.315058124927397
Bronze Medals Model Evaluation:
MSE: 12.197801233909257
Total Medals Model Evaluation:
MSE: 319.9670015264475
Gold Medals Model Evaluation:
MSE: 31.05901598394929
Silver Medals Model Evaluation:
MSE: 38.39943386640468
Bronze Medals Model Evaluation:
MSE: 24.50816027411338
Total Medals Model Evaluation:
MSE: 136.15506462077178
Gold Medals Model Evaluation:
MSE: 18.447981328716264
Silver Medals Model Evaluation:
MSE: 14.707436911961903
Bronze Medals Model Evaluation:
MSE: 14.775797220374905
Total Medals Model Evaluation:
MSE: 127.30748648909412
Gold Medals Model Evaluation:
MSE: 16.634659806400304
Silver Medals Model Evaluation:
MSE: 12.609374066776969
Bronze Medals Model Evaluation:
MSE: 13.842519328445977
Total Medals Model Evaluation:
MSE: 115.70549389757369
Gold Medals Model Evaluation:
MSE: 15.432247519733274
```

```
Silver Medals Model Evaluation:
MSE: 12.93039784822176
Bronze Medals Model Evaluation:
MSE: 12.110291854474921
Total Medals Model Evaluation:
MSE: 139.74747766272486
Gold Medals Model Evaluation:
MSE: 18.432234553194622
Silver Medals Model Evaluation:
MSE: 14.848137494719394
Bronze Medals Model Evaluation:
MSE: 16.22947823700332
Total Medals Model Evaluation:
MSE: 140.11930942429288
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 15.662211468322043
Total Medals Model Evaluation:
MSE: 133.10545942321878
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 13.96414535656642
Bronze Medals Model Evaluation:
MSE: 14.855834224010652
Total Medals Model Evaluation:
MSE: 127.56652924353895
Gold Medals Model Evaluation:
MSE: 17.353212941067035
Silver Medals Model Evaluation:
MSE: 13.276219113771015
Bronze Medals Model Evaluation:
MSE: 14.97416532796742
Total Medals Model Evaluation:
```

```
MSE: 137.71314003465486
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 13.785144539581065
Bronze Medals Model Evaluation:
MSE: 14.956013439102191
Total Medals Model Evaluation:
MSE: 125.57589470566808
Gold Medals Model Evaluation:
MSE: 16.80106889405876
Silver Medals Model Evaluation:
MSE: 14.256961497853823
Bronze Medals Model Evaluation:
MSE: 13.590768783485288
Total Medals Model Evaluation:
MSE: 137.07256891101002
Gold Medals Model Evaluation:
MSE: 16.767026971472696
Silver Medals Model Evaluation:
MSE: 15.144168640389212
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 141.87124104542065
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.243400642547297
Total Medals Model Evaluation:
MSE: 132.9885280255021
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.025048059925746
```

```
Bronze Medals Model Evaluation:
MSE: 15.550514312800422
Total Medals Model Evaluation:
MSE: 142.41286633228276
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.391406811866954
Total Medals Model Evaluation:
MSE: 122.88854761340191
Gold Medals Model Evaluation:
MSE: 17.41744942829359
Silver Medals Model Evaluation:
MSE: 13.266356384500956
Bronze Medals Model Evaluation:
MSE: 13.925095231808378
Total Medals Model Evaluation:
MSE: 130.3344856428949
Gold Medals Model Evaluation:
MSE: 23.178505781786185
Silver Medals Model Evaluation:
MSE: 13.800624435420994
Bronze Medals Model Evaluation:
MSE: 17.20539461309142
Total Medals Model Evaluation:
MSE: 135.3465064408034
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.570818820045854
Bronze Medals Model Evaluation:
MSE: 14.808219510499175
Total Medals Model Evaluation:
MSE: 133.5653302599893
Gold Medals Model Evaluation:
```

```
MSE: 18.243516810099045
Silver Medals Model Evaluation:
MSE: 14.158554731579681
Bronze Medals Model Evaluation:
MSE: 14.799591778455794
Total Medals Model Evaluation:
MSE: 141.9789980744976
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.847146113828398
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 128.53279094919372
Gold Medals Model Evaluation:
MSE: 15.163652603577788
Silver Medals Model Evaluation:
MSE: 14.394880057608226
Bronze Medals Model Evaluation:
MSE: 13.801221794303313
Total Medals Model Evaluation:
MSE: 139.33559570427354
Gold Medals Model Evaluation:
MSE: 17.19828224455516
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.79758977136095
Total Medals Model Evaluation:
MSE: 136.40619780497929
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 13.509131303570495
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
```

```
Total Medals Model Evaluation:
MSE: 127.97074906745453
Gold Medals Model Evaluation:
MSE: 13.446127503491082
Silver Medals Model Evaluation:
MSE: 9.709605096279564
Bronze Medals Model Evaluation:
MSE: 22.473188101012962
Total Medals Model Evaluation:
MSE: 141.41526602534674
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.719530727107722
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 90.82187183131279
Gold Medals Model Evaluation:
MSE: 16.46292436843005
Silver Medals Model Evaluation:
MSE: 9.861979940395406
Bronze Medals Model Evaluation:
MSE: 10.557522512299439
Total Medals Model Evaluation:
MSE: 138.90460745989
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.334087824543742
Bronze Medals Model Evaluation:
MSE: 16.269595238379605
Total Medals Model Evaluation:
MSE: 130.24560627768508
Gold Medals Model Evaluation:
MSE: 17.046973306958115
Silver Medals Model Evaluation:
```

```
MSE: 13.388307855682035
Bronze Medals Model Evaluation:
MSE: 12.61663651919816
Total Medals Model Evaluation:
MSE: 117.16959512126158
Gold Medals Model Evaluation:
MSE: 15.025275128218055
Silver Medals Model Evaluation:
MSE: 13.17042773656496
Bronze Medals Model Evaluation:
MSE: 13.332845766228688
Total Medals Model Evaluation:
MSE: 142.02253023859032
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.879037674693903
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 113.12692967068391
Gold Medals Model Evaluation:
MSE: 12.567166176786662
Silver Medals Model Evaluation:
MSE: 12.097676294688442
Bronze Medals Model Evaluation:
MSE: 14.28169880610954
Total Medals Model Evaluation:
MSE: 137.73642807920788
Gold Medals Model Evaluation:
MSE: 17.805580306523346
Silver Medals Model Evaluation:
MSE: 14.831100015655679
Bronze Medals Model Evaluation:
MSE: 15.91288272227091
Total Medals Model Evaluation:
MSE: 137.56921310056828
```

```
Gold Medals Model Evaluation:
MSE: 17.115933522281484
Silver Medals Model Evaluation:
MSE: 15.028158419773177
Bronze Medals Model Evaluation:
MSE: 15.923797272210706
Total Medals Model Evaluation:
MSE: 142.67113750188773
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.087768574724542
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 137.6247990519643
Gold Medals Model Evaluation:
MSE: 18.015592930757705
Silver Medals Model Evaluation:
MSE: 13.891151909137237
Bronze Medals Model Evaluation:
MSE: 16.222980978212686
Total Medals Model Evaluation:
MSE: 130.93607685319475
Gold Medals Model Evaluation:
MSE: 17.55580040628031
Silver Medals Model Evaluation:
MSE: 14.047186837285457
Bronze Medals Model Evaluation:
MSE: 14.64044132805513
Total Medals Model Evaluation:
MSE: 141.09046314385375
Gold Medals Model Evaluation:
MSE: 15.615346401008328
Silver Medals Model Evaluation:
MSE: 9.488869177705215
Bronze Medals Model Evaluation:
```

```
MSE: 14.394994150607133
Total Medals Model Evaluation:
MSE: 128.74027775749815
Gold Medals Model Evaluation:
MSE: 16.882727234912707
Silver Medals Model Evaluation:
MSE: 13.574545565240095
Bronze Medals Model Evaluation:
MSE: 15.263664923091076
Total Medals Model Evaluation:
MSE: 136.06583345801198
Gold Medals Model Evaluation:
MSE: 18.282738631898457
Silver Medals Model Evaluation:
MSE: 15.022675106118193
Bronze Medals Model Evaluation:
MSE: 15.207178453684888
Total Medals Model Evaluation:
MSE: 137.2541636466537
Gold Medals Model Evaluation:
MSE: 18.2053941034054
Silver Medals Model Evaluation:
MSE: 14.758058418403055
Bronze Medals Model Evaluation:
MSE: 15.314030286556767
Total Medals Model Evaluation:
MSE: 319.3562992481888
Gold Medals Model Evaluation:
MSE: 32.45522085934934
Silver Medals Model Evaluation:
MSE: 44.730654201336975
Bronze Medals Model Evaluation:
MSE: 34.2966550913708
Total Medals Model Evaluation:
MSE: 143.3969044824103
Gold Medals Model Evaluation:
MSE: 18.508602150537634
```

```
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.724709920071003
Total Medals Model Evaluation:
MSE: 69.0968085735777
Gold Medals Model Evaluation:
MSE: 11.332462973290282
Silver Medals Model Evaluation:
MSE: 8.005806236692317
Bronze Medals Model Evaluation:
MSE: 6.9620993715241
Total Medals Model Evaluation:
MSE: 731.6196669968692
Gold Medals Model Evaluation:
MSE: 155.5471407104923
Silver Medals Model Evaluation:
MSE: 62.76565025740928
Bronze Medals Model Evaluation:
MSE: 34.14865917254159
Total Medals Model Evaluation:
MSE: 142.85282353556897
Gold Medals Model Evaluation:
MSE: 18.35809521683373
Silver Medals Model Evaluation:
MSE: 15.297862980408816
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 141.97211501849736
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.868688356632255
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
```

```
MSE: 140.1860723417631
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.89386352183329
Bronze Medals Model Evaluation:
MSE: 16.3834643829581
Total Medals Model Evaluation:
MSE: 139.81409086366125
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.990330475436913
Bronze Medals Model Evaluation:
MSE: 15.767712838466531
Total Medals Model Evaluation:
MSE: 141.41526602534674
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.719530727107722
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 115.50343250893141
Gold Medals Model Evaluation:
MSE: 16.76679836055545
Silver Medals Model Evaluation:
MSE: 12.520437626804206
Bronze Medals Model Evaluation:
MSE: 13.279401321327365
Total Medals Model Evaluation:
MSE: 139.60382040217826
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.212751059806186
```

```
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 137.50041052809894
Gold Medals Model Evaluation:
MSE: 16.404693533998483
Silver Medals Model Evaluation:
MSE: 14.816481700720088
Bronze Medals Model Evaluation:
MSE: 14.844592006576514
Total Medals Model Evaluation:
MSE: 126.16630244990345
Gold Medals Model Evaluation:
MSE: 15.071809809910627
Silver Medals Model Evaluation:
MSE: 13.40853226322626
Bronze Medals Model Evaluation:
MSE: 12.597443131742804
Total Medals Model Evaluation:
MSE: 127.25009954091041
Gold Medals Model Evaluation:
MSE: 16.926480710184762
Silver Medals Model Evaluation:
MSE: 13.85025155682014
Bronze Medals Model Evaluation:
MSE: 15.943679690749073
Total Medals Model Evaluation:
MSE: 122.22429946498771
Gold Medals Model Evaluation:
MSE: 16.772517866978095
Silver Medals Model Evaluation:
MSE: 11.583161634916555
Bronze Medals Model Evaluation:
MSE: 13.999412456879316
Total Medals Model Evaluation:
MSE: 251.61842026089872
Gold Medals Model Evaluation:
```

```
MSE: 33.140948130894614
Silver Medals Model Evaluation:
MSE: 19.51450542095045
Bronze Medals Model Evaluation:
MSE: 26.495273911833902
Total Medals Model Evaluation:
MSE: 105.98040261668906
Gold Medals Model Evaluation:
MSE: 15.69860768926409
Silver Medals Model Evaluation:
MSE: 15.896625274102412
Bronze Medals Model Evaluation:
MSE: 14.196455796077425
Total Medals Model Evaluation:
MSE: 141.82337932321212
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.772864172316424
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 141.97211501849736
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.868688356632255
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 138.76663959741063
Gold Medals Model Evaluation:
MSE: 18.348737894531723
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 15.32975484106139
```

```
Total Medals Model Evaluation:
MSE: 97.86561132643006
Gold Medals Model Evaluation:
MSE: 9.939944801859536
Silver Medals Model Evaluation:
MSE: 14.530194606984049
Bronze Medals Model Evaluation:
MSE: 11.830940058907796
Total Medals Model Evaluation:
MSE: 92.3029029187137
Gold Medals Model Evaluation:
MSE: 16.731164303443318
Silver Medals Model Evaluation:
MSE: 10.859344784883668
Bronze Medals Model Evaluation:
MSE: 9.421127814825896
Total Medals Model Evaluation:
MSE: 138.59047770663238
Gold Medals Model Evaluation:
MSE: 17.75290432646527
Silver Medals Model Evaluation:
MSE: 15.428913291547145
Bronze Medals Model Evaluation:
MSE: 15.990374443350733
Total Medals Model Evaluation:
MSE: 144.00537634408602
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 139.89583852281243
Gold Medals Model Evaluation:
MSE: 18.49097762254012
Silver Medals Model Evaluation:
```

```
MSE: 15.364905985556241
Bronze Medals Model Evaluation:
MSE: 15.351029718397214
Total Medals Model Evaluation:
MSE: 138.15749736287572
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 13.867874875710406
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 124.52643597291664
Gold Medals Model Evaluation:
MSE: 16.54440089048447
Silver Medals Model Evaluation:
MSE: 13.363132120760671
Bronze Medals Model Evaluation:
MSE: 14.387371590097464
Total Medals Model Evaluation:
MSE: 142.41286633228276
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.391406811866954
Total Medals Model Evaluation:
MSE: 141.23447219747518
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.637407466189542
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 133.09967521980215
```

```
Gold Medals Model Evaluation:
MSE: 18.044336810714906
Silver Medals Model Evaluation:
MSE: 14.63179011803488
Bronze Medals Model Evaluation:
MSE: 14.792626154088062
Total Medals Model Evaluation:
MSE: 133.92583877046084
Gold Medals Model Evaluation:
MSE: 17.53268434088007
Silver Medals Model Evaluation:
MSE: 14.570375991830685
Bronze Medals Model Evaluation:
MSE: 12.342067246971313
Total Medals Model Evaluation:
MSE: 118.09401599766086
Gold Medals Model Evaluation:
MSE: 15.895064846442166
Silver Medals Model Evaluation:
MSE: 13.026615279307633
Bronze Medals Model Evaluation:
MSE: 13.59106185654135
Total Medals Model Evaluation:
MSE: 142.02253023859032
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.879037674693903
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
MSE: 136.1892923916188
Gold Medals Model Evaluation:
MSE: 17.199064555728967
Silver Medals Model Evaluation:
MSE: 14.479878408930748
Bronze Medals Model Evaluation:
```

```
MSE: 16.335528680973404
Total Medals Model Evaluation:
MSE: 147.09534075675091
Gold Medals Model Evaluation:
MSE: 17.525681307294043
Silver Medals Model Evaluation:
MSE: 11.920742772536746
Bronze Medals Model Evaluation:
MSE: 21.738538680584604
Total Medals Model Evaluation:
MSE: 801.8796592665005
Gold Medals Model Evaluation:
MSE: 127.08414639875762
Silver Medals Model Evaluation:
MSE: 92.8079718222651
Bronze Medals Model Evaluation:
MSE: 54.35878860000859
Total Medals Model Evaluation:
MSE: 139.97747351836793
Gold Medals Model Evaluation:
MSE: 18.31775223173615
Silver Medals Model Evaluation:
MSE: 15.480645161290322
Bronze Medals Model Evaluation:
MSE: 16.10785772122929
Total Medals Model Evaluation:
MSE: 7955.440914422284
Gold Medals Model Evaluation:
MSE: 1287.7319429982033
Silver Medals Model Evaluation:
MSE: 886.8534870824524
Bronze Medals Model Evaluation:
MSE: 502.76065810972136
Total Medals Model Evaluation:
MSE: 136.6027273902471
Gold Medals Model Evaluation:
MSE: 17.93912512270018
```

```
Silver Medals Model Evaluation:
MSE: 14.284246468040052
Bronze Medals Model Evaluation:
MSE: 15.606827977360092
Total Medals Model Evaluation:
MSE: 118.05266389190098
Gold Medals Model Evaluation:
MSE: 17.147729741230755
Silver Medals Model Evaluation:
MSE: 14.076711138301276
Bronze Medals Model Evaluation:
MSE: 13.454165858708564
Total Medals Model Evaluation:
MSE: 130.08125955237986
Gold Medals Model Evaluation:
MSE: 18.012003816764853
Silver Medals Model Evaluation:
MSE: 12.81432007096309
Bronze Medals Model Evaluation:
MSE: 14.951866215159672
Total Medals Model Evaluation:
MSE: 138.9081210348589
Gold Medals Model Evaluation:
MSE: 17.97066037439278
Silver Medals Model Evaluation:
MSE: 14.325087813911944
Bronze Medals Model Evaluation:
MSE: 16.856653003907994
Total Medals Model Evaluation:
MSE: 141.41526602534674
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.719530727107722
Bronze Medals Model Evaluation:
MSE: 16.93763440860215
Total Medals Model Evaluation:
```

```
MSE: 138.59047770663238
Gold Medals Model Evaluation:
MSE: 18.508602150537634
Silver Medals Model Evaluation:
MSE: 14.379276547459726
Bronze Medals Model Evaluation:
MSE: 15.834178407042506
Total Medals Model Evaluation:
MSE: 128.59097654913361
Gold Medals Model Evaluation:
MSE: 16.939784606300343
Silver Medals Model Evaluation:
MSE: 13.285004550045176
Bronze Medals Model Evaluation:
MSE: 16.012281431777485
               NOC  Total_Predicted  Gold_Predicted  Silver_Predicted  \
0      Afghanistan                1               0                 0
1          Albania                2               0                 0
2          Algeria                4               2                 0
3        Argentina                3               1                 1
4          Armenia                4               0                 3
..             …              …             …               …
150      Venezuela                3               0                 2
151        Vietnam                1               0                 1
152  VirginIslands                0               0                 0
153         Zambia                1               0                 0
154       Zimbabwe                3               1                 2

     Bronze_Predicted
0                   1
1                   2
2                   2
3                   1
4                   1
..                …
150                 0
151                 0
```

```
152                   0
153                   1
154                   0

[155 rows x 5 columns]
```

3.1.2 非超参数优化模型

[71]:

```python
import pandas as pd
import numpy as np
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder

# 加载数据
data = pd.read_csv('Generated\\summerOly_medal_counts_processed.csv')

# 数据预处理
# 将国家代码转换为数值标签
label_encoder = LabelEncoder()
data['NOC'] = label_encoder.fit_transform(data['NOC'])

# 处理缺失值
data = data.fillna(0)

# 创建特征：前一届奥运会的奖牌总数、金牌数、银牌数、铜牌数
data['Prev_Total'] = data.groupby('NOC')['Total'].shift(1)
data['Prev_Gold'] = data.groupby('NOC')['Gold'].shift(1)
data['Prev_Silver'] = data.groupby('NOC')['Silver'].shift(1)
data['Prev_Bronze'] = data.groupby('NOC')['Bronze'].shift(1)

# 填充缺失值
data['Prev_Total'] = data['Prev_Total'].fillna(0)
data['Prev_Gold'] = data['Prev_Gold'].fillna(0)
data['Prev_Silver'] = data['Prev_Silver'].fillna(0)
data['Prev_Bronze'] = data['Prev_Bronze'].fillna(0)
```

```
# 选择特征和目标变量
features = data[['Year', 'NOC', 'Prev_Total', 'Prev_Gold', 'Prev_Silver',␣
 ↪'Prev_Bronze']]
target_total = data['Total']
target_gold = data['Gold']
target_silver = data['Silver']
target_bronze = data['Bronze']

# 划分训练集和测试集
X_train_total, X_test_total, y_train_total, y_test_total =␣
 ↪train_test_split(features, target_total, test_size=0.2, random_state=42)
X_train_gold, X_test_gold, y_train_gold, y_test_gold =␣
 ↪train_test_split(features, target_gold, test_size=0.2, random_state=42)
X_train_silver, X_test_silver, y_train_silver, y_test_silver =␣
 ↪train_test_split(features, target_silver, test_size=0.2, random_state=42)
X_train_bronze, X_test_bronze, y_train_bronze, y_test_bronze =␣
 ↪train_test_split(features, target_bronze, test_size=0.2, random_state=42)

# 定义 XGBoost 模型
model_total = XGBRegressor(objective='reg:squarederror', random_state=42)
model_gold = XGBRegressor(objective='reg:squarederror', random_state=42)
model_silver = XGBRegressor(objective='reg:squarederror', random_state=42)
model_bronze = XGBRegressor(objective='reg:squarederror', random_state=42)

# 超参数优化
param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.7, 0.8, 0.9]
}

# 使用 GridSearchCV 进行超参数优化
grid_search_total = GridSearchCV(estimator=model_total, param_grid=param_grid,␣
 ↪cv=3, scoring='neg_mean_squared_error', n_jobs=-1)
```

```
grid_search_gold = GridSearchCV(estimator=model_gold, param_grid=param_grid,␣
 ↪cv=3, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_silver = GridSearchCV(estimator=model_silver,␣
 ↪param_grid=param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_bronze = GridSearchCV(estimator=model_bronze,␣
 ↪param_grid=param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1)

# 训练模型
grid_search_total.fit(X_train_total, y_train_total)
grid_search_gold.fit(X_train_gold, y_train_gold)
grid_search_silver.fit(X_train_silver, y_train_silver)
grid_search_bronze.fit(X_train_bronze, y_train_bronze)

# 获取最佳模型
best_model_total = grid_search_total.best_estimator_
best_model_gold = grid_search_gold.best_estimator_
best_model_silver = grid_search_silver.best_estimator_
best_model_bronze = grid_search_bronze.best_estimator_

# 评估模型
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f'MSE: {mse}')
    return y_pred

print("Total Medals Model Evaluation:")
evaluate_model(best_model_total, X_test_total, y_test_total)

print("Gold Medals Model Evaluation:")
evaluate_model(best_model_gold, X_test_gold, y_test_gold)

print("Silver Medals Model Evaluation:")
evaluate_model(best_model_silver, X_test_silver, y_test_silver)

print("Bronze Medals Model Evaluation:")
```

```
evaluate_model(best_model_bronze, X_test_bronze, y_test_bronze)

# 为每个国家单独训练模型
country_models = {}
for country in data['NOC'].unique():
    country_data = data[data['NOC'] == country]
    if len(country_data) > 1:  # 确保每个国家至少有两条记录
        country_features = country_data[['Year','NOC', 'Prev_Total',␣
 ↪'Prev_Gold', 'Prev_Silver', 'Prev_Bronze']]
        country_target_total = country_data['Total']
        country_target_gold = country_data['Gold']
        country_target_silver = country_data['Silver']
        country_target_bronze = country_data['Bronze']

        country_model_total = XGBRegressor(objective='reg:squarederror',␣
 ↪random_state=42)
        country_model_gold = XGBRegressor(objective='reg:squarederror',␣
 ↪random_state=42)
        country_model_silver = XGBRegressor(objective='reg:squarederror',␣
 ↪random_state=42)
        country_model_bronze = XGBRegressor(objective='reg:squarederror',␣
 ↪random_state=42)

        country_model_total.fit(country_features, country_target_total)
        country_model_gold.fit(country_features, country_target_gold)
        country_model_silver.fit(country_features, country_target_silver)
        country_model_bronze.fit(country_features, country_target_bronze)

        country_models[country] = {
            'total': country_model_total,
            'gold': country_model_gold,
            'silver': country_model_silver,
            'bronze': country_model_bronze
        }

# 预测 2028 年奥运会的奖牌数
```

```
next_year = 2028
predictions = []

for country in data['NOC'].unique():
    country_data = data[data['NOC'] == country]
    if len(country_data) > 1:
        prev_total = country_data['Total'].iloc[-1]
        prev_gold = country_data['Gold'].iloc[-1]
        prev_silver = country_data['Silver'].iloc[-1]
        prev_bronze = country_data['Bronze'].iloc[-1]

        next_data = pd.DataFrame({
            'Year': [next_year],
            'NOC': [country],  # 添加 NOC 列
            'Prev_Total': [prev_total],
            'Prev_Gold': [prev_gold],
            'Prev_Silver': [prev_silver],
            'Prev_Bronze': [prev_bronze]
        })

        # 使用单独模型预测
        total_pred = country_models[country]['total'].predict(next_data)
        gold_pred = country_models[country]['gold'].predict(next_data)
        silver_pred = country_models[country]['silver'].predict(next_data)
        bronze_pred = country_models[country]['bronze'].predict(next_data)

        # 使用整体模型预测
        total_pred_global = best_model_total.predict(next_data)
        gold_pred_global = best_model_gold.predict(next_data)
        silver_pred_global = best_model_silver.predict(next_data)
        bronze_pred_global = best_model_bronze.predict(next_data)

        # 根据数据量分配权重
        data_count = len(country_data)
        weight = min(data_count / 10, 1)  # 数据量越多，权重越高，但不超过 1
```

```python
        total_pred_combined = weight * total_pred + (1 - weight) * total_pred_global
        gold_pred_combined = weight * gold_pred + (1 - weight) * gold_pred_global
        silver_pred_combined = weight * silver_pred + (1 - weight) * silver_pred_global
        bronze_pred_combined = weight * bronze_pred + (1 - weight) * bronze_pred_global

        # 对预测结果取整
        total_pred_combined = round(total_pred_combined[0])
        gold_pred_combined = round(gold_pred_combined[0])
        silver_pred_combined = round(silver_pred_combined[0])
        bronze_pred_combined = round(bronze_pred_combined[0])

        predictions.append({
            'NOC': country,
            'Total_Predicted': total_pred_combined,
            'Gold_Predicted': gold_pred_combined,
            'Silver_Predicted': silver_pred_combined,
            'Bronze_Predicted': bronze_pred_combined
        })

# 将预测结果转换为 DataFrame
predictions_df = pd.DataFrame(predictions)

# 将 NOC 标签转换回国家代码
predictions_df['NOC'] = label_encoder.inverse_transform(predictions_df['NOC'])

# 输出预测结果
print(predictions_df)

# 保存预测结果到 CSV 文件
predictions_df.to_csv('Result\\2028_olympics_medal_predictions_2.csv', index=False)
```

```
Total Medals Model Evaluation:
MSE: 14.183948320207696
Gold Medals Model Evaluation:
MSE: 2.9414593946059715
Silver Medals Model Evaluation:
MSE: 1.8287626291987034
Bronze Medals Model Evaluation:
MSE: 2.185780634244944
               NOC  Total_Predicted  Gold_Predicted  Silver_Predicted  \
0      Afghanistan                1               0                 0
1          Albania                2               0                 0
2          Algeria                5               2                 0
3        Argentina                3               1                 1
4          Armenia                4               0                 3
..             …               …             …                 …
150      Venezuela                3               1                 2
151        Vietnam                1               0                 1
152   VirginIslands                0               0                 0
153         Zambia                1               0                 0
154       Zimbabwe                3               1                 2

     Bronze_Predicted
0                   1
1                   2
2                   2
3                   1
4                   1
..                …
150                 0
151                 0
152                 0
153                 1
154                 0

[155 rows x 5 columns]
```

3.1.3 评估预测区间

[72]:

```
import math

# 加载数据
data = pd.read_csv('Generated\\summerOly_medal_counts_processed.csv')
predictions_df = pd.read_csv('Result\\2028_olympics_medal_predictions_2.csv')

# 数据预处理
# 将国家代码转换为数值标签
label_encoder = LabelEncoder()
data['NOC2'] = data['NOC'].copy()
data['NOC'] = label_encoder.fit_transform(data['NOC'])
predictions_df['NOC2'] = predictions_df['NOC'].copy()
predictions_df['NOC'] = label_encoder.fit_transform(predictions_df['NOC'])

# 输出预测结果
print(predictions_df)

# 处理缺失值
data = data.fillna(0)

# 创建特征：前一届奥运会的奖牌总数、金牌数、银牌数、铜牌数
data['Prev_Total'] = data.groupby('NOC')['Total'].shift(1)
data['Prev_Gold'] = data.groupby('NOC')['Gold'].shift(1)
data['Prev_Silver'] = data.groupby('NOC')['Silver'].shift(1)
data['Prev_Bronze'] = data.groupby('NOC')['Bronze'].shift(1)

# 填充缺失值
data['Prev_Total'] = data['Prev_Total'].fillna(0)
data['Prev_Gold'] = data['Prev_Gold'].fillna(0)
data['Prev_Silver'] = data['Prev_Silver'].fillna(0)
data['Prev_Bronze'] = data['Prev_Bronze'].fillna(0)

def evaluate_model_2(model, X_test, y_test):
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f'MSE: {mse}')
    return mse

# 计算预测区间
def prediction_interval(model, X, y, confidence=0.95):
    # 评估区间
    interval_get = math.sqrt(int(evaluate_model_2(model, X, y)))
    preds = []
    for i in range(10):  # 进行 10 次预测以估计不确定性
        preds.append(model.predict(X))
    preds = np.array(preds)
    lower = np.percentile(preds, (1 - confidence) / 2 * 100, axis=0) -␣
 ↪round(interval_get/2)
    upper = np.percentile(preds, (1 + confidence) / 2 * 100, axis=0) +␣
 ↪round(interval_get/2)
    return lower, upper

# 计算每个国家的预测区间
prediction_intervals = []
for country in data['NOC'].unique():
    country_data = data[data['NOC'] == country]
    print(country, label_encoder.inverse_transform([country])[0])
    if len(country_data) > 1:
        prev_total = country_data['Total'].iloc[-1]
        prev_gold = country_data['Gold'].iloc[-1]
        prev_silver = country_data['Silver'].iloc[-1]
        prev_bronze = country_data['Bronze'].iloc[-1]

        next_data = pd.DataFrame({
            'Year': [next_year],
            'NOC': [country],
            'Prev_Total': [prev_total],
            'Prev_Gold': [prev_gold],
            'Prev_Silver': [prev_silver],
            'Prev_Bronze': [prev_bronze]
        })
        total_lower, total_upper =␣
↪prediction_interval(country_models[country]['total'], next_data,␣
↪country_data['Total'][country_data['Year']==2024], 0.95)
        gold_lower, gold_upper =␣
↪prediction_interval(country_models[country]['gold'], next_data,␣
↪country_data['Gold'][country_data['Year']==2024], 0.95)
        silver_lower, silver_upper =␣
↪prediction_interval(country_models[country]['silver'], next_data,␣
↪country_data['Silver'][country_data['Year']==2024], 0.95)
        bronze_lower, bronze_upper =␣
↪prediction_interval(country_models[country]['bronze'], next_data,␣
↪country_data['Bronze'][country_data['Year']==2024], 0.95)

        prediction_intervals.append({
            'NOC': country,
            'Total_Predicted': predictions_df.loc[predictions_df['NOC2'] ==␣
↪label_encoder.inverse_transform([country])[0], 'Total_Predicted'].values[0],
            'Total_Lower': round(total_lower[0]),
            'Total_Upper': round(total_upper[0]),
            'Gold_Predicted': predictions_df.loc[predictions_df['NOC2'] ==␣
↪label_encoder.inverse_transform([country])[0], 'Gold_Predicted'].values[0],
            'Gold_Lower': round(gold_lower[0]),
            'Gold_Upper': round(gold_upper[0]),
            'Silver_Predicted': predictions_df.loc[predictions_df['NOC2'] ==␣
↪label_encoder.inverse_transform([country])[0], 'Silver_Predicted'].values[0],
            'Silver_Lower': round(silver_lower[0]),
            'Silver_Upper': round(silver_upper[0]),
            'Bronze_Predicted': predictions_df.loc[predictions_df['NOC2'] ==␣
↪label_encoder.inverse_transform([country])[0], 'Bronze_Predicted'].values[0],
            'Bronze_Lower': round(bronze_lower[0]),
            'Bronze_Upper': round(bronze_upper[0])
        })

# 将预测区间转换为 DataFrame
prediction_intervals_df = pd.DataFrame(prediction_intervals)
```

```
# 将 NOC 标签转换回国家代码
prediction_intervals_df['NOC'] = label_encoder.
 ↪inverse_transform(prediction_intervals_df['NOC'])

# 输出预测区间
print(prediction_intervals_df)

# 保存预测区间到 CSV 文件
prediction_intervals_df.
 ↪to_csv('Result\\2028_olympics_medal_predictions_intervals.csv', index=False)
```

```
     NOC  Total_Predicted  Gold_Predicted  Silver_Predicted  Bronze_Predicted  \
0      0                1               0                 0                 1
1      1                2               0                 0                 2
2      2                5               2                 0                 2
3      3                3               1                 1                 1
4      4                4               0                 3                 1
..   …              …             …               …               …
150  150                3               1                 2                 0
151  151                1               0                 1                 0
152  152                0               0                 0                 0
153  153                1               0                 0                 1
154  154                3               1                 2                 0

              NOC2
0      Afghanistan
1          Albania
2          Algeria
3        Argentina
4          Armenia
..             …
150      Venezuela
151        Vietnam
152  VirginIslands
153         Zambia
154       Zimbabwe
```

```
[155 rows x 6 columns]
0 Afghanistan
MSE: 1.6973353922367096e-07
MSE: 0.0
MSE: 0.0
MSE: 1.6973353922367096e-07
1 Albania
MSE: 1.5524549326073611e-06
MSE: 0.0
MSE: 0.0
MSE: 1.5524549326073611e-06
2 Algeria
MSE: 3.994724107171578
MSE: 0.0715187132484516
MSE: 6.735792893548496e-07
MSE: 0.40100133269567095
3 Argentina
MSE: 2.3470784071832895e-07
MSE: 3.494688058935935e-07
MSE: 4.82183182271001e-09
MSE: 1.0942446948547513e-08
4 Armenia
MSE: 1.7953652786673047e-08
MSE: 0.022775736567688165
MSE: 1.1654483387246728e-06
MSE: 0.000602870343314521
5 Australasia
MSE: 7.366907084360719e-09
MSE: 2.8141045049778768e-09
MSE: 0.0
MSE: 3.5811922316497657e-09
6 Australia
MSE: 46.04812975054665
MSE: 18.760339548826778
MSE: 14.428052094081067
MSE: 2.297645300950535
```

```
7 Austria
MSE: 3.2837787522751114
MSE: 0.929088444383197
MSE: 0.3103930830840831
MSE: 0.12714879985674088
8 Azerbaijan
MSE: 0.9369287917252223
MSE: 5.139809786669503e-07
MSE: 0.01512698294305892
MSE: 3.4371402080068947e-07
9 Bahamas
MSE: 2.9368317200351157e-07
MSE: 4.619677440587111e-08
MSE: 2.775757794742491e-08
MSE: 4.4608173054226145e-07
10 Bahrain
MSE: 1.4410503013095877e-06
MSE: 1.3981571242993596e-06
MSE: 2.537326828644382e-07
MSE: 1.4199488305166597e-06
11 Barbados
MSE: 1.140026007704961e-08
MSE: 0.0
MSE: 0.0
MSE: 1.140026007704961e-08
12 Belarus
MSE: 0.04860342527626926
MSE: 5.0977689625142375e-08
MSE: 0.10681934870467558
MSE: 3.903685922068689e-06
13 Belgium
MSE: 1.1185702533111908e-06
MSE: 0.10785510490313754
MSE: 2.5321249417231684e-06
MSE: 8.881784197001252e-08
14 Bermuda
MSE: 1.1742159244931827e-08
```

```
MSE: 1.6973353922367096e-07
MSE: 0.0
MSE: 2.3612407219856805e-08
15 Botswana
MSE: 1.3981571242993596e-06
MSE: 1.4199488305166597e-06
MSE: 0.06726882339650331
MSE: 9.92802048623378e-08
16 Brazil
MSE: 2.201559254899621e-07
MSE: 0.019516072094120318
MSE: 1.5123171124287182e-06
MSE: 3.873096941505537
17 BritishWestIndies
MSE: 2.3428810891346075e-08
MSE: 0.0
MSE: 0.0
MSE: 2.3428810891346075e-08
18 Bulgaria
MSE: 3.1766921892995015e-07
MSE: 0.011571393059114143
MSE: 0.42826602829359217
MSE: 0.0004790748798768618
19 BurkinaFaso
MSE: 4.778883209155538e-07
MSE: 0.0
MSE: 0.0
MSE: 4.778883209155538e-07
20 Burundi
MSE: 7.42305452661185e-08
MSE: 1.0778264337465494e-08
MSE: 2.1742369327171218e-07
MSE: 0.0
21 CaboVerde
MSE: 1.4199488305166597e-06
MSE: 0.0
MSE: 0.0
```

```
MSE: 1.4199488305166597e-06
22 Cameroon
MSE: 3.263864201130673e-08
MSE: 4.982917154450206e-07
MSE: 3.155964636928306e-08
MSE: 2.0903196290074218e-07
23 Canada
MSE: 1.63796721608378e-06
MSE: 1.4971092241466977e-06
MSE: 1.342618816124741e-08
MSE: 0.008354335444892058
24 Ceylon
MSE: 7.160666008459318e-09
MSE: 0.0
MSE: 7.160666008459318e-09
MSE: 0.0
25 Chile
MSE: 3.476111487543676e-08
MSE: 7.718895744801557e-09
MSE: 1.955996253855119e-09
MSE: 1.9367148421502254e-08
26 China
MSE: 1.6994433920481242
MSE: 3.6888543146196753
MSE: 0.049751780348742614
MSE: 0.22513277226607897
27 ChineseTaipei
MSE: 4.584762791637331e-07
MSE: 2.8574106636369834e-08
MSE: 9.210824938766463e-05
MSE: 1.4479246601695195e-07
28 Colombia
MSE: 4.800773467650288e-08
MSE: 7.667576085625353e-08
MSE: 1.885928213596344e-08
MSE: 1.248465650860453e-07
29 CostaRica
```

```
MSE: 3.2878375577638508e-09

MSE: 1.1996345016534961e-08

MSE: 1.654898907929553e-08

MSE: 2.1325163857000007e-08

30 Croatia

MSE: 2.7412261260906234e-07

MSE: 1.1951328815484885e-09

MSE: 2.786348431982333e-07

MSE: 0.09114096443249764

31 Cuba

MSE: 33.48610947668567

MSE: 1.2728315964550916

MSE: 0.948688959761057

MSE: 2.3940999722990455

32 Cyprus

MSE: 2.1742369327171218e-07

MSE: 0.0

MSE: 2.1742369327171218e-07

MSE: 0.0

33 CzechRepublic

MSE: 0.84680817145113

MSE: 1.0291515195604006e-06

MSE: 0.9990270121516573

MSE: 3.725290298461914e-09

34 Denmark

MSE: 0.03616618936939631

MSE: 9.044993589668593e-07

MSE: 0.1823740395662412

MSE: 1.0471276254975237e-06

35 Djibouti

MSE: 4.99626951493399e-10

MSE: 0.0

MSE: 0.0

MSE: 4.99626951493399e-10

36 Dominica

MSE: 1.4199488305166597e-06

MSE: 1.4199488305166597e-06
```

```
MSE: 0.0
MSE: 0.0
37 DominicanRepublic
MSE: 2.525930540286936e-07
MSE: 1.868126137338777e-06
MSE: 9.782178919332541e-08
MSE: 6.22405934791459e-07
38 Ecuador
MSE: 1.5181867638602853e-06
MSE: 6.74118041388283e-08
MSE: 1.7011171991043739e-06
MSE: 1.5524549326073611e-06
39 Egypt
MSE: 0.23856807980359918
MSE: 3.078030772485363e-07
MSE: 1.7557830744863168e-09
MSE: 4.140958503623779
40 Eritrea
MSE: 1.2923835163312744e-08
MSE: 0.0
MSE: 0.0
MSE: 1.2923835163312744e-08
41 Estonia
MSE: 2.1151436158106662e-10
MSE: 6.444296118957027e-08
MSE: 1.6270200604035775e-08
MSE: 3.412026217120001e-09
42 Ethiopia
MSE: 0.010629131050393426
MSE: 1.8673951274195133e-07
MSE: 0.019763054388022283
MSE: 1.2788976758932105e-06
43 FRYugoslavia
MSE: 5.835327101522125e-07
MSE: 7.42305452661185e-08
MSE: 7.42305452661185e-08
MSE: 1.0830504493242188e-08
```

```
44 Fiji
MSE: 6.195568857947364e-08
MSE: 3.211040390258338e-07
MSE: 1.4199488305166597e-06
MSE: 9.92802048623378e-08
45 Finland
MSE: 1.1690545420606213e-06
MSE: 2.1794434937132136e-08
MSE: 1.4613750559701322e-06
MSE: 6.722712555529142e-07
46 France
MSE: 43.71525864351133
MSE: 7.979276914310503
MSE: 12.340475586766843
MSE: 53.4596161448726
47 Gabon
MSE: 2.3095451991457522e-08
MSE: 0.0
MSE: 2.3095451991457522e-08
MSE: 0.0
48 Georgia
MSE: 0.7632527873668096
MSE: 0.16041672148645603
MSE: 2.521139776945347e-07
MSE: 8.145667749204222e-07
49 Germany
MSE: 6.055925041437149e-07
MSE: 0.06465836433926597
MSE: 0.00010631933400873095
MSE: 1.0608346201479435e-08
50 Ghana
MSE: 2.5669578462839127e-08
MSE: 0.0
MSE: 2.263556034862694e-08
MSE: 2.873554194593453e-08
51 GreatBritain
MSE: 2.852175384759903e-09
```

```
MSE: 1.6575540939811617e-08
MSE: 0.00024034082889556885
MSE: 34.540550915757194
52 Greece
MSE: 3.0976461244392794
MSE: 1.2118888018651433
MSE: 0.9956312315707692
MSE: 5.115907697472721e-07
53 Grenada
MSE: 1.3981571242993596e-06
MSE: 6.5833540192780865e-09
MSE: 1.6884047836144305e-09
MSE: 1.4026714012516095e-06
54 Guatemala
MSE: 1.3981571242993596e-06
MSE: 4.778883209155538e-07
MSE: 6.5833540192780865e-09
MSE: 4.778883209155538e-07
55 Guyana
MSE: 1.6995968167330504e-08
MSE: 0.0
MSE: 0.0
MSE: 1.6995968167330504e-08
56 Haiti
MSE: 5.802590408165997e-09
MSE: 0.0
MSE: 1.1631658389687339e-08
MSE: 8.517478996543298e-09
57 HongKong
MSE: 1.248465650860453e-07
MSE: 1.8416621969663538e-06
MSE: 1.2191963105777634e-07
MSE: 1.2249893188709393e-07
58 Hungary
MSE: 0.002195656594267348
MSE: 0.0002019169187406078
MSE: 1.1612176542187171
```

```
MSE: 0.006406245419384504
59 Iceland
MSE: 2.324669878817076e-08
MSE: 0.0
MSE: 0.002013544599727135
MSE: 1.4244981230149278e-06
60 IndependentOlympicAthletes
MSE: 1.1061871418860392e-09
MSE: 3.211040390258338e-07
MSE: 0.0
MSE: 3.211040390258338e-07
61 IndependentOlympicParticipants
MSE: 3.8314794892357895e-08
MSE: 0.0
MSE: 1.2548357216595072e-08
MSE: 1.0830504493242188e-08
62 India
MSE: 0.06979242902184524
MSE: 0.7265408555982198
MSE: 0.9983279547677739
MSE: 4.9885456974152476e-06
63 Indonesia
MSE: 1.8737522608134896
MSE: 1.6257845345535316e-06
MSE: 1.3687797402614174e-06
MSE: 2.4663728481755243e-07
64 Iran
MSE: 1.642853021621704e-06
MSE: 5.83309542889765e-08
MSE: 7.531583619324067
MSE: 2.180833917009295e-06
65 Iraq
MSE: 2.7549018638868555e-08
MSE: 0.0
MSE: 0.0
MSE: 2.7549018638868555e-08
66 Ireland
```

```
MSE: 1.4142724467092194e-06
MSE: 1.8448991454533825e-06
MSE: 0.46593481471610687
MSE: 0.7807063998482136
67 Israel
MSE: 1.7584991383046145e-06
MSE: 1.3219647598816664e-07
MSE: 1.8261644072481431e-06
MSE: 2.7850900607973017e-07
68 Italy
MSE: 0.00036612991243600845
MSE: 1.4288358099438483
MSE: 0.564170645521699
MSE: 1.7591466999292606
69 IvoryCoast
MSE: 2.6284396881237626e-08
MSE: 2.3095451991457522e-08
MSE: 1.8298635804604773e-08
MSE: 9.313225746154785e-08
70 Jamaica
MSE: 3.1977305135480947
MSE: 0.013250568242767713
MSE: 0.0024697603890331266
MSE: 0.3690294030780592
71 Japan
MSE: 3.2145180739462376e-08
MSE: 5.820766091346741e-11
MSE: 1.2450072972569615e-06
MSE: 0.000157535324433411
72 Jordan
MSE: 3.32009939540967e-08
MSE: 5.387291710150455e-09
MSE: 4.778883209155538e-07
MSE: 1.6884047836144305e-09
73 Kazakhstan
MSE: 3.9617589209228754e-07
MSE: 1.0344371048631729e-07
```

```
MSE: 1.3376143215282355e-07

MSE: 1.7768252291716635e-07

74 Kenya

MSE: 4.193143468000926e-07

MSE: 1.161333784693852e-08

MSE: 1.8744076498933282e-05

MSE: 1.8814792641026088

75 Kosovo

MSE: 4.5204160414868966e-07

MSE: 1.4409998954043146e-07

MSE: 1.4199488305166597e-06

MSE: 1.4199488305166597e-06

76 Kuwait

MSE: 1.0754776269550348e-07

MSE: 0.0

MSE: 0.0

MSE: 1.0754776269550348e-07

77 Kyrgyzstan

MSE: 0.08774251273848677

MSE: 0.0

MSE: 3.089947000489701e-07

MSE: 1.5193620583886513e-06

78 Latvia

MSE: 6.0040861171728466e-09

MSE: 1.0754776269550348e-07

MSE: 5.725222185767824e-11

MSE: 1.3648104868480004e-10

79 Lebanon

MSE: 2.1325163857000007e-08

MSE: 0.0

MSE: 5.551115123125783e-09

MSE: 3.915943125321064e-10

80 Lithuania

MSE: 0.6522478321587641

MSE: 7.01580543894945e-07

MSE: 0.9995318007261176

MSE: 1.0194995070378354e-08
```

```
81 Luxembourg
MSE: 1.6816557035781443e-09
MSE: 6.814904196517091e-09
MSE: 3.3506523167585813e-09
MSE: 0.0
82 Macedonia
MSE: 1.140026007704961e-08
MSE: 0.0
MSE: 0.0
MSE: 1.140026007704961e-08
83 Malaysia
MSE: 8.543997864762787e-08
MSE: 0.0
MSE: 1.4577337912290291e-06
MSE: 1.5757123605908419e-06
84 Mauritius
MSE: 1.074163096010802e-08
MSE: 0.0
MSE: 0.0
MSE: 1.074163096010802e-08
85 Mexico
MSE: 0.9750407944522976
MSE: 0.00123627023016383
MSE: 0.9532222128905801
MSE: 3.5144368818055796
86 Mixedteam
MSE: 9.094947017729282e-13
MSE: 4.330104275140911e-09
MSE: 1.035971308738226e-09
MSE: 4.8466972657479346e-09
87 Moldova
MSE: 0.12468672506435041
MSE: 0.0
MSE: 1.4780625043897544e-06
MSE: 1.81073210114846e-06
88 Mongolia
MSE: 0.5925653473187964
```

```
MSE: 6.8889623910832355e-09
MSE: 0.8167207442879034
MSE: 3.9594106877861037
89 Montenegro
MSE: 2.3095451991457522e-08
MSE: 0.0
MSE: 2.3095451991457522e-08
MSE: 0.0
90 Morocco
MSE: 0.0621849701985866
MSE: 4.6511901530266186e-07
MSE: 6.6136397777894295e-09
MSE: 0.06726647361860216
91 Mozambique
MSE: 7.42305452661185e-08
MSE: 6.5513532170963446e-09
MSE: 0.0
MSE: 1.1996345016534961e-08
92 Namibia
MSE: 2.0497751052062085e-08
MSE: 0.0
MSE: 2.0497751052062085e-08
MSE: 0.0
93 Netherlands
MSE: 4.895264282822609e-08
MSE: 1.682201400399208e-06
MSE: 3.4904610402008984e-07
MSE: 1.101434463635087e-07
94 NetherlandsAntilles
MSE: 4.99626951493399e-10
MSE: 0.0
MSE: 4.99626951493399e-10
MSE: 0.0
95 NewZealand
MSE: 5.392394086811692e-07
MSE: 1.813299604691565e-06
MSE: 1.4324768926599063e-06
```

```
MSE: 1.1054964943468804e-08
96 Niger
MSE: 6.407848474054845e-09
MSE: 0.0
MSE: 9.313225746154785e-08
MSE: 2.1719322495902997e-08
97 Nigeria
MSE: 2.1082144030515337e-08
MSE: 2.5066654497944264e-09
MSE: 5.0816325369851256e-08
MSE: 6.369734961708673e-09
98 NorthKorea
MSE: 5.8215846365783364e-08
MSE: 1.792197035308668e-06
MSE: 3.387004312571662e-07
MSE: 1.0614653211096083
99 NorthMacedonia
MSE: 4.778883209155538e-07
MSE: 0.0
MSE: 4.778883209155538e-07
MSE: 0.0
100 Norway
MSE: 0.09249974467115862
MSE: 0.06336245393163153
MSE: 0.8465350429990082
MSE: 2.7309997733482305e-05
101 Pakistan
MSE: 2.063451631784119e-08
MSE: 1.4284850990975428e-07
MSE: 9.562590587676924e-09
MSE: 2.0079145783913323e-09
102 Panama
MSE: 2.0259506072761724e-08
MSE: 1.46794109540162e-09
MSE: 4.778883209155538e-07
MSE: 8.960664046499676e-10
103 Paraguay
```

```
MSE: 1.2923835163312744e-08
MSE: 0.0
MSE: 1.2923835163312744e-08
MSE: 0.0
104 Peru
MSE: 1.0424972884948147e-08
MSE: 5.429920696925306e-09
MSE: 2.5110489548319094e-09
MSE: 2.537326828644382e-07
105 Philippines
MSE: 4.760770480061183e-07
MSE: 1.3981571242993596e-06
MSE: 0.22906018066709866
MSE: 1.762611660183211e-06
106 Poland
MSE: 1.8835820108652115
MSE: 1.4319062415779626e-06
MSE: 0.011657366299914429
MSE: 0.06389915492400178
107 Portugal
MSE: 0.17264335821215582
MSE: 7.583341954386924e-07
MSE: 0.06631665280133348
MSE: 0.4225406888982528
108 PuertoRico
MSE: 1.7581829894197654e-06
MSE: 3.211040390258338e-07
MSE: 2.2689356340973604e-07
MSE: 1.4889521935401717e-06
109 Qatar
MSE: 1.4262332115322351e-07
MSE: 1.0822434133541845e-07
MSE: 9.578828589701526e-08
MSE: 5.445324191555301e-08
110 ROC
MSE: 1.6350531950592995e-07
MSE: 1.752787284203805e-07
```

```
MSE: 6.007121555740014e-08
MSE: 1.3271710486151278e-07
111 RefugeeOlympicTeam
MSE: 1.4199488305166597e-06
MSE: 0.0
MSE: 0.0
MSE: 1.4199488305166597e-06
112 Romania
MSE: 4.682085129465122
MSE: 0.0023860022158714855
MSE: 0.03766681315846654
MSE: 0.4349356939110862
113 Russia
MSE: 0.010707588458899409
MSE: 0.027287651206279406
MSE: 8.185452315956354e-08
MSE: 0.07871044465355226
114 SaintLucia
MSE: 1.5524549326073611e-06
MSE: 1.4199488305166597e-06
MSE: 1.4199488305166597e-06
MSE: 0.0
115 Samoa
MSE: 1.074163096010802e-08
MSE: 0.0
MSE: 1.074163096010802e-08
MSE: 0.0
116 SanMarino
MSE: 7.834591997379903e-08
MSE: 0.0
MSE: 4.778883209155538e-07
MSE: 1.0707914555041498e-07
117 SaudiArabia
MSE: 7.887379638304992e-09
MSE: 0.0
MSE: 2.623921915301253e-07
MSE: 4.986535256637552e-07
```

```
118 Senegal
MSE: 4.99626951493399e-10
MSE: 0.0
MSE: 4.99626951493399e-10
MSE: 0.0
119 Serbia
MSE: 1.477101250202395e-07
MSE: 2.1570394892478362e-07
MSE: 2.8090477144360193e-07
MSE: 1.4190391084412113e-07
120 SerbiaandMontenegro
MSE: 7.718895744801557e-09
MSE: 0.0
MSE: 7.718895744801557e-09
MSE: 0.0
121 Singapore
MSE: 1.1510792319313623e-10
MSE: 1.6884047836144305e-09
MSE: 1.398155706801059e-07
MSE: 5.7985971579910256e-08
122 Slovakia
MSE: 4.5220191680073185e-07
MSE: 1.1319621282318755e-06
MSE: 1.5416651523103239e-06
MSE: 2.8032619781015455e-08
123 Slovenia
MSE: 0.2425715807476081
MSE: 2.461115400365088e-08
MSE: 5.994852614321644e-09
MSE: 0.3530150804849974
124 SouthAfrica
MSE: 0.5164398081578838
MSE: 0.0506667008408499
MSE: 0.08793004583191077
MSE: 0.0007638633409356999
125 SouthKorea
MSE: 43.26979633342489
```

```
MSE: 8.856364980260878
MSE: 1.2459555987063595
MSE: 1.0008031549741645
126 Spain
MSE: 35.331490358920746
MSE: 0.10381211041021743
MSE: 0.17627111272395268
MSE: 2.291215537134349
127 SriLanka
MSE: 1.140026007704961e-08
MSE: 0.0
MSE: 1.140026007704961e-08
MSE: 0.0
128 Sudan
MSE: 1.074163096010802e-08
MSE: 0.0
MSE: 1.074163096010802e-08
MSE: 0.0
129 Suriname
MSE: 4.1797321159720013e-08
MSE: 1.654898907929553e-08
MSE: 0.0
MSE: 1.1631658389687339e-08
130 Sweden
MSE: 1.9083821927861209
MSE: 0.06598845990106383
MSE: 0.06784283701654203
MSE: 0.7962807892091064
131 Switzerland
MSE: 0.9667695876871676
MSE: 3.8678848104464123
MSE: 0.011723307024169571
MSE: 6.526305188073422
132 Syria
MSE: 3.324445074781579e-08
MSE: 1.6080186172781866e-08
MSE: 1.65282874911838e-08
```

```
MSE: 9.313225746154785e-08
133 Taiwan
MSE: 0.0
MSE: 0.0
MSE: 0.0
MSE: 0.0
134 Tajikistan
MSE: 1.6581680597482773e-06
MSE: 5.387291710150455e-09
MSE: 6.3767993767271316e-09
MSE: 1.377653063627804e-06
135 Tanzania
MSE: 1.6760225207690382e-08
MSE: 0.0
MSE: 1.6760225207690382e-08
MSE: 0.0
136 Thailand
MSE: 0.9632442612428349
MSE: 0.00017719934754723
MSE: 1.7095235307351686e-06
MSE: 0.21871474907038646
137 Togo
MSE: 1.074163096010802e-08
MSE: 0.0
MSE: 0.0
MSE: 1.074163096010802e-08
138 Tonga
MSE: 1.0778264337465494e-08
MSE: 0.0
MSE: 1.0778264337465494e-08
MSE: 0.0
139 TrinidadandTobago
MSE: 0.0032727213500862717
MSE: 1.8668228882745552e-06
MSE: 0.0019722895922882344
MSE: 0.00010224747132170364
140 Tunisia
```

```
MSE: 4.1224262758987607e-07
MSE: 4.604316927725449e-08
MSE: 3.049969166113442e-08
MSE: 1.6554061090801042e-06
141 Turkey
MSE: 4.860666316170864
MSE: 1.1744494875247256
MSE: 4.015513412259143
MSE: 4.807044206245337e-06
142 Turkmenistan
MSE: 4.778883209155538e-07
MSE: 0.0
MSE: 4.778883209155538e-07
MSE: 0.0
143 Uganda
MSE: 1.3815684951623552e-07
MSE: 5.908192690640135e-08
MSE: 4.330104275140911e-09
MSE: 4.754275237188478e-07
144 Ukraine
MSE: 3.40419490225122
MSE: 0.7299633708512943
MSE: 0.137955934971842
MSE: 0.03334851470049216
145 UnifiedTeam
MSE: 3.230670699849725e-07
MSE: 2.516353561077267e-07
MSE: 5.047922968515195e-07
MSE: 1.6350531950592995e-07
146 UnitedArabEmirates
MSE: 9.313225746154785e-08
MSE: 8.74015187083042e-09
MSE: 0.0
MSE: 2.1742369327171218e-07
147 UnitedStates
MSE: 5.0188551566097885
MSE: 14.137472632122808
```

```
MSE: 1.0510191714856774
MSE: 31.46059445689025
148 Uruguay
MSE: 1.2185466857772553e-08
MSE: 8.925877397551354e-09
MSE: 2.0251356147582555e-09
MSE: 1.2979553837339814e-09
149 Uzbekistan
MSE: 1.7145148376584984e-06
MSE: 0.45250979676438874
MSE: 0.9990391656192656
MSE: 1.9813847984551103e-07
150 Venezuela
MSE: 1.602439283487911e-07
MSE: 5.948828629698255e-08
MSE: 1.9244907889515162e-07
MSE: 3.937924236597676e-08
151 Vietnam
MSE: 2.516353561077267e-07
MSE: 3.211040390258338e-07
MSE: 1.0754776269550348e-07
MSE: 6.5833540192780865e-09
152 VirginIslands
MSE: 4.99626951493399e-10
MSE: 0.0
MSE: 4.99626951493399e-10
MSE: 0.0
153 Zambia
MSE: 3.324445074781579e-08
MSE: 0.0
MSE: 7.859672177473758e-10
MSE: 2.9385830657702172e-08
154 Zimbabwe
MSE: 7.781864042044617e-11
MSE: 5.2332339350869006e-08
MSE: 1.2535252835732535e-09
MSE: 1.140026007704961e-08
```

```
               NOC  Total_Predicted  Total_Lower  Total_Upper  Gold_Predicted  \
0      Afghanistan                1            1            1               0
1          Albania                2            2            2               0
2          Algeria                5            4            6               2
3        Argentina                3            3            3               1
4          Armenia                4            4            4               0
..             …               …           …           …             …
150      Venezuela                3            3            3               1
151        Vietnam                1            1            1               0
152  VirginIslands                0            0            0               0
153         Zambia                1            1            1               0
154       Zimbabwe                3            3            3               1

     Gold_Lower  Gold_Upper  Silver_Predicted  Silver_Lower  Silver_Upper  \
0             0           0                 0             0             0
1             0           0                 0             0             0
2             2           2                 0             0             0
3             1           1                 1             1             1
4             0           0                 3             3             3
..          …         …               …            …            …
150           1           1                 2             2             2
151           0           0                 1             1             1
152           0           0                 0             0             0
153           0           0                 0             0             0
154           1           1                 2             2             2

     Bronze_Predicted  Bronze_Lower  Bronze_Upper
0                   1             1             1
1                   2             2             2
2                   2             2             2
3                   1             1             1
4                   1             1             1
..                …           …           …
150                 0             0             0
151                 0             0             0
152                 0             0             0
153                 1             1             1
```

```
154                  0              0              0

[155 rows x 13 columns]
```

- 结论：XGBoost 已经训练好的模型的值趋近于不变

3.2 贝叶斯方法

3.2.1 先验预测

[73]:

```python
import pandas as pd
import numpy as np
from scipy.stats import norm
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.preprocessing import LabelEncoder

# 加载数据
data = pd.read_csv('Generated\\summerOly_medal_counts_processed.csv')

# 数据预处理
# 将国家代码转换为数值标签
label_encoder = LabelEncoder()
data['NOC'] = label_encoder.fit_transform(data['NOC'])

# 处理缺失值
data = data.fillna(0)

# 创建特征：前一届奥运会的奖牌总数、金牌数、银牌数、铜牌数
data['Prev_Total'] = data.groupby('NOC')['Total'].shift(1)
data['Prev_Gold'] = data.groupby('NOC')['Gold'].shift(1)
data['Prev_Silver'] = data.groupby('NOC')['Silver'].shift(1)
data['Prev_Bronze'] = data.groupby('NOC')['Bronze'].shift(1)

# 填充缺失值
data['Prev_Total'] = data['Prev_Total'].fillna(0)
data['Prev_Gold'] = data['Prev_Gold'].fillna(0)
```

```
data['Prev_Silver'] = data['Prev_Silver'].fillna(0)
data['Prev_Bronze'] = data['Prev_Bronze'].fillna(0)

# 选择特征和目标变量
features = data[['Year', 'NOC', 'Prev_Total', 'Prev_Gold', 'Prev_Silver',␣
 ↪'Prev_Bronze']]
target_total = data['Total']
target_gold = data['Gold']
target_silver = data['Silver']
target_bronze = data['Bronze']

# 划分训练集和测试集
X_train_total, X_test_total, y_train_total, y_test_total =␣
 ↪train_test_split(features, target_total, test_size=0.2, random_state=42)
X_train_gold, X_test_gold, y_train_gold, y_test_gold =␣
 ↪train_test_split(features, target_gold, test_size=0.2, random_state=42)
X_train_silver, X_test_silver, y_train_silver, y_test_silver =␣
 ↪train_test_split(features, target_silver, test_size=0.2, random_state=42)
X_train_bronze, X_test_bronze, y_train_bronze, y_test_bronze =␣
 ↪train_test_split(features, target_bronze, test_size=0.2, random_state=42)

# 定义 XGBoost 模型
model_total = XGBRegressor(objective='reg:squarederror', random_state=42)
model_gold = XGBRegressor(objective='reg:squarederror', random_state=42)
model_silver = XGBRegressor(objective='reg:squarederror', random_state=42)
model_bronze = XGBRegressor(objective='reg:squarederror', random_state=42)

# 超参数优化
param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.7, 0.8, 0.9]
}

# 使用 GridSearchCV 进行超参数优化
```

```python
grid_search_total = GridSearchCV(estimator=model_total, param_grid=param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_gold = GridSearchCV(estimator=model_gold, param_grid=param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_silver = GridSearchCV(estimator=model_silver, param_grid=param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_bronze = GridSearchCV(estimator=model_bronze, param_grid=param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1)

# 训练模型
grid_search_total.fit(X_train_total, y_train_total)
grid_search_gold.fit(X_train_gold, y_train_gold)
grid_search_silver.fit(X_train_silver, y_train_silver)
grid_search_bronze.fit(X_train_bronze, y_train_bronze)

# 获取最佳模型
best_model_total = grid_search_total.best_estimator_
best_model_gold = grid_search_gold.best_estimator_
best_model_silver = grid_search_silver.best_estimator_
best_model_bronze = grid_search_bronze.best_estimator_

# 评估模型
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    print(f'MSE: {mse}, R2: {r2}, MAE: {mae}')
    return y_pred

print("Total Medals Model Evaluation:")
evaluate_model(best_model_total, X_test_total, y_test_total)

print("Gold Medals Model Evaluation:")
evaluate_model(best_model_gold, X_test_gold, y_test_gold)
```

```
print("Silver Medals Model Evaluation:")
evaluate_model(best_model_silver, X_test_silver, y_test_silver)

print("Bronze Medals Model Evaluation:")
evaluate_model(best_model_bronze, X_test_bronze, y_test_bronze)

# 贝叶斯更新
def bayesian_update(prior_mean, prior_std, new_data):
    if np.isnan(prior_mean) or np.isnan(prior_std) or np.isnan(new_data).any():
        return np.nan, np.nan

    n = len(new_data)
    new_mean = np.mean(new_data)
    new_std = np.std(new_data)

    # 避免除以零
    if prior_std == 0:
        prior_std = 1e-6
    if new_std == 0:
        new_std = 1e-6

    # 更新后验分布的参数
    posterior_mean = (prior_mean / prior_std**2 + new_mean * n / new_std**2) / (1 / prior_std**2 + n / new_std**2)
    posterior_std = np.sqrt(1 / (1 / prior_std**2 + n / new_std**2))

    return posterior_mean, posterior_std

# 使用历史数据作为先验分布
prior_mean_total = np.mean(data['Total'])
prior_std_total = np.std(data['Total'])
prior_mean_gold = np.mean(data['Gold'])
prior_std_gold = np.std(data['Gold'])
prior_mean_silver = np.mean(data['Silver'])
prior_std_silver = np.std(data['Silver'])
prior_mean_bronze = np.mean(data['Bronze'])
```

```
prior_std_bronze = np.std(data['Bronze'])

# 为每个国家单独训练模型
country_models = {}
for country in data['NOC'].unique():
    country_data = data[data['NOC'] == country]
    if len(country_data) > 1:  # 确保每个国家至少有两条记录
        country_features = country_data[['Year', 'NOC', 'Prev_Total',␣
 ↪'Prev_Gold', 'Prev_Silver', 'Prev_Bronze']]
        country_target_total = country_data['Total']
        country_target_gold = country_data['Gold']
        country_target_silver = country_data['Silver']
        country_target_bronze = country_data['Bronze']

        country_model_total = XGBRegressor(objective='reg:squarederror',␣
 ↪random_state=42)
        country_model_gold = XGBRegressor(objective='reg:squarederror',␣
 ↪random_state=42)
        country_model_silver = XGBRegressor(objective='reg:squarederror',␣
 ↪random_state=42)
        country_model_bronze = XGBRegressor(objective='reg:squarederror',␣
 ↪random_state=42)

        country_model_total.fit(country_features, country_target_total)
        country_model_gold.fit(country_features, country_target_gold)
        country_model_silver.fit(country_features, country_target_silver)
        country_model_bronze.fit(country_features, country_target_bronze)

        country_models[country] = {
            'total': country_model_total,
            'gold': country_model_gold,
            'silver': country_model_silver,
            'bronze': country_model_bronze
        }

# 预测 2028 年奥运会的奖牌数
```

```
next_year = 2028
predictions = []

for country in data['NOC'].unique():
    country_data = data[data['NOC'] == country]
    if len(country_data) > 1:
        prev_total = country_data['Total'].iloc[-1]
        prev_gold = country_data['Gold'].iloc[-1]
        prev_silver = country_data['Silver'].iloc[-1]
        prev_bronze = country_data['Bronze'].iloc[-1]

        next_data = pd.DataFrame({
            'Year': [next_year],
            'NOC': [country],
            'Prev_Total': [prev_total],
            'Prev_Gold': [prev_gold],
            'Prev_Silver': [prev_silver],
            'Prev_Bronze': [prev_bronze]
        })

        # 使用单独模型预测
        total_pred_private = country_models[country]['total'].predict(next_data)
        gold_pred_private = country_models[country]['gold'].predict(next_data)
        silver_pred_private = country_models[country]['silver'].
 ↪predict(next_data)
        bronze_pred_private = country_models[country]['bronze'].
 ↪predict(next_data)

        # 使用整体模型预测
        total_pred_global = best_model_total.predict(next_data)
        gold_pred_global = best_model_gold.predict(next_data)
        silver_pred_global = best_model_silver.predict(next_data)
        bronze_pred_global = best_model_bronze.predict(next_data)

        # 根据数据量分配权重
        data_count = len(country_data)
        weight = min(data_count / 10, 1)  # 数据量越多，权重越高，但不超过 1

        # 贝叶斯更新
        total_posterior_mean_private, total_posterior_std_private =␣
↪bayesian_update(prior_mean_total, prior_std_total, [total_pred_private])
        gold_posterior_mean_private, gold_posterior_std_private =␣
↪bayesian_update(prior_mean_gold, prior_std_gold, [gold_pred_private])
        silver_posterior_mean_private, silver_posterior_std_private =␣
↪bayesian_update(prior_mean_silver, prior_std_silver, [silver_pred_private])
        bronze_posterior_mean_private, bronze_posterior_std_private =␣
↪bayesian_update(prior_mean_bronze, prior_std_bronze, [bronze_pred_private])

        total_posterior_mean_global, total_posterior_std_global =␣
↪bayesian_update(prior_mean_total, prior_std_total, [total_pred_global])
        gold_posterior_mean_global, gold_posterior_std_global =␣
↪bayesian_update(prior_mean_gold, prior_std_gold, [gold_pred_global])
        silver_posterior_mean_global, silver_posterior_std_global =␣
↪bayesian_update(prior_mean_silver, prior_std_silver, [silver_pred_global])
        bronze_posterior_mean_global, bronze_posterior_std_global =␣
↪bayesian_update(prior_mean_bronze, prior_std_bronze, [bronze_pred_global])

        # 合成预测结果
        total_posterior_mean_combined = weight * total_posterior_mean_private +␣
↪(1 - weight) * total_posterior_mean_global
        gold_posterior_mean_combined = weight * gold_posterior_mean_private +␣
↪(1 - weight) * gold_posterior_mean_global
        silver_posterior_mean_combined = weight * silver_posterior_mean_private␣
↪+ (1 - weight) * silver_posterior_mean_global
        bronze_posterior_mean_combined = weight * bronze_posterior_mean_private␣
↪+ (1 - weight) * bronze_posterior_mean_global

        total_posterior_std_combined = weight * total_posterior_std_private +␣
↪(1 - weight) * total_posterior_std_global
        gold_posterior_std_combined = weight * gold_posterior_std_private + (1␣
↪- weight) * gold_posterior_std_global
```

```
        silver_posterior_std_combined = weight * silver_posterior_std_private + (1 - weight) * silver_posterior_std_global
        bronze_posterior_std_combined = weight * bronze_posterior_std_private + (1 - weight) * bronze_posterior_std_global

        # 计算 95% 置信区间
        total_lower, total_upper = norm.interval(0.95, loc=total_posterior_mean_combined, scale=total_posterior_std_combined)
        gold_lower, gold_upper = norm.interval(0.95, loc=gold_posterior_mean_combined, scale=gold_posterior_std_combined)
        silver_lower, silver_upper = norm.interval(0.95, loc=silver_posterior_mean_combined, scale=silver_posterior_std_combined)
        bronze_lower, bronze_upper = norm.interval(0.95, loc=bronze_posterior_mean_combined, scale=bronze_posterior_std_combined)

        predictions.append({
            'NOC': country,
            'Total_Predicted': round(total_posterior_mean_combined),
            'Total_Lower': round(total_lower),
            'Total_Upper': round(total_upper),
            'Gold_Predicted': round(gold_posterior_mean_combined),
            'Gold_Lower': round(gold_lower),
            'Gold_Upper': round(gold_upper),
            'Silver_Predicted': round(silver_posterior_mean_combined),
            'Silver_Lower': round(silver_lower),
            'Silver_Upper': round(silver_upper),
            'Bronze_Predicted': round(bronze_posterior_mean_combined),
            'Bronze_Lower': round(bronze_lower),
            'Bronze_Upper': round(bronze_upper)
        })

# 将预测结果转换为 DataFrame
predictions_df = pd.DataFrame(predictions)

# 将 NOC 标签转换回国家代码
```

```
predictions_df['NOC'] = label_encoder.inverse_transform(predictions_df['NOC'].
  ↪astype(int))

# 输出预测结果
print(predictions_df)

# 保存预测结果到 CSV 文件
predictions_df.to_csv('Result\\2028_olympics_medal_predictions_3.csv',␣
  ↪index=False)
```

```
Total Medals Model Evaluation:
MSE: 14.183948320207696, R2: 0.8897282113283617, MAE: 1.2766138611301299
Gold Medals Model Evaluation:
MSE: 2.9414593946059715, R2: 0.82618743758341, MAE: 0.553917856282124
Silver Medals Model Evaluation:
MSE: 1.8287626291987034, R2: 0.867498342532686, MAE: 0.5179618502816846
Bronze Medals Model Evaluation:
MSE: 2.185780634244944, R2: 0.8524910127837848, MAE: 0.5823690356586569
               NOC  Total_Predicted  Total_Lower  Total_Upper  Gold_Predicted  \
0      Afghanistan                1            1            1               0
1          Albania                2            2            2               0
2          Algeria                5            5            5               2
3        Argentina                3            3            3               1
4          Armenia                4            4            4               0
..             …              …          …          …             …
150      Venezuela                3            3            3               1
151        Vietnam                1            1            1               0
152  VirginIslands                0            0            0               0
153         Zambia                1            1            1               0
154       Zimbabwe                3            3            3               1

     Gold_Lower  Gold_Upper  Silver_Predicted  Silver_Lower  Silver_Upper  \
0             0           0                 0             0             0
1             0           0                 0             0             0
2             2           2                 0             0             0
3             1           1                 1             1             1
4             0           0                 3             3             3
```

```
..          …         …               …           …           …
150           1           1                 2             2             2
151           0           0                 1             1             1
152           0           0                 0             0             0
153           0           0                 0             0             0
154           1           1                 2             2             2

     Bronze_Predicted  Bronze_Lower  Bronze_Upper
0                   1             1             1
1                   2             2             2
2                   2             2             2
3                   1             1             1
4                   1             1             1
..                …           …           …
150                 0             0             0
151                 0             0             0
152                 0             0             0
153                 1             1             1
154                 0             0             0

[155 rows x 13 columns]
```

3.2.2 区间合成

[74]:

```python
import pandas as pd

# 读取 CSV 文件
file1 = 'Result\\2028_olympics_medal_predictions_3.csv'
file2 = 'Result\\2028_olympics_medal_predictions_intervals.csv'

# 读取数据
df1 = pd.read_csv(file1)
df2 = pd.read_csv(file2)

# 合并两个数据框，基于 NOC 列
merged_df = pd.merge(df1, df2, on='NOC', suffixes=('_file1', '_file2'))
```

```python
# 计算均值、最大值和最小值
merged_df['Total_Predicted'] = round((merged_df['Total_Predicted_file1'] + merged_df['Total_Predicted_file2']*2) / 3).astype(int)
merged_df['Total_Lower'] = merged_df[['Total_Lower_file1', 'Total_Lower_file2']].min(axis=1)
merged_df['Total_Upper'] = merged_df[['Total_Upper_file1', 'Total_Upper_file2']].max(axis=1)

merged_df['Gold_Predicted'] = round((merged_df['Gold_Predicted_file1'] + merged_df['Gold_Predicted_file2']*2) / 3).astype(int)
merged_df['Gold_Lower'] = merged_df[['Gold_Lower_file1', 'Gold_Lower_file2']].min(axis=1)
merged_df['Gold_Upper'] = merged_df[['Gold_Upper_file1', 'Gold_Upper_file2']].max(axis=1)

merged_df['Silver_Predicted'] = round((merged_df['Silver_Predicted_file1'] + merged_df['Silver_Predicted_file2']*2) / 3).astype(int)
merged_df['Silver_Lower'] = merged_df[['Silver_Lower_file1', 'Silver_Lower_file2']].min(axis=1)
merged_df['Silver_Upper'] = merged_df[['Silver_Upper_file1', 'Silver_Upper_file2']].max(axis=1)

merged_df['Bronze_Predicted'] = round((merged_df['Bronze_Predicted_file1'] + merged_df['Bronze_Predicted_file2']*2) / 3).astype(int)
merged_df['Bronze_Lower'] = merged_df[['Bronze_Lower_file1', 'Bronze_Lower_file2']].min(axis=1)
merged_df['Bronze_Upper'] = merged_df[['Bronze_Upper_file1', 'Bronze_Upper_file2']].max(axis=1)

# 选择需要的列
final_df = merged_df[['NOC', 'Total_Predicted', 'Total_Lower', 'Total_Upper',
                      'Gold_Predicted', 'Gold_Lower', 'Gold_Upper',
                      'Silver_Predicted', 'Silver_Lower', 'Silver_Upper',
                      'Bronze_Predicted', 'Bronze_Lower', 'Bronze_Upper']]
```

```python
# 保存结果到新的 CSV 文件
final_df.to_csv('Result\\merged_olympics_medal_predictions.csv', index=False)

# 显示结果
final_df.head()
```

```
[74]:           NOC  Total_Predicted  Total_Lower  Total_Upper  Gold_Predicted  \
      0  Afghanistan                1            1            1               0
      1      Albania                2            2            2               0
      2      Algeria                5            4            6               2
      3    Argentina                3            3            3               1
      4      Armenia                4            4            4               0

         Gold_Lower  Gold_Upper  Silver_Predicted  Silver_Lower  Silver_Upper  \
      0           0           0                 0             0             0
      1           0           0                 0             0             0
      2           2           2                 0             0             0
      3           1           1                 1             1             1
      4           0           0                 3             3             3

         Bronze_Predicted  Bronze_Lower  Bronze_Upper
      0                 1             1             1
      1                 2             2             2
      2                 2             2             2
      3                 1             1             1
      4                 1             1             1
```

4 结果分析

4.1 第一问

- 构建一个模型，用于预测每个国家的奖牌数（至少包括金牌数和奖牌总数）。请包含对模型预测的不确定度/精确度的估计以及模型性能的衡量指标。根据您的模型，您对 2028 年美国洛杉矶夏季奥运会奖牌榜的预测结果是什么？请给出所有结果的预测区间。您认为哪些国家最有可能取得进步？哪些国家的表现会不如 2024 年？

4.1.1 国家相比 2024 年进步或退步

[75]:

```
# 读取合并后的 CSV 文件
merged_file = 'Result\\merged_olympics_medal_predictions.csv'

# 读取数据
merged_df = pd.read_csv(merged_file)

# 提取预测值列
predictions_df = merged_df[['NOC', 'Total_Predicted', 'Gold_Predicted',␣
  ↪'Silver_Predicted', 'Bronze_Predicted']].sort_values(by='Total_Predicted',␣
  ↪ascending=False).reset_index(drop=True)

# 显示结果
predictions_df.head()

predictions_df.to_csv('Generated\\2028_Prediction_data.csv')
```

[76]:

```
# 读取合并后的 CSV 文件
medal_file = '2025_Problem_C_Data\\summerOly_medal_counts.csv'

# 读取数据
medal_df = pd.read_csv(medal_file)
medal_df = medal_df[medal_df['Year'] == 2024]

# 提取预测值列
predictions_df = medal_df[['NOC', 'Total', 'Gold', 'Silver', 'Bronze']].
  ↪reset_index(drop=True)
predictions_df['NOC'] = predictions_df['NOC'].str.replace(r'[^a-zA-Z]', '',␣
  ↪regex=True)

# 显示结果
predictions_df.head()

predictions_df.to_csv('Generated\\2024_Observation_data.csv')
```

[77]:

```
import pandas as pd

# 2024 年奥运会奖牌榜
data_2024 = pd.read_csv('Generated\\2024_Observation_data.csv')

# 2028 年奥运会奖牌榜预测结果
data_2028 = pd.read_csv('Generated\\2028_Prediction_data.csv')

# 创建 DataFrame
df_2024 = pd.DataFrame(data_2024)
df_2028 = pd.DataFrame(data_2028)

# 合并两个 DataFrame
merged_df = pd.merge(df_2024, df_2028, on='NOC', how='outer')

# 计算奖牌总数和金牌数的变化
merged_df['Total_Change'] = merged_df['Total_Predicted'] - merged_df['Total']
merged_df['Gold_Change'] = merged_df['Gold_Predicted'] - merged_df['Gold']

# 判断进步或退步
#merged_df['Total_Progress'] = merged_df.apply(lambda row: 'Front' if␣
 ↪row['Total_Change'] / row['Total'] > 0.15 else 'Back' if row['Total_Change']␣
 ↪/ row['Total'] < -0.15 else 'Stable', axis=1)
#merged_df['Gold_Progress'] = merged_df.apply(lambda row: 'Front' if␣
 ↪row['Gold_Change'] / row['Gold'] > 0.15 else 'Back' if row['Gold_Change'] /␣
 ↪row['Gold'] < -0.15 else 'Stable', axis=1)

# 判断进步或退步 (0 检验)
merged_df['Total_Progress'] = merged_df.apply(lambda row: 'Front' if␣
 ↪row['Total'] != 0 and row['Total_Change'] / row['Total'] > 0.15 else 'Back'␣
 ↪if row['Total'] != 0 and row['Total_Change'] / row['Total'] < -0.15 else␣
 ↪'Stable' if row['Total'] != 0 else 'Front' if row['Total_Change'] > 0 else␣
 ↪'Stable' if row['Total_Change'] == 0 else 'NaN', axis=1)
```

```
merged_df['Gold_Progress'] = merged_df.apply(lambda row: 'Front' if row['Gold']␣
  ↪!= 0 and row['Gold_Change'] / row['Gold'] > 0.15 else 'Back' if row['Gold'] !
  ↪= 0 and row['Gold_Change'] / row['Gold'] < -0.15 else 'Stable' if␣
  ↪row['Gold'] != 0 else 'Front' if row['Gold_Change'] > 0 else 'Stable' if␣
  ↪row['Gold_Change'] == 0  else 'NaN', axis=1)

# 生成新的 DataFrame
result_df = merged_df[['NOC', 'Total_Progress', 'Gold_Progress']]

# 重命名列
result_df.columns = ['NOC', 'Total', 'Gold']

print(result_df)

result_df.to_csv('Result\\2028_Olympics_country_progress.csv')
```

```
              NOC   Total    Gold
0     Afghanistan  Stable  Stable
1         Albania  Stable  Stable
2         Algeria   Front  Stable
3       Argentina  Stable  Stable
4         Armenia  Stable  Stable
..            …     …     …
150     Venezuela  Stable  Stable
151       Vietnam  Stable  Stable
152  VirginIslands  Stable  Stable
153        Zambia  Stable  Stable
154      Zimbabwe  Stable  Stable

[155 rows x 3 columns]
```

4.2 第二问

- 您的模型应涵盖尚未获得奖牌的国家；您预计在下一届奥运会中会有多少个国家获得其首枚奖牌？对于这个估计，您认为可能性有多大？

4.2.1 预处理

[78]:

```python
import pandas as pd
import re

# 加载数据
file_path = '2025_Problem_C_Data\\summerOly_medal_counts.csv'
data = pd.read_csv(file_path)

# 清洗国家名 NOC，只保留英文字母
data['NOC'] = data['NOC'].apply(lambda x: ''.join(re.findall(r'[A-Za-z]', x)))

# 初始化字典来存储每个国家的第一枚奖牌时间和第一枚金牌时间
first_medal_time = {}
first_gold_time = {}

# 遍历数据
for index, row in data.iterrows():
    year = row['Year']
    noc = row['NOC']
    gold = row['Gold']
    total = row['Total']

    # 如果国家尚未记录第一枚奖牌时间
    if noc not in first_medal_time and total > 0:
        first_medal_time[noc] = year

    # 如果国家尚未记录第一枚金牌时间
    if noc not in first_gold_time and gold > 0:
        first_gold_time[noc] = year

# 将结果转换为 DataFrame
result = pd.DataFrame({
    'NOC': list(first_medal_time.keys()),
    'First Medal Time': list(first_medal_time.values()),
    'First Gold Time': [first_gold_time.get(noc, None) for noc in␣
 ↪first_medal_time.keys()]
})

# 保存结果到 CSV 文件
result.to_csv('Generated\\first_medal_and_gold_times.csv', index=False)

print("结果已保存到 first_medal_and_gold_times.csv 文件中。")
```

结果已保存到 first_medal_and_gold_times.csv 文件中。

[79]:

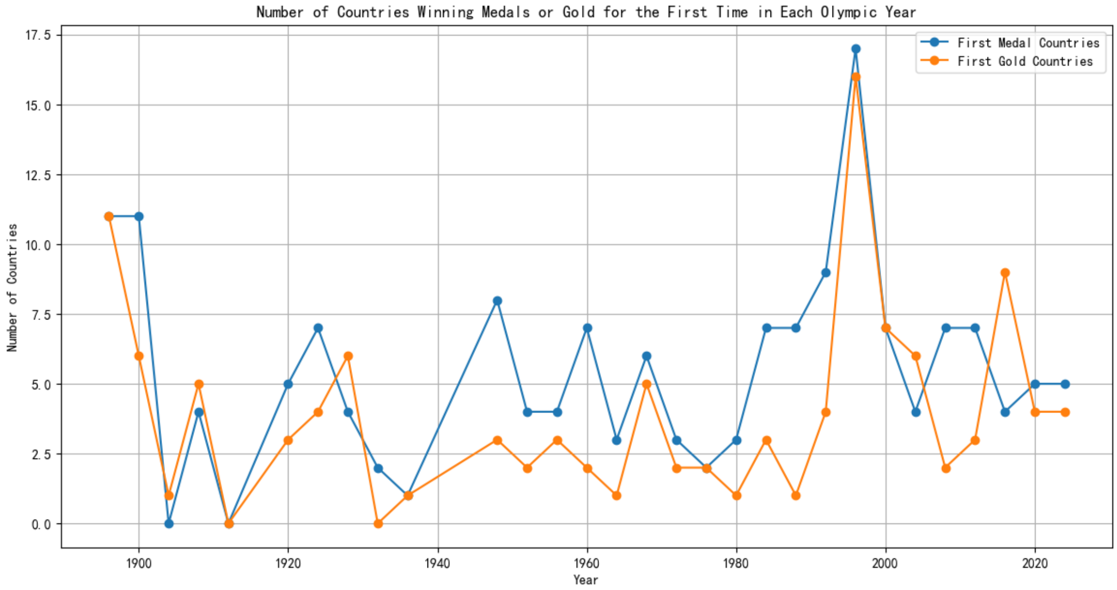
```
import pandas as pd

# 加载数据
file_path = 'Generated\\first_medal_and_gold_times.csv'
data = pd.read_csv(file_path)

# 定义实际的奥运会年份
olympic_years = [1896, 1900, 1904, 1908, 1912, 1920, 1924, 1928, 1932, 1936,␣
 ↪1948, 1952, 1956, 1960, 1964, 1968, 1972, 1976, 1980, 1984, 1988, 1992,␣
 ↪1996, 2000, 2004, 2008, 2012, 2016, 2020, 2024]

# 初始化字典来存储每届奥运会首次获得奖牌和金牌的国家数量
first_medal_counts = {year: 0 for year in olympic_years}
first_gold_counts = {year: 0 for year in olympic_years}

# 遍历数据
for index, row in data.iterrows():
    first_medal_time = row['First Medal Time']
    first_gold_time = row['First Gold Time']

    if first_medal_time in first_medal_counts:
        first_medal_counts[first_medal_time] += 1

    if first_gold_time in first_gold_counts and not pd.isna(first_gold_time):
        first_gold_counts[first_gold_time] += 1

# 将结果转换为 DataFrame
result = pd.DataFrame({
    'Year': olympic_years,
    'First Medal Countries': [first_medal_counts[year] for year in␣
  ↪olympic_years],
    'First Gold Countries': [first_gold_counts[year] for year in olympic_years]
})

# 保存结果到 CSV 文件
result.to_csv('Generated\\first_medal_and_gold_countries.csv', index=False)

print("结果已保存到 first_medal_and_gold_countries.csv 文件中。")
```

结果已保存到 first_medal_and_gold_countries.csv 文件中。

[80]:

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

# 加载数据
file_path = 'Generated\\first_medal_and_gold_countries.csv'
data = pd.read_csv(file_path)

# 绘制折线图
plt.figure(figsize=(14, 7))

# 绘制第一次获得奖牌的国家数
plt.plot(data['Year'], data['First Medal Countries'], label='First Medal␣
  ↪Countries', marker='o')

# 绘制第一次获得金牌的国家数
plt.plot(data['Year'], data['First Gold Countries'], label='First Gold␣
  ↪Countries', marker='o')

# 添加标题和标签
plt.title('Number of Countries Winning Medals or Gold for the First Time in␣
  ↪Each Olympic Year')
plt.xlabel('Year')
plt.ylabel('Number of Countries')
plt.legend()
```

```
# 显示网格
plt.grid(True)

# 显示图表
plt.show()
```



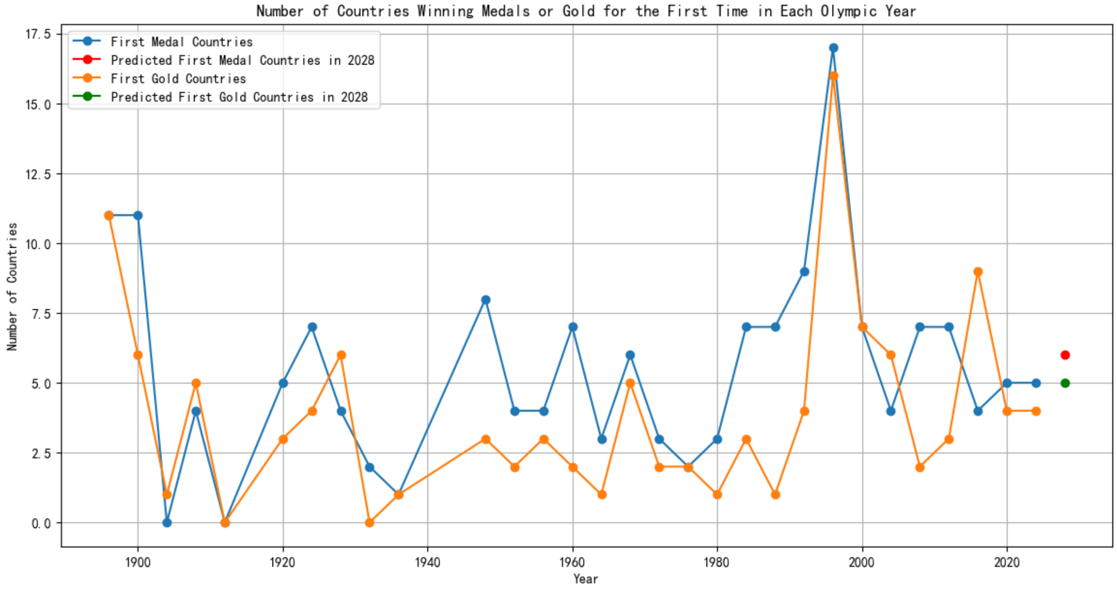
4.2.2 线性回归预测

[81]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# 加载数据
file_path = 'Generated\\first_medal_and_gold_countries.csv'
data = pd.read_csv(file_path)

# 准备数据
years = data['Year'].values.reshape(-1, 1)
```

```
first_medal_countries = data['First Medal Countries'].values
first_gold_countries = data['First Gold Countries'].values

# 训练线性回归模型
model_medal = LinearRegression()
model_medal.fit(years, first_medal_countries)

model_gold = LinearRegression()
model_gold.fit(years, first_gold_countries)

# 预测 2028 年的值
year_2028 = np.array([2028]).reshape(-1, 1)
pre_medal_2028 = [round(model_medal.predict(year_2028)[0])]
pre_gold_2028 = [round(model_gold.predict(year_2028)[0])]

# 绘制折线图
plt.figure(figsize=(14, 7))

# 绘制第一次获得奖牌的国家数
plt.plot(data['Year'], data['First Medal Countries'], label='First Medal Countries', marker='o')
plt.plot([2028], pre_medal_2028, marker='o', color='red', label='Predicted First Medal Countries in 2028')

# 绘制第一次获得金牌的国家数
plt.plot(data['Year'], data['First Gold Countries'], label='First Gold Countries', marker='o')
plt.plot([2028], pre_gold_2028, marker='o', color='green', label='Predicted First Gold Countries in 2028')

# 添加标题和标签
plt.title('Number of Countries Winning Medals or Gold for the First Time in Each Olympic Year')
plt.xlabel('Year')
plt.ylabel('Number of Countries')
plt.legend()
```

```python

# 显示网格
plt.grid(True)

# 显示图表
plt.show()

# 输出预测结果
print(f"预测 2028 年首次获得奖牌的国家数量: {pre_medal_2028[0]}")
print(f"预测 2028 年首次获得金牌的国家数量: {pre_gold_2028[0]}")
```



```
预测 2028 年首次获得奖牌的国家数量: 6
预测 2028 年首次获得金牌的国家数量: 5
```

- 分析效果

[82]:
```python
# 计算模型的拟合度
r2_medal = model_medal.score(years, first_medal_countries)
r2_gold = model_gold.score(years, first_gold_countries)

print(f"首次获得奖牌的国家数量模型的拟合度 (R^2): {r2_medal:.2f}")
print(f"首次获得金牌的国家数量模型的拟合度 (R^2): {r2_gold:.2f}")
```

```
首次获得奖牌的国家数量模型的拟合度 (R^2): 0.02
首次获得金牌的国家数量模型的拟合度 (R^2): 0.02
```

- 这说明有 2% 的可能完全准确
- 考虑到数目只为整数，可能性会更大

[137]:
```
# 进行预测
pre_medal = model_medal.predict(years)
pre_gold = model_gold.predict(years)

# 四舍五入预测值
pre_medal_rounded = np.round(pre_medal)
pre_gold_rounded = np.round(pre_gold)

# 计算偏离
deviation_medal = np.abs(pre_medal_rounded - first_medal_countries)
deviation_gold = np.abs(pre_gold_rounded - first_gold_countries)

# 统计偏离超过 1 的次数
count_deviation_medal_over_1 = np.sum(deviation_medal > 1)
count_deviation_gold_over_1 = np.sum(deviation_gold > 1)

# 计算比例
proportion_deviation_medal_over_1 = count_deviation_medal_over_1 / len(years)
proportion_deviation_gold_over_1 = count_deviation_gold_over_1 / len(years)

# 输出比例
print(f"首次获得奖牌的国家数量预测偏离超过 1 的比例:␣
  ↪{proportion_deviation_medal_over_1:.2f}")
print(f"首次获得金牌的国家数量预测偏离超过 1 的比例:␣
  ↪{proportion_deviation_gold_over_1:.2f}")
print(f"首次获得奖牌的国家数量预测准确度:␣
  ↪{(1-proportion_deviation_medal_over_1)*100:.2f}%")
print(f"首次获得金牌的国家数量预测准确度:␣
  ↪{(1-proportion_deviation_gold_over_1)*100:.2f}%")
```

```
首次获得奖牌的国家数量预测偏离超过 1 的比例: 0.57
首次获得金牌的国家数量预测偏离超过 1 的比例: 0.67
首次获得奖牌的国家数量预测准确度: 43.33%
首次获得金牌的国家数量预测准确度: 33.33%
```

4.2.3 评估可能性

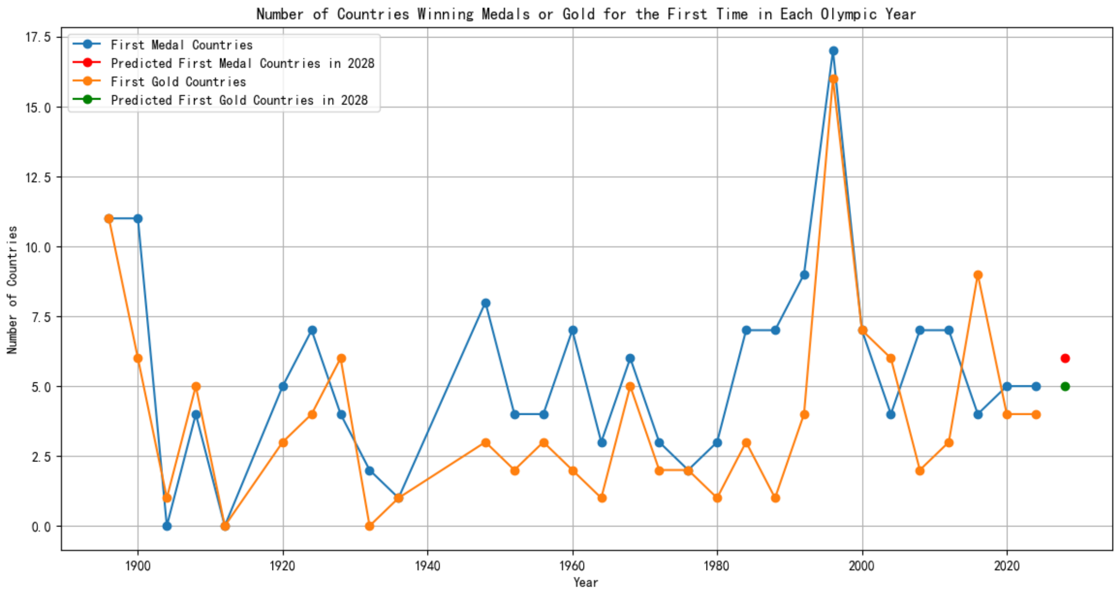
[135]:

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
import matplotlib.pyplot as plt

# 加载数据
file_path = 'Generated\\first_medal_and_gold_countries.csv'
data = pd.read_csv(file_path)

# 准备数据
years = data['Year'].values.reshape(-1, 1)
first_medal_countries = data['First Medal Countries'].values
first_gold_countries = data['First Gold Countries'].values

# 训练线性回归模型
model_medal = LinearRegression()
model_medal.fit(years, first_medal_countries)

model_gold = LinearRegression()
model_gold.fit(years, first_gold_countries)

# 预测 2028 年的值
year_2028 = np.array([[2028,1]])
pre_medal_2028 = round(model_medal.predict(year_2028.reshape(-1,1))[0])
pre_gold_2028 = round(model_gold.predict(year_2028.reshape(-1,1))[0])

# 使用 statsmodels 计算置信区间
X = sm.add_constant(years)  # 添加常数项
model_medal_sm = sm.OLS(first_medal_countries, X).fit()
```

```python
model_gold_sm = sm.OLS(first_gold_countries, X).fit()

print(model_medal_sm.summary())
print(model_gold_sm.summary())

# 预测 2028 年的值及其置信区间
year_2028_sm = sm.add_constant(year_2028).reshape(-1,2)  # 添加常数项
pre_medal_2028_sm = model_medal_sm.get_prediction(year_2028_sm).summary_frame()
pre_gold_2028_sm = model_gold_sm.get_prediction(year_2028_sm).summary_frame()

# 绘制折线图
plt.figure(figsize=(14, 7))

# 绘制第一次获得奖牌的国家数
plt.plot(data['Year'], data['First Medal Countries'], label='First Medal Countries', marker='o')
plt.plot([2028], pre_medal_2028, marker='o', color='red', label='Predicted First Medal Countries in 2028')

# 绘制第一次获得金牌的国家数
plt.plot(data['Year'], data['First Gold Countries'], label='First Gold Countries', marker='o')
plt.plot([2028], pre_gold_2028, marker='o', color='green', label='Predicted First Gold Countries in 2028')

# 添加置信区间
#plt.fill_between([2028], pre_medal_2028_sm['mean_ci_lower'], pre_medal_2028_sm['mean_ci_upper'], color='red', alpha=0.2)
#plt.fill_between([2028], pre_gold_2028_sm['mean_ci_lower'], pre_gold_2028_sm['mean_ci_upper'], color='green', alpha=0.2)

# 添加标题和标签
plt.title('Number of Countries Winning Medals or Gold for the First Time in Each Olympic Year')
plt.xlabel('Year')
plt.ylabel('Number of Countries')
plt.legend()

# 显示网格
plt.grid(True)

# 显示图表
plt.show()

# 输出预测结果及其置信区间
print(f"预测 2028 年首次获得奖牌的国家数量: {pre_medal_2028}")
print(f"预测 2028 年首次获得金牌的国家数量: {pre_gold_2028}")
print(f"预测 2028 年首次获得奖牌的国家数量的置信区间:␣
  ↪[{pre_medal_2028_sm['mean_ci_lower'][0]:.2f},␣
  ↪{pre_medal_2028_sm['mean_ci_upper'][0]:.2f}]")
print(f"预测 2028 年首次获得金牌的国家数量的置信区间:␣
  ↪[{pre_gold_2028_sm['mean_ci_lower'][0]:.2f},␣
  ↪{pre_gold_2028_sm['mean_ci_upper'][0]:.2f}]")
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.025
Model:                            OLS   Adj. R-squared:                 -0.010
Method:                 Least Squares   F-statistic:                    0.7110
Date:                Tue, 28 Jan 2025   Prob (F-statistic):              0.406
Time:                        02:34:37   Log-Likelihood:                -79.692
No. Observations:                  30   AIC:                             163.4
Df Residuals:                      28   BIC:                             166.2
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -22.3475     32.992     -0.677      0.504     -89.928      45.233
x1             0.0142      0.017      0.843      0.406      -0.020       0.049
==============================================================================
Omnibus:                       12.296   Durbin-Watson:                   1.450
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               11.745
```

```
Skew:                           1.241   Prob(JB):                      0.00282
Kurtosis:                       4.798   Cond. No.                     9.94e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 9.94e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.015
Model:                            OLS   Adj. R-squared:                 -0.020
Method:                 Least Squares   F-statistic:                    0.4312
Date:                Tue, 28 Jan 2025   Prob (F-statistic):              0.517
Time:                        02:34:37   Log-Likelihood:                -78.874
No. Observations:                  30   AIC:                             161.7
Df Residuals:                      28   BIC:                             164.5
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -17.1783     32.104     -0.535      0.597     -82.941      48.584
x1             0.0107      0.016      0.657      0.517      -0.023       0.044
==============================================================================
Omnibus:                       21.261   Durbin-Watson:                   1.473
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               29.640
Skew:                           1.775   Prob(JB):                     3.66e-07
Kurtosis:                       6.334   Cond. No.                     9.94e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 9.94e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```



```
预测 2028 年首次获得奖牌的国家数量: 6
预测 2028 年首次获得金牌的国家数量: 5
预测 2028 年首次获得奖牌的国家数量的置信区间: [-182373.15, 91731.90]
预测 2028 年首次获得金牌的国家数量的置信区间: [-168204.18, 98528.84]
```

4.3 第三问

- 您的模型还应考虑特定奥运会的赛事（数量和类型）。
- 探究赛事与各国所获奖牌数量之间的关系。
- 对于不同国家而言，哪些体育项目最为重要？原因何在？
- 主办国所选择的赛事如何影响比赛结果？

4.3.1 各国优势项目

预处理

[ ]:
```python
import pandas as pd

# 读取 CSV 文件
df = pd.read_csv('2025_Problem_C_Data\\summerOly_athletes.csv')

# 定义一个函数，将 Medal 列中的值转换为对应的奖牌类型
```

```
def medal_to_type(medal):
    if medal == 'Gold':
        return 'Gold'
    elif medal == 'Silver':
        return 'Silver'
    elif medal == 'Bronze':
        return 'Bronze'
    else:
        return 'No Medal'

# 应用函数转换 Medal 列
df['Medal_Type'] = df['Medal'].apply(medal_to_type)

# 按 NOC 和 Sport 分组，统计每种奖牌的数量
medal_counts = df[df['Medal_Type'] != 'No Medal'].groupby(['NOC', 'Sport',␣
 ↪'Medal_Type']).size().unstack(fill_value=0)

# 重置索引，以便将 NOC 和 Sport 作为列
medal_counts = medal_counts.reset_index()

# 填充缺失的奖牌类型列
medal_counts = medal_counts.fillna(0)

# 确保奖牌列是数值类型
medal_counts['Gold'] = medal_counts.get('Gold', 0).astype(int)
medal_counts['Silver'] = medal_counts.get('Silver', 0).astype(int)
medal_counts['Bronze'] = medal_counts.get('Bronze', 0).astype(int)

# 计算总奖牌数
medal_counts['Total'] = medal_counts[['Gold', 'Silver', 'Bronze']].sum(axis=1)

# 重新排列列的顺序
medal_counts = medal_counts[['NOC', 'Sport', 'Gold', 'Silver', 'Bronze',␣
 ↪'Total']]

# 保存结果到 CSV 文件
```

```
medal_counts.to_csv('Generated2\\medal_counts.csv', index=False)

# 显示结果
medal_counts.head()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[245], line 38
     35 medal_counts['Total'] = medal_counts[['Gold', 'Silver', 'Bronze']].
 ↪sum(axis=1)
     37 # 重新排列列的顺序
---> 38 medal_counts =␣
 ↪medal_counts[['NOC', 'Sport','Year', 'Gold', 'Silver', 'Bronze', 'Total']]
     40 # 保存结果到 CSV 文件
     41 medal_counts.to_csv('Generated2\\medal_counts.csv', index=False)

File c:
 ↪\Users\Ziqi\Documents\Python\2025-MCM-C\new_env\Lib\site-packages\pandas\core\frame.
 ↪py:4108, in DataFrame.__getitem__(self, key)
   4106     if is_iterator(key):
   4107         key = list(key)
-> 4108     indexer = self.columns._get_indexer_strict(key, "columns")[1]
   4110 # take() does not accept boolean indexers
   4111 if getattr(indexer, "dtype", None) == bool:

File c:
 ↪\Users\Ziqi\Documents\Python\2025-MCM-C\new_env\Lib\site-packages\pandas\core\indexes\base
 ↪py:6200, in Index._get_indexer_strict(self, key, axis_name)
   6197 else:
   6198     keyarr, indexer, new_indexer = self._reindex_non_unique(keyarr)
-> 6200 self._raise_if_missing(keyarr, indexer, axis_name)
   6202 keyarr = self.take(indexer)
   6203 if isinstance(key, Index):
   6204     # GH 42790 - Preserve name from an Index

File c:
 ↪\Users\Ziqi\Documents\Python\2025-MCM-C\new_env\Lib\site-packages\pandas\core\indexes\base
 ↪py:6252, in Index._raise_if_missing(self, key, indexer, axis_name)
```

```
   6249     raise KeyError(f"None of [{key}] are in the [{axis_name}]")
   6251 not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())
-> 6252 raise KeyError(f"{not_found} not in index")

KeyError: "['Year'] not in index"
```

4.3.2 得出优势项目

[162]:

```
import pandas as pd

# 读取 CSV 文件
df = pd.read_csv('Generated2\\medal_counts.csv')

# 定义一个函数，用于获取每个国家总奖牌榜和金牌榜前 2 的运动项目
def get_top_sports(group):
    #print(group)
    # 按总奖牌数降序排列
    total_sorted = group.sort_values(by='Total', ascending=False)
    # 按金牌数降序排列
    gold_sorted = group.sort_values(by='Gold', ascending=False)

    # 获取前 2 的运动项目
    total_top2 = total_sorted.head(2)[['Sport', 'Total']]
    gold_top2 = gold_sorted.head(2)[['Sport', 'Gold']]

    # 提取结果
    result = {
        'Gold1': gold_top2.iloc[0]['Sport'] if len(gold_top2) > 0 else None,
        'Gold2': gold_top2.iloc[1]['Sport'] if len(gold_top2) > 1 else None,
        'Total1': total_top2.iloc[0]['Sport'] if len(total_top2) > 0 else None,
        'Total2': total_top2.iloc[1]['Sport'] if len(total_top2) > 1 else None
    }

    #print(result)

    return pd.Series(result)
```

```
# 应用函数，获取每个国家的前 2 运动项目
results = df.groupby('NOC').apply(get_top_sports).reset_index()

# 保存到新的 CSV 文件
results.to_csv('Generated2\\top_sports.csv', index=False)

# 打印结果
print(results)
```

```
     NOC                  Gold1       Gold2      Total1                  Total2
0    AFG              Taekwondo        None   Taekwondo                    None
1    AHO                Sailing        None     Sailing                    None
2    AIN  Trampoline Gymnastics      Rowing      Tennis   Trampoline Gymnastics
3    ALB              Wrestling        None   Wrestling                    None
4    ALG              Athletics      Boxing   Athletics                  Boxing
..   …                     …         …         …                    …
152  VIE               Shooting   Taekwondo    Shooting               Taekwondo
153  WIF              Athletics        None   Athletics                    None
154  YUG               Handball  Water Polo  Basketball              Water Polo
155  ZAM              Athletics      Boxing   Athletics                  Boxing
156  ZIM                 Hockey    Swimming      Hockey                Swimming

[157 rows x 5 columns]

C:\Users\Ziqi\AppData\Local\Temp\ipykernel_20880\3729593612.py:31:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns.
This behavior is deprecated, and in a future version of pandas the grouping
columns will be excluded from the operation. Either pass `include_groups=False`
to exclude the groupings or explicitly select the grouping columns after groupby
to silence this warning.
  results = df.groupby('NOC').apply(get_top_sports).reset_index()
```

4.4 东道主效应

4.4.1 回归分析模型

- 考虑金牌数对奖牌数贡献

- 考虑项目数量增长对奖牌数贡献

[244]:

```
# 导入必要的库
import pandas as pd
import statsmodels.api as sm
import re
from sklearn.preprocessing import StandardScaler

# 读取 CSV 文件
medal_counts = pd.read_csv('2025_Problem_C_Data\\summerOly_medal_counts.csv')
hosts = pd.read_csv('2025_Problem_C_Data\\summerOly_hosts.csv')
project_amount = pd.read_csv('Generated\\Project_amount.csv')

# 数据预处理
# 去除 NOC 列和 Host 列中的非英文字符
medal_counts['NOC'] = medal_counts['NOC'].apply(lambda x: re.sub(r'[^a-zA-Z]',␣
 ↪'', x))
hosts['Host'] = hosts['Host'].apply(lambda x: re.sub(r'[^a-zA-Z]', '', x))

# 如果 Host 中的字符串包含 NOC 中的字符串，就将其替换为 NOC 的值
def replace_host(row):
    for noc in medal_counts['NOC']:
        if noc in row:
            return noc
    return row

hosts['Host'] = hosts['Host'].apply(replace_host)
hosts.loc[hosts['Host'] == 'LondonUnitedKingdom', 'Host'] = 'GreatBritain'

# 将 medal_counts 和 hosts 数据按 Year 列合并
merged_data = pd.merge(medal_counts, hosts, on='Year', how='left')

# 将 merged_data 和 project_amount 数据按 Year 列合并
merged_data = pd.merge(merged_data, project_amount, on='Year', how='left')

# 填充缺失值
merged_data['Host'] = merged_data['Host'].fillna('Not Host')
```

```
merged_data['Amount'] = merged_data['Amount'].fillna(merged_data['Amount'].
 ↪mean())

# 创建东道主标识变量
merged_data['Is_Host'] = merged_data['NOC'] == merged_data['Host']

# 选择特征和目标变量
X = merged_data[['Gold', 'Is_Host', 'Amount']]
y = merged_data['Total']

# 数据标准化
#scaler = StandardScaler()
#X_scaled = scaler.fit_transform(X)

# 添加常数项
#X_scaled = sm.add_constant(X_scaled)
X = sm.add_constant(X)

# 构建回归模型
model = sm.OLS(y, X.astype(float)).fit()

# 输出模型结果
print(model.summary())

# 对每个国家分别进行回归分析
results = []
for noc in merged_data['NOC'].unique():
    country_data = merged_data[merged_data['NOC'] == noc]
    if len(country_data) > 1:  # 确保每个国家至少有两条数据
        X_country = country_data[['Gold', 'Is_Host', 'Amount']]
        y_country = country_data['Total']

        # 数据标准化
        #X_country_scaled = scaler.fit_transform(X_country)

        # 添加常数项
        #X_country_scaled = sm.add_constant(X_country_scaled)
        X_country = sm.add_constant(X_country)

        # 构建回归模型
        model_country = sm.OLS(y_country, X_country.astype(float)).fit()

        # 保存结果
        results.append({
            'NOC': noc,
            'Is_Host_Coef': model_country.params['Is_Host'],
            'Is_Host_PValue': model_country.pvalues['Is_Host']
        })

# 将结果转换为 DataFrame
results_df = pd.DataFrame(results)

# 输出每个国家的东道主效应结果
print(results_df)

# 计算并输出 Is_Host 系数的均值
is_host_mean = results_df['Is_Host_Coef'].mean()
print(f"Is_Host 系数的均值: {is_host_mean:.2f}")
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Total   R-squared:                       0.942
Model:                            OLS   Adj. R-squared:                  0.942
Method:                 Least Squares   F-statistic:                     7701.
Date:                 周二, 28 1月 2025   Prob (F-statistic):               0.00
Time:                        06:12:02   Log-Likelihood:                -4405.0
No. Observations:                1435   AIC:                             8818.
Df Residuals:                    1431   BIC:                             8839.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
const          2.0739      0.447      4.636      0.000       1.196       2.951
Gold           2.4847      0.018    140.611      0.000       2.450       2.519
Is_Host        1.9067      1.055      1.807      0.071      -0.163       3.977
Amount         0.0005      0.001      0.689      0.491      -0.001       0.002
==============================================================================
Omnibus:                      448.481   Durbin-Watson:                   1.979
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             5078.169
Skew:                           1.119   Prob(JB):                         0.00
Kurtosis:                      11.940   Cond. No.                     4.23e+03
==============================================================================


Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 4.23e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
              NOC  Is_Host_Coef  Is_Host_PValue
0    UnitedStates      5.446768        0.652329
1          Greece      5.424184        0.311642
2         Germany     16.535894        0.213560
3          France     14.879555        0.030750
4    GreatBritain      1.250883        0.837392
..            …             …             …
122        Cyprus      0.000000             NaN
123     Guatemala      0.000000             NaN
124          Fiji      0.000000             NaN
125        Jordan      0.000000             NaN
126        Kosovo      0.000000             NaN

[127 rows x 3 columns]
Is_Host 系数的均值: 1.01
```

- Is_Host 系数均值明显为正，说明成为东道主对总奖牌数有正面促进作用。

4.5 伟大教练效应

4.5.1 预处理

[247]:

```python
import pandas as pd

# 读取 CSV 文件
df = pd.read_csv('2025_Problem_C_Data\\summerOly_athletes.csv')

# 将 Medal 列中的值转换为 Gold, Silver, Bronze
df['Medal'] = df['Medal'].replace({'No medal': None, 'Gold': 'Gold', 'Silver':␣
 ↪'Silver', 'Bronze': 'Bronze'})

# 按 NOC, Year, Sport 分组，统计每种奖牌的数量
medal_counts = df.groupby(['NOC', 'Year', 'Sport'])['Medal'].value_counts().
 ↪unstack(fill_value=0)

# 重置索引，将 NOC, Year, Sport 作为列
medal_counts = medal_counts.reset_index()

# 填充缺失的奖牌类型列
medal_counts = medal_counts.fillna(0)

# 计算 Total 列
medal_counts['Total'] = medal_counts['Gold'] + medal_counts['Silver'] +␣
 ↪medal_counts['Bronze']

# 重新排列列的顺序
medal_counts = medal_counts[['NOC', 'Sport', 'Year', 'Gold', 'Silver',␣
 ↪'Bronze', 'Total']]

# 显示结果
print(medal_counts)
medal_counts.to_csv('Generated2\\sports_medal_counts.csv')
```

```
Medal  NOC       Sport  Year  Gold  Silver  Bronze  Total
0      AFG  Taekwondo  2008     0       0       1      1
1      AFG  Taekwondo  2012     0       0       1      1
```

```
2       AHO    Sailing  1988     0       1       0       1
3       AIN     Rowing  2024     0       1       0       1
4       AIN     Tennis  2024     0       2       0       2
…     …         …    …    …      …      …      …
6740    ZAM  Athletics  1996     0       1       0       1
6741    ZAM  Athletics  2024     0       0       1       1
6742    ZIM     Hockey  1980    15       0       0      15
6743    ZIM   Swimming  2004     1       1       1       3
6744    ZIM   Swimming  2008     1       3       0       4

[6745 rows x 7 columns]
```

4.5.2 LSTM + Transformer 模型

[ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Model
from tensorflow.keras.layers import LSTM, Dense, Input, TimeDistributed,␣
  ↪MultiHeadAttention, LayerNormalization, Dropout
from tensorflow.keras.optimizers import Adam

# 1. 数据准备
# 读取 CSV 文件
data = pd.read_csv('Generated2\\sports_medal_counts.csv')

# 2. 数据预处理
# 归一化处理
def preprocess_data(data, time_step=1):
    data = data.sort_values(by='Year')
    data.set_index('Year', inplace=True)
    scaler = MinMaxScaler()
    data_scaled = scaler.fit_transform(data)
    X, Y = create_dataset(data_scaled, time_step)
    X = X.reshape(X.shape[0], X.shape[1], 1)
    return X, Y, scaler
```

```
# 创建时间序列数据
def create_dataset(data, time_step=1):
    X, Y = [], []
    for i in range(len(data) - time_step - 1):
        a = data[i:(i + time_step), 0]
        X.append(a)
        Y.append(data[i + time_step, 0])
    return np.array(X), np.array(Y)

# 3. 构建 LSTM + Transformer 模型
def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout):
    x = MultiHeadAttention(key_dim=head_size, num_heads=num_heads)(inputs,␣
 ↪inputs)
    x = Dropout(dropout)(x)
    x = LayerNormalization(epsilon=1e-6)(x)
    x = Dense(ff_dim, activation="relu")(x)
    x = Dense(inputs.shape[-1])(x)
    x = Dropout(dropout)(x)
    x = LayerNormalization(epsilon=1e-6)(x)
    return x

def build_model(time_step):
    input_shape = (time_step, 1)
    inputs = Input(shape=input_shape)
    x = LSTM(50, return_sequences=True)(inputs)
    x = transformer_encoder(x, head_size=160, num_heads=4, ff_dim=4, dropout=0.
 ↪25)
    x = LSTM(50)(x)
    outputs = Dense(1)(x)
    model = Model(inputs, outputs)
    model.compile(optimizer=Adam(learning_rate=1e-4), loss='mean_squared_error')
    return model

# 4. 遍历所有国家和项目
results = []
```

```
for country in data['NOC'].unique():
    for sport in data[data['NOC'] == country]['Sport'].unique():
        project_data = data[(data['NOC'] == country) & (data['Sport'] == sport)][['Year', 'Total']]

        # 如果数据量太少，跳过
        if len(project_data) < 10:
            continue

        # 数据预处理
        X, Y, scaler = preprocess_data(project_data, time_step=5)

        # 训练模型
        model = build_model(time_step=5)
        model.fit(X, Y, epochs=100, batch_size=32, verbose=0)

        # 使用模型进行预测
        Y_pred = model.predict(X)

        # 反归一化
        Y_pred = scaler.inverse_transform(Y_pred)
        Y_true = scaler.inverse_transform(Y.reshape(-1, 1))

        # 计算异常分数
        anomaly_score = np.abs(Y_true - Y_pred)

        # 找出异常上升或下降的年份
        threshold = np.percentile(anomaly_score, 95)
        anomaly_indices = np.where(anomaly_score.flatten() > threshold)[0]

        # 调整索引以匹配原始数据
        anomaly_years = project_data.index[anomaly_indices + 5]

        # 区分异常上升和异常下降
        for year in anomaly_years:
```

```python
            index = anomaly_indices[anomaly_years.get_loc(year)]
            if index - 5 < 0:
                continue  # 跳过索引超出范围的情况
            if Y_pred[index - 5] > Y_true[index - 5]:
                anomaly_type = '下降'
            else:
                anomaly_type = '上升'
            results.append((country, sport, year, anomaly_type))

# 5. 输出结果
results_df = pd.DataFrame(results, columns=['国家', '项目', '年份', '异常类型'])
print(results_df)
```

```
1/1               0s 153ms/step
1/1               0s 202ms/step
WARNING:tensorflow:5 out of the last 5 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x0000018E0A42EC00> triggered tf.function retracing. Tracing is expensive and
the excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1               0s 173ms/step
WARNING:tensorflow:6 out of the last 6 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x0000018E10E33100> triggered tf.function retracing. Tracing is expensive and
the excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1               0s 153ms/step
```

```
1/1             0s 150ms/step
1/1             0s 162ms/step
1/1             0s 155ms/step
1/1             0s 152ms/step
1/1             0s 169ms/step
1/1             0s 156ms/step
1/1             0s 163ms/step
1/1             0s 166ms/step
1/1             0s 161ms/step
1/1             0s 155ms/step
1/1             0s 152ms/step
1/1             0s 156ms/step
1/1             0s 151ms/step
1/1             0s 155ms/step
1/1             0s 153ms/step
1/1             0s 154ms/step
1/1             0s 151ms/step
1/1             0s 152ms/step
1/1             0s 277ms/step
1/1             0s 166ms/step
1/1             0s 163ms/step
1/1             0s 155ms/step
1/1             0s 278ms/step
1/1             0s 158ms/step
1/1             0s 150ms/step
1/1             0s 154ms/step
1/1             0s 153ms/step
1/1             0s 280ms/step
1/1             0s 153ms/step
1/1             0s 150ms/step
1/1             0s 149ms/step
1/1             0s 152ms/step
1/1             0s 154ms/step
1/1             0s 156ms/step
1/1             0s 148ms/step
1/1             0s 312ms/step
1/1             0s 290ms/step
```

```
1/1             0s 280ms/step
1/1             0s 165ms/step
1/1             0s 166ms/step
1/1             0s 165ms/step
1/1             0s 166ms/step
1/1             1s 1s/step
1/1             0s 153ms/step
1/1             0s 153ms/step
1/1             0s 152ms/step
1/1             0s 155ms/step
1/1             0s 152ms/step
1/1             0s 158ms/step
```