Project Name - Olympics Data Analysis _ ML _ FA _ DA projects (Part 1)

Project Type - Data Analysis

Industry - Unified Mentor

Contribution - Individual

Member Name - Hare Krishana Mishra

Task - 1

Project Summary -

Project Description:

This project focuses on analyzing Summer Olympic data from 1976 to 2008, using data-driven approaches to extract meaningful insights, visualize patterns, and build a predictive model to identify athletes or events most likely to secure a medal. By cleaning and encoding the dataset, various trends such as medal distribution across countries, genders, sports, and years were examined. Additionally, a logistic regression model was developed to classify whether an athlete or event won a medal based on categorical features.

Objective:

- To perform exploratory data analysis (EDA) on historical Olympic data.
- To uncover key trends related to medal wins by country, gender, sport, and year.
- To preprocess and encode data for modeling.
- To train a logistic regression model that predicts medal wins based on encoded features.

Key Project Details:

Dataset: Summer Olympic Medals (1976–2008)

Total Records Analyzed: 15,433

Features Used: Country, Sport, Gender, Event_gender, Year, Medal

Techniques Applied:

Data Cleaning and Handling Missing Values

Label Encoding of Categorical Variables

Visualizations with Seaborn & Matplotlib

Binary Classification using Logistic Regression

Evaluation Metrics: Accuracy Score, Confusion Matrix, Classification Report

Tech Stack:

Python, Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn

Key Insights:

Top-performing countries and sports identified

Medal distribution across genders analyzed

Participation and medal trends visualized by Olympic year

Let's Begin:-

Step 1: Data Preparation

In [ ]:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:

```
# Load the dataset (assume CSV format
df = pd.read_csv('/content/Summer-Olympic-medals-1976-to-2008 (1).csv', encoding='latin1
```

In [ ]:

```
# Check the first few rows of the dataset
df.head()
```

Out[ ]:

	City	Year	Sport	Discipline	Event	Athlete	Gender	Country_Code	Country	Ev
0	Montreal	1976.0	Aquatics	Diving	3m springboard	KÖHLER, Christa	Women	GDR	East Germany	
1	Montreal	1976.0	Aquatics	Diving	3m springboard	KOSENKOV, Aleksandr	Men	URS	Soviet Union	
2	Montreal	1976.0	Aquatics	Diving	3m springboard	BOGGS, Philip George	Men	USA	United States	
3	Montreal	1976.0	Aquatics	Diving	3m springboard	CAGNOTTO, Giorgio Franco	Men	ITA	Italy	
4	Montreal	1976.0	Aquatics	Diving	10m platform	WILSON, Deborah Keplar	Women	USA	United States	

In [ ]:

```
# Summary of the dataset
print(df.info())
print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15433 entries, 0 to 15432
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   City          15316 non-null  object 
 1   Year          15316 non-null  float64
 2   Sport         15316 non-null  object 
 3   Discipline    15316 non-null  object 
 4   Event         15316 non-null  object 
 5   Athlete       15316 non-null  object 
 6   Gender        15316 non-null  object 
 7   Country_Code  15316 non-null  object 
 8   Country       15316 non-null  object 
 9   Event_gender  15316 non-null  object 
 10  Medal         15316 non-null  object 
dtypes: float64(1), object(10)
memory usage: 1.3+ MB
None
               Year
count  15316.000000
mean    1993.620789
std       10.159851
min     1976.000000
25%     1984.000000
50%     1996.000000
75%     2004.000000
max     2008.000000
```

Step 2: Data Cleaning

In [ ]:

```
# Check for missing values
print(df.isnull().sum())
```

```
City            117
Year            117
Sport           117
Discipline      117
Event           117
Athlete         117
Gender          117
Country_Code    117
Country         117
Event_gender    117
Medal           117
dtype: int64
```

In [ ]:

```
df.shape
```

Out[ ]:

```
(15433, 11)
```

In [ ]:

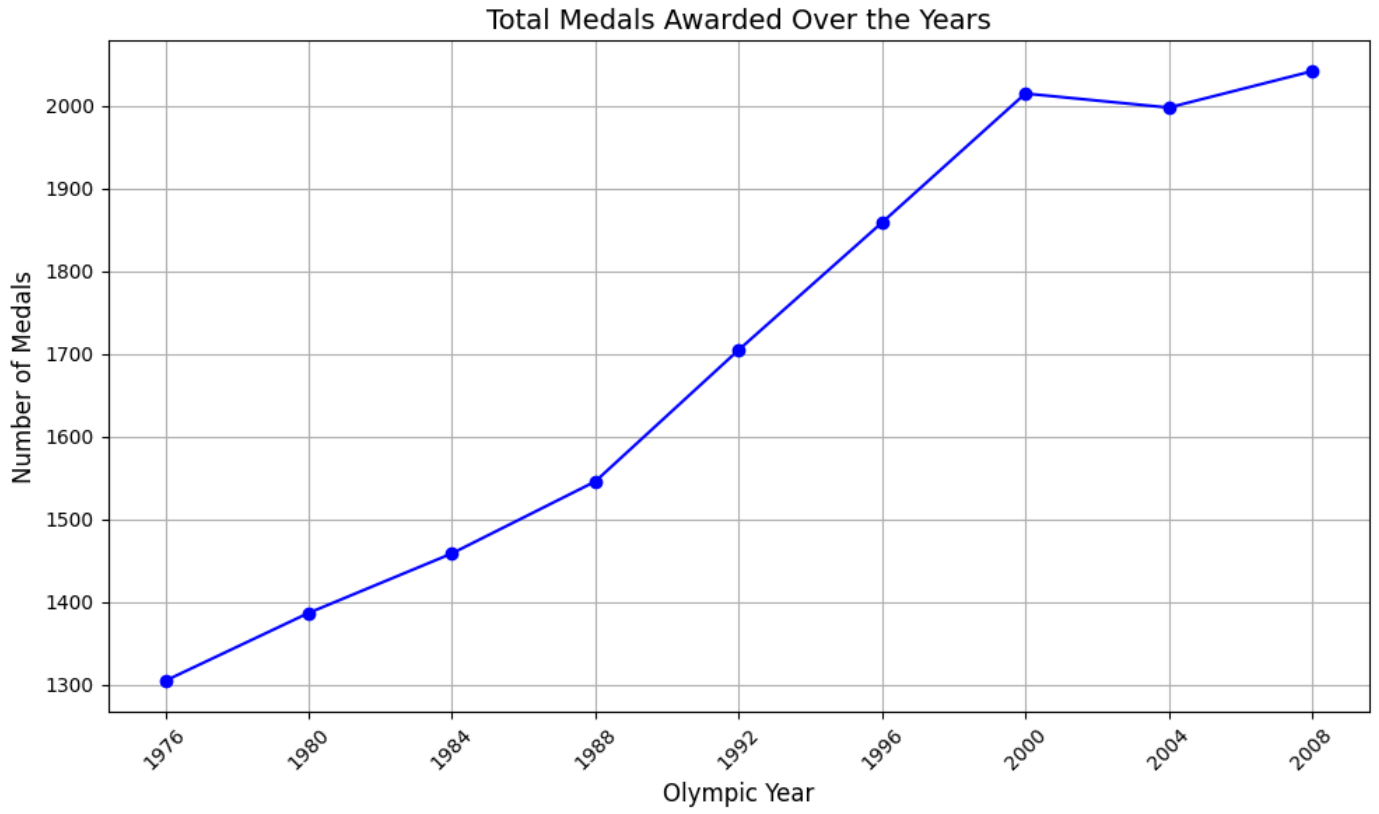
```
# Drop rows with missing values if any
df_cleaned = df.dropna()
```

Step 3: Exploratory Data Analysis (EDA)

3.1 Medals Won Over the Years

In [ ]:

```
# Grouping by Year and counting the medals won
medals_over_years = df_cleaned.groupby('Year')['Medal'].count().sort_index()

# Plotting the trend of medals won over the years
plt.figure(figsize=(10, 6))
plt.plot(medals_over_years.index.astype(int), medals_over_years.values, marker='o', line
plt.title("Total Medals Awarded Over the Years", fontsize=14)
plt.xlabel("Olympic Year", fontsize=12)
plt.ylabel("Number of Medals", fontsize=12)
plt.xticks(medals_over_years.index.astype(int), rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



3.2 Total Medal Count by Country

In [ ]:

```
# Total medals won by each country
medals_by_country = df_cleaned.groupby('Country')['Medal'].count().sort_values(ascending
medals_by_country
```

Out[ ]:

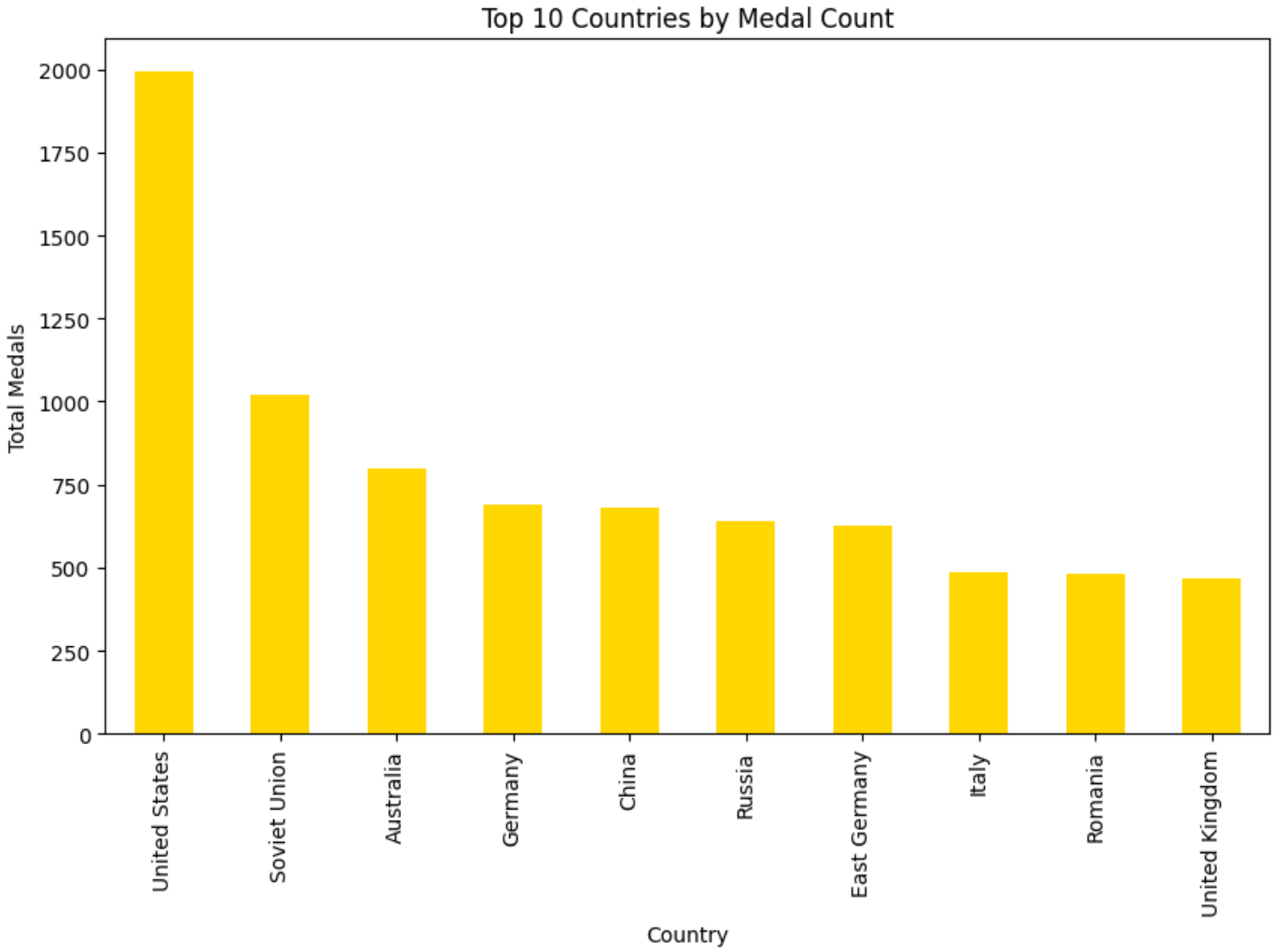
Country	Medal
United States	1992
Soviet Union	1021
Australia	798
Germany	691
China	679
...	...
Sri Lanka	1
Togo	1
United Arab Emirates	1
Uruguay	1
Virgin Islands*	1

127 rows × 1 columns

dtype: int64

In [ ]:

```
# Plotting the top 10 countries by medals
plt.figure(figsize=(10, 6))
medals_by_country.head(10).plot(kind='bar', color='gold')
plt.title("Top 10 Countries by Medal Count")
plt.xlabel("Country")
plt.ylabel("Total Medals")
plt.show()
```



3.3 Top Athletes with Most Medals

In [ ]:

```
# Group by Athlete and count the number of medals
athlete_medal_count = df_cleaned.groupby('Athlete')['Medal'].count().sort_values(ascendi
athlete_medal_count
```

Out[ ]:

Athlete	Medal
PHELPS, Michael	16
FISCHER, Birgit	12
ANDRIANOV, Nikolay	12
TORRES, Dara	12
THOMPSON, Jenny	12
...	...
ZVYAGINTSEV, Viktor	1
ZWEHL, Julia	1
ZWERING, Klaas-Erik	1

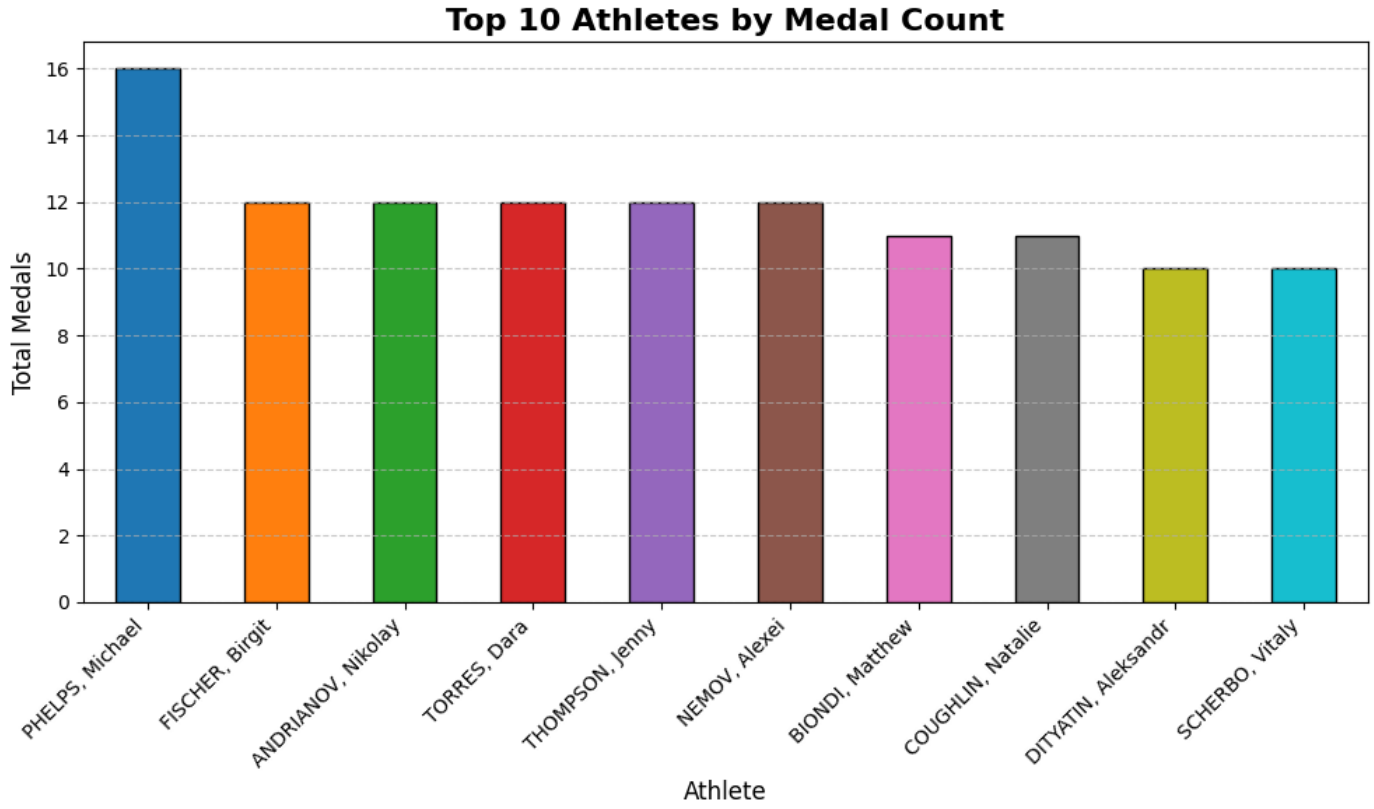
Athlete	Medal
ZUEVA, Natalia	1
ZUIJDWEG, Martijn	1

11337 rows × 1 columns

dtype: int64

In [ ]:

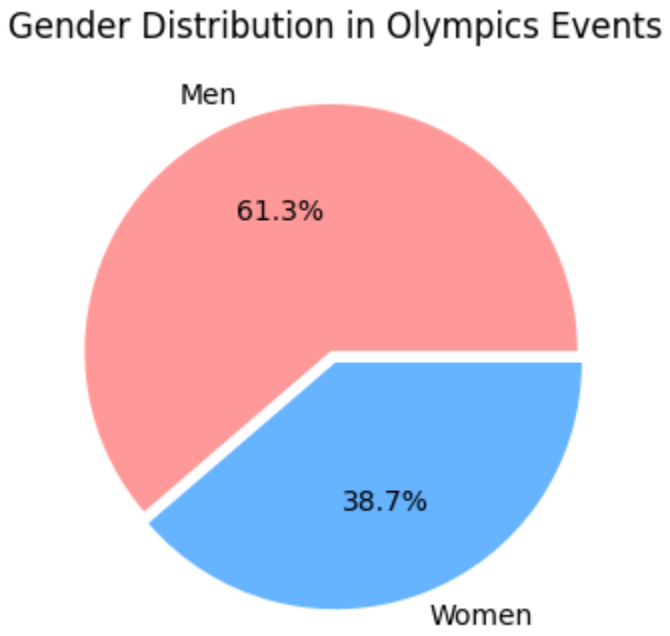
```
# Get a list of 10 different colors
colors = cm.tab10(np.linspace(0, 1, 10))

plt.figure(figsize=(10, 6))
athlete_medal_count.head(10).plot(
    kind='bar',
    color=colors,
    edgecolor='black'
)
plt.title("Top 10 Athletes by Medal Count", fontsize=16, fontweight='bold')
plt.xlabel("Athlete", fontsize=12)
plt.ylabel("Total Medals", fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



3.4 Gender Distribution in Events

In [ ]:

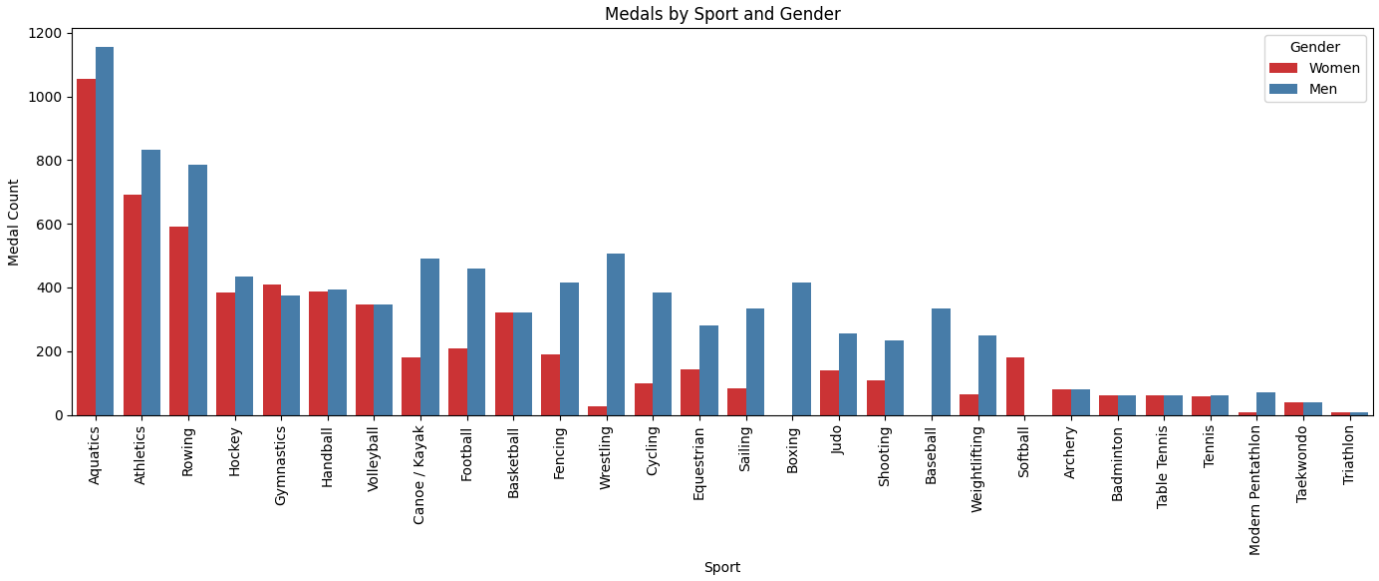
```
# Gender distribution in events
gender_distribution = df_cleaned['Gender'].value_counts()
# Plotting gender distribution
plt.figure(figsize=(6, 4))
gender_distribution.plot(kind='pie', autopct='%1.1f%%',
colors=['#ff9999','#66b3ff'], explode=[0.05, 0])
plt.title("Gender Distribution in Olympics Events")
plt.ylabel('')
plt.show()
```



3.5 Medals by Sport and Gender

In [ ]:

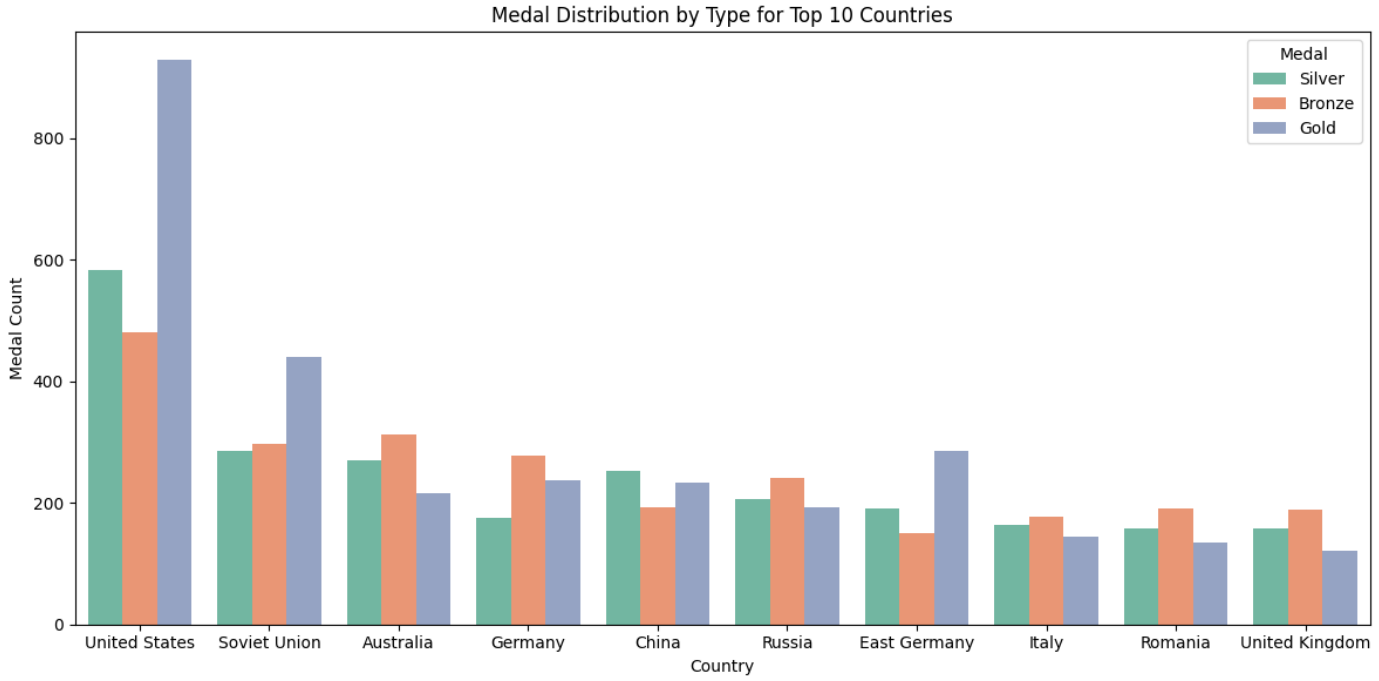
```
plt.figure(figsize=(14, 6))
sns.countplot(
    data=df_cleaned,
    x='Sport',
    hue='Gender',
    order=df_cleaned['Sport'].value_counts().index,
    palette='Set1'
)
plt.xticks(rotation=90)
plt.title("Medals by Sport and Gender")
plt.xlabel("Sport")
plt.ylabel("Medal Count")
plt.tight_layout()
plt.show()
```



3.6 Medal Distribution by Type for Top 10 Countries

In [ ]:

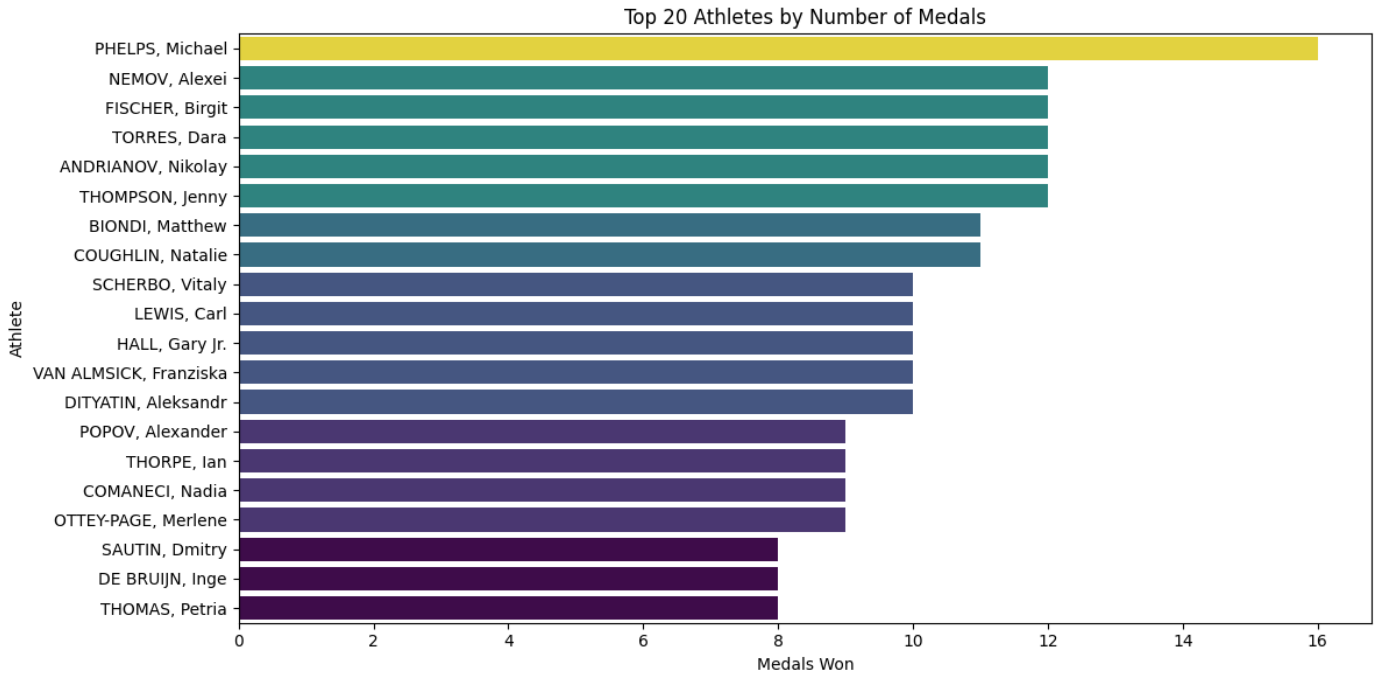
```
top_10_countries = df_cleaned['Country'].value_counts().head(10).index

plt.figure(figsize=(12, 6))
sns.countplot(
    data=df_cleaned[df_cleaned['Country'].isin(top_10_countries)],
    x='Country',
    hue='Medal',
    order=top_10_countries,
    palette='Set2'
)
plt.title("Medal Distribution by Type for Top 10 Countries")
plt.xlabel("Country")
plt.ylabel("Medal Count")
plt.tight_layout()
plt.show()
```



3.7 Top 20 Athletes by Number of Medals

In [ ]:

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

# Prepare the top 20 athlete data
athlete_medal_counts = df_cleaned['Athlete'].value_counts().head(20).reset_index()
athlete_medal_counts.columns = ['Athlete', 'Medal_Count']

# Add a fake hue column to enable palette usage
athlete_medal_counts['ColorGroup'] = athlete_medal_counts['Medal_Count']

# Plot with custom palette and no legend
plt.figure(figsize=(12, 6))
sns.barplot(
    data=athlete_medal_counts,
    x='Medal_Count',
    y='Athlete',
    hue='ColorGroup',
    palette='viridis',   # 🎨 You can try: 'coolwarm', 'rocket', 'cubehelix', 'mako', et
    dodge=False,
    legend=False
)
plt.title("Top 20 Athletes by Number of Medals")
plt.xlabel("Medals Won")
plt.ylabel("Athlete")
plt.tight_layout()
plt.show()
```



Step 4: Predictive Analysis (Machine Learning)

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report
```

In [ ]:

```
# Encode categorical variables using LabelEncoder
le = LabelEncoder()
df_cleaned['Country_Code'] = le.fit_transform(df_cleaned['Country_Code'])
df_cleaned['Sport'] = le.fit_transform(df_cleaned['Sport'])
df_cleaned['Gender'] = le.fit_transform(df_cleaned['Gender'])
df_cleaned['Event_gender'] = le.fit_transform(df_cleaned['Event_gender'])
df_cleaned['Medal'] = df_cleaned['Medal'].map({'Gold': 1,'Silver': 1, 'Bronze': 1, np.na
```

```
/tmp/ipython-input-3408056398.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['Country_Code'] = le.fit_transform(df_cleaned['Country_Code'])
/tmp/ipython-input-3408056398.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['Sport'] = le.fit_transform(df_cleaned['Sport'])
/tmp/ipython-input-3408056398.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['Gender'] = le.fit_transform(df_cleaned['Gender'])
/tmp/ipython-input-3408056398.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['Event_gender'] = le.fit_transform(df_cleaned['Event_gender'])
/tmp/ipython-input-3408056398.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['Medal'] = df_cleaned['Medal'].map({'Gold': 1,'Silver': 1, 'Bronze': 1, np.
nan: 0})
```

In [ ]:

```
# Features and target
X = df_cleaned[['Country_Code', 'Sport', 'Gender','Event_gender']]
y = df_cleaned['Medal']
```

In [ ]:

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3, random_state=42)
```

In [ ]:

```
y = y.fillna(0)
```

In [ ]:

```
y = y.astype(int)
```

In [ ]:

```
model = LogisticRegression(max_iter=1000)
```

Step 5: Conclusion and Insights

- Top Performing Countries: We identified which countries won the most medals.
- Top Athletes: We identified athletes who won the most medals.
- Gender Participation: The gender distribution in different sports events was explored.
- Trend of Medals Over Years: We visualized the trend of medal wins over the years.

The logistic regression model allowed us to predict whether an athlete would win a medal based on various attributes like country, sport, and gender. This project can be extended by adding more sophisticated machine learning models (like decision trees or random forests), and further fine-tuning the models by including more features.