🧰 Step 1: Importing Libraries

In this step, I imported the essential Python libraries for data analysis and visualization:

- **pandas** → for data manipulation
- **numpy** → for numerical operations
- **matplotlib & seaborn** → for clean, professional visualizations

I also applied a default seaborn style to keep the visualizations clean and consistent.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)


plt.style.use('seaborn-v0_8')
```

📂 Step 2: Loading the Dataset

Here I loaded the *Olympics Dataset (* `athlete_events.csv` *)*, which contains data of over **271,000 athlete entries** from **1896 to 2016**.

Key Actions:

- Used `pd.read_csv()` to load the file.
- Verified dataset structure using `df.head()` .

Columns: ID, Name, Sex, Age, Height, Weight, Team, NOC, Games, Year, Season, City, Sport, Event, Medal.

In [2]:
```
file_path = 'athlete_events.csv'  # make sure the file is in the same folder as the notebook
df = pd.read_csv(file_path)
df.head()
```

Out[2]:

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	Ci
0	1	A Dijiang	M	24.0	180.0	80.0	China	CHN	1992 Summer	1992	Summer	Barcelo
1	2	A Lamusi	M	23.0	170.0	60.0	China	CHN	2012 Summer	2012	Summer	Londo
2	3	Gunnar Nielsen Aaby	M	24.0	NaN	NaN	Denmark	DEN	1920 Summer	1920	Summer	Antwerp
3	4	Edgar Lindenau Aabye	M	34.0	NaN	NaN	Denmark/Sweden	DEN	1900 Summer	1900	Summer	Pa
4	5	Christine Jacoba Aaftink	F	21.0	185.0	82.0	Netherlands	NED	1988 Winter	1988	Winter	Calga

📜 Step 3: Data Inspection

I used:

- `df.info()` → To check column types and missing values.
- `df.describe()` → To get descriptive statistics of numerical columns.
- `df.isnull().sum()` → To understand missing data distribution.

Key Findings:

- No missing values in critical columns like `Team`, `Year`, `Sport`.
- Missing values in `Age`, `Height`, `Weight` and `Medal`.
- `Medal` column has missing values for non-medalists (expected).

In [3]:
```
df.info()
df.describe()
df.isnull().sum().head(15)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271116 entries, 0 to 271115
Data columns (total 15 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   ID      271116 non-null  int64  
 1   Name    271116 non-null  object 
 2   Sex     271116 non-null  object 
 3   Age     261642 non-null  float64
 4   Height  210945 non-null  float64
 5   Weight  208241 non-null  float64
 6   Team    271116 non-null  object 
 7   NOC     271116 non-null  object 
 8   Games   271116 non-null  object 
 9   Year    271116 non-null  int64  
 10  Season  271116 non-null  object 
 11  City    271116 non-null  object 
 12  Sport   271116 non-null  object 
 13  Event   271116 non-null  object 
 14  Medal   39783 non-null   object 
dtypes: float64(3), int64(2), object(10)
memory usage: 31.0+ MB
```

Out[3]:

```
ID             0
Name           0
Sex            0
Age         9474
Height     60171
Weight     62875
Team           0
NOC            0
Games          0
Year           0
Season         0
City           0
Sport          0
Event          0
Medal     231333
dtype: int64
```

🛡️ Step 4: Protecting Raw Data

To maintain clean workflow and data integrity:

- Kept `df_raw` as the untouched original dataset.
- Worked with `df` as a separate exploration copy.
- Created `medal_data` for medal-based analysis.
- Created `age_series` to handle missing Age values without altering raw data.

This step ensures **reproducibility** and allows for **safe experimentation**.

In [4]:

```
df_raw = pd.read_csv('athlete_events.csv')  # raw stays untouched
df = df_raw.copy()                          # working copy for exploration
```

🧼 Step 5: Handling Missing Data (Smartly)

I used a **non-destructive approach** to handle missing values:

- Derived `Age_imputed` column using median age (instead of overwriting).
- Skipped imputing Height & Weight for now due to large missing percentage.
- Used `medal_data` to filter only rows where Medal is not null.

Why this is good practice:

- It keeps raw data intact.
- Allows transparent tracking of changes.
- Prevents introducing noise into the dataset.

👉 With this step complete, the dataset is clean and structured — ready for visual exploration.

In [5]:
```
# Option A: Derived column (preferred for transparency)
df = df.assign(Age_imputed = df['Age'].fillna(df['Age'].median()))

# Option B: Separate working dataframe
df_age = df.copy()
df_age['Age'] = df_age['Age'].fillna(df_age['Age'].median())
```

In [6]:
```
# Step 5: Final cleaned helper datasets
medal_data = df[df['Medal'].notna()]    # for medal analysis
age_series = df['Age'].dropna()         # for age distribution
# no need to reload df again
```

🏅 Step 6: Medal Trends Over Time

Goal: Understand how the total number of medals awarded has changed across Olympic history, and compare **Summer vs Winter** trends.

Method:

- Filtered to rows where `Medal` is present.
- Aggregated medal counts by `Year` (overall) and by `Year × Season` (Summer/Winter).
- Plotted line charts; added a rolling-average line to smooth year-to-year noise.

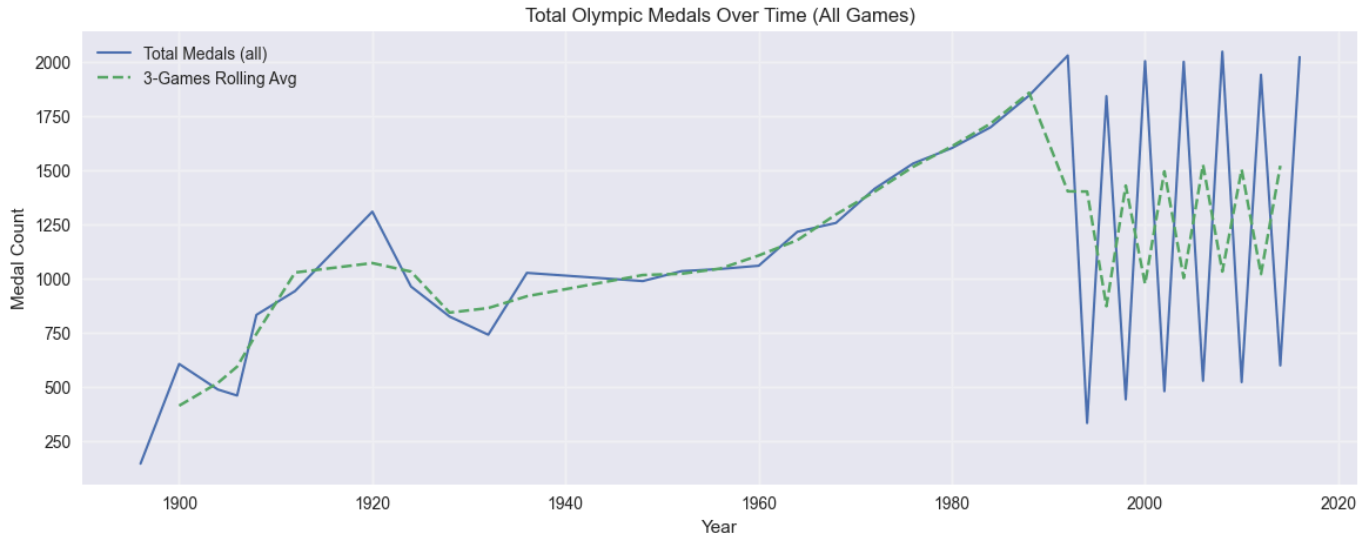
Why this matters:
This shows Olympic growth (more sports/events/participants) and helps spot historical dips (e.g., world wars, boycotts) or expansions (new events).

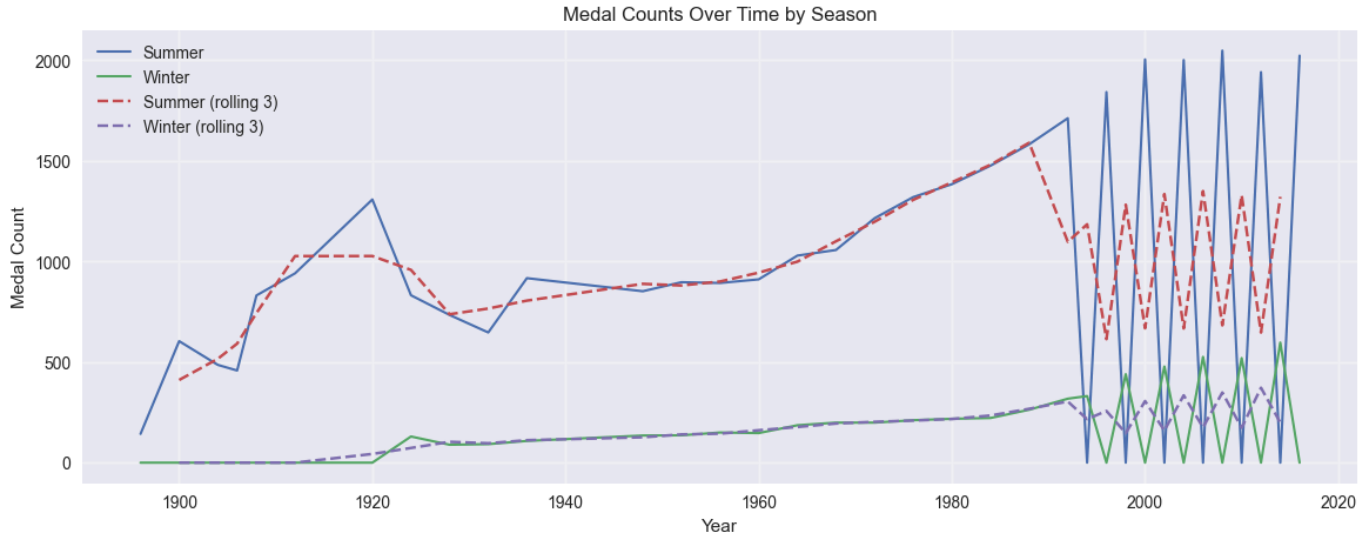
Notes / Caveats:

- More medals can reflect **more events**, not just stronger performances.
- Optional next step: normalize by number of **Events per Year** to compare fairly over time.

In [7]:
```
# --- Medal Trends Over Time ---

# 1) Filter to medal-winning rows
medal_data = df[df['Medal'].notna()].copy()

# 2) Overall medal count per year
medals_by_year = (
    medal_data.groupby('Year')
    .size()
    .reset_index(name='Medal_Count')
    .sort_values('Year')
)

# 3) Add a 3-Games rolling average to smooth the curve
medals_by_year['Medals_rolling3'] = medals_by_year['Medal_Count'].rolling(window=3, center=Tr

# 4) Plot overall medals per year
plt.figure(figsize=(14,5))
plt.plot(medals_by_year['Year'], medals_by_year['Medal_Count'], label='Total Medals (all)', l
```

```
plt.plot(medals_by_year['Year'], medals_by_year['Medals_rolling3'], label='3-Games Rolling Av
plt.title('Total Olympic Medals Over Time (All Games)')
plt.xlabel('Year')
plt.ylabel('Medal Count')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# 5) Summer vs Winter comparison
medals_by_year_season = (
    medal_data.groupby(['Year', 'Season'])
    .size()
    .reset_index(name='Medal_Count')
    .sort_values(['Year','Season'])
)

# Pivot to columns: Season = {Summer, Winter}
pivot_season = medals_by_year_season.pivot(index='Year', columns='Season', values='Medal_Coun

plt.figure(figsize=(14,5))
plt.plot(pivot_season.index, pivot_season.get('Summer', 0), label='Summer', linewidth=1.5)
plt.plot(pivot_season.index, pivot_season.get('Winter', 0), label='Winter', linewidth=1.5)
# Rolling averages for smoother curves
plt.plot(pivot_season.index, pivot_season.get('Summer', 0).rolling(3, center=True).mean(), li
plt.plot(pivot_season.index, pivot_season.get('Winter', 0).rolling(3, center=True).mean(), li
plt.title('Medal Counts Over Time by Season')
plt.xlabel('Year')
plt.ylabel('Medal Count')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```





🌍 Step 7: Top Countries by Total Medals

Goal: Identify the countries with the highest historical medal counts and see the Gold/Silver/Bronze mix.

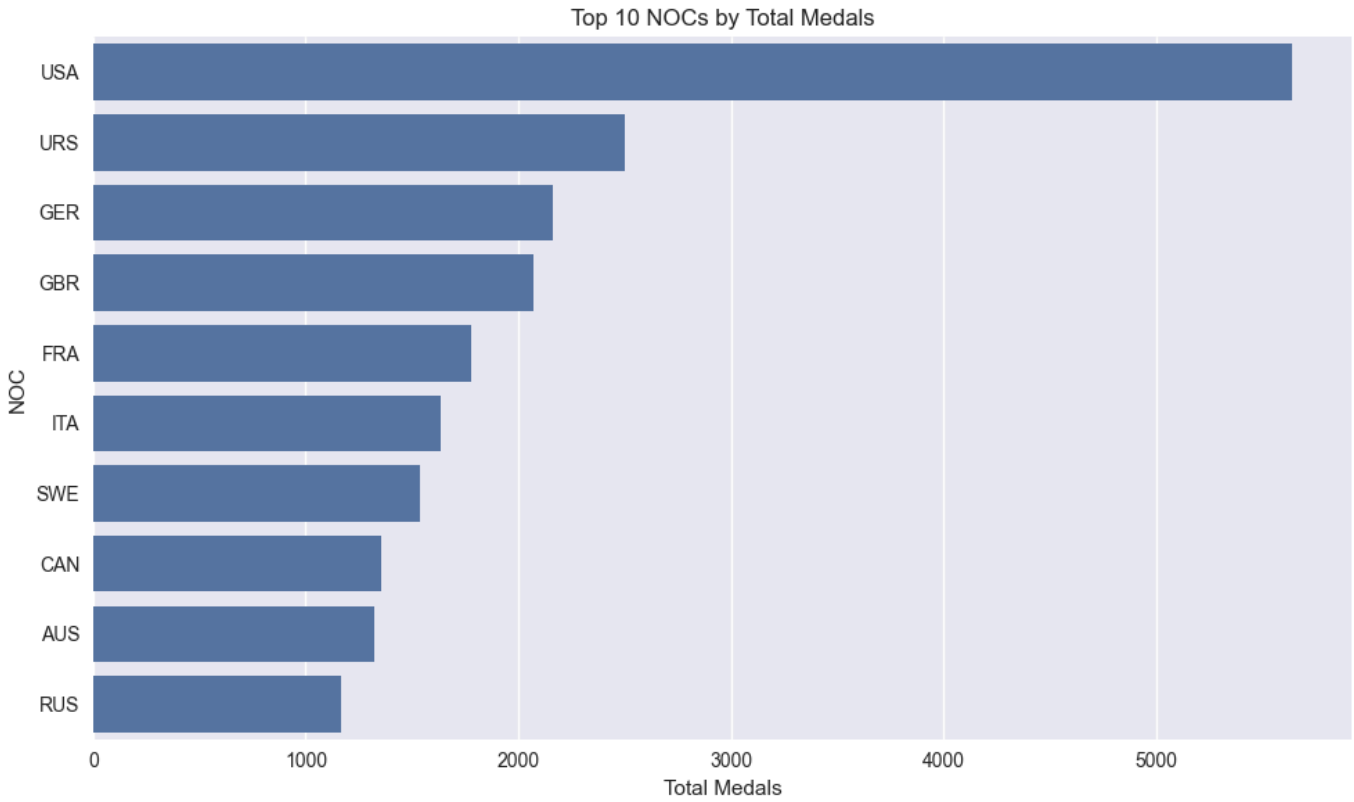
Method:

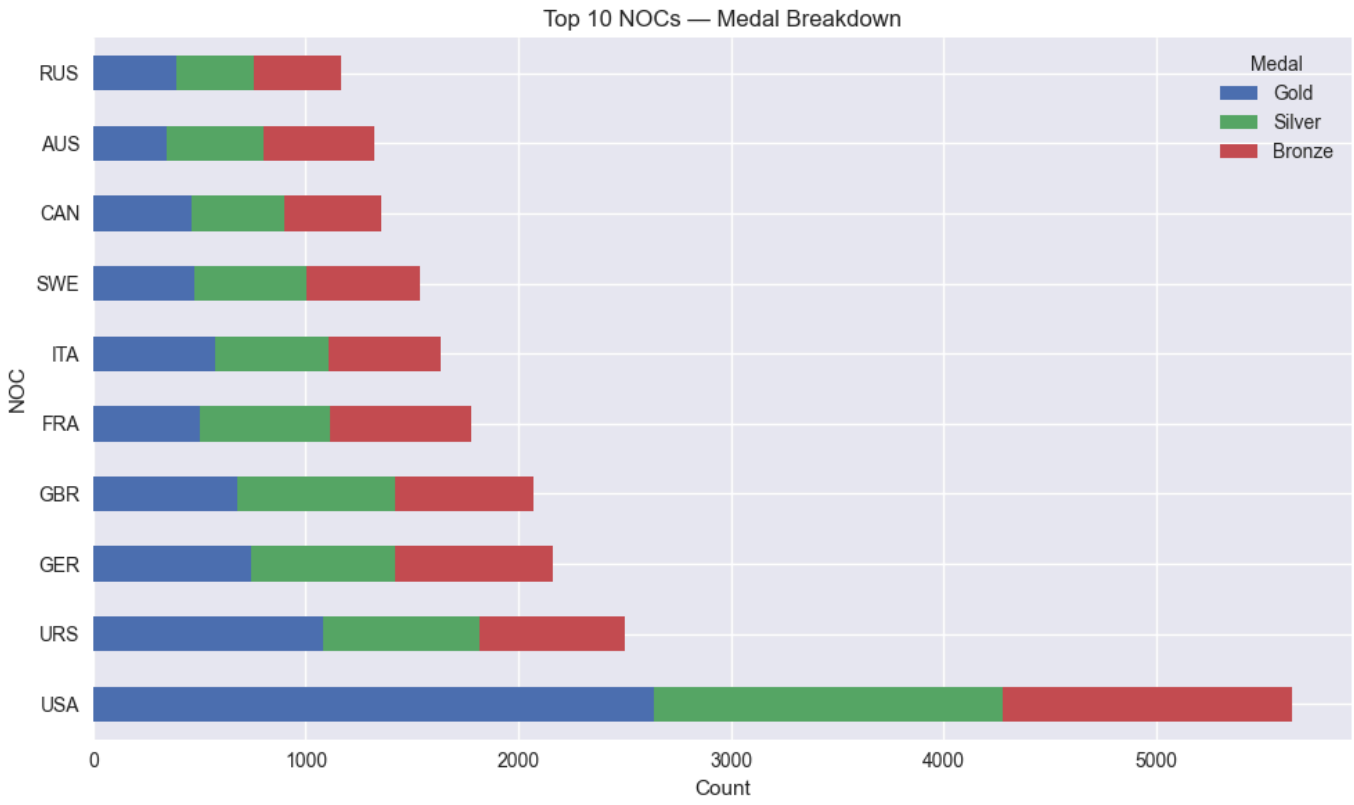
- Used `medal_data` (non-null `Medal` rows).
- Aggregated medals by `NOC` (stable country code) and sorted to get Top 10.
- Visualized (1) total medals and (2) stacked medal breakdown.

Notes/Caveats:

- Country codes can change or split/merge historically (e.g., URS, FRG).
- High totals reflect *participation scale* and *number of events* as well as performance.

In [8]:
```
# --- Top Countries by Total Medals ---

# 1) Total medals by NOC
top_noc = (
    medal_data.groupby('NOC')
    .size()
    .reset_index(name='Medals')
    .sort_values('Medals', ascending=False)
    .head(10)
)

# Bar chart: total medals
plt.figure(figsize=(10,6))
sns.barplot(data=top_noc, x='Medals', y='NOC')
plt.title('Top 10 NOCs by Total Medals')
plt.xlabel('Total Medals')
plt.ylabel('NOC')
plt.tight_layout()
plt.show()

# 2) Stacked breakdown (Gold/Silver/Bronze) for those Top 10 NOCs
top_noc_codes = top_noc['NOC'].tolist()
breakdown = (
    medal_data[medal_data['NOC'].isin(top_noc_codes)]
    .pivot_table(index='NOC', columns='Medal', values='ID', aggfunc='count', fill_value=0)
    .reindex(top_noc_codes) # keep same order as totals
)

# Ensure consistent medal order
for col in ['Gold','Silver','Bronze']:
    if col not in breakdown.columns:
        breakdown[col] = 0
breakdown = breakdown[['Gold','Silver','Bronze']]

# Stacked bar
ax = breakdown.plot(kind='barh', stacked=True, figsize=(10,6))
plt.title('Top 10 NOCs — Medal Breakdown')
plt.xlabel('Count')
plt.ylabel('NOC')
plt.tight_layout()
plt.show()
```





🏃 Step 8: Top Sports by Total Medals

Goal: Discover which sports have historically produced the most medals.

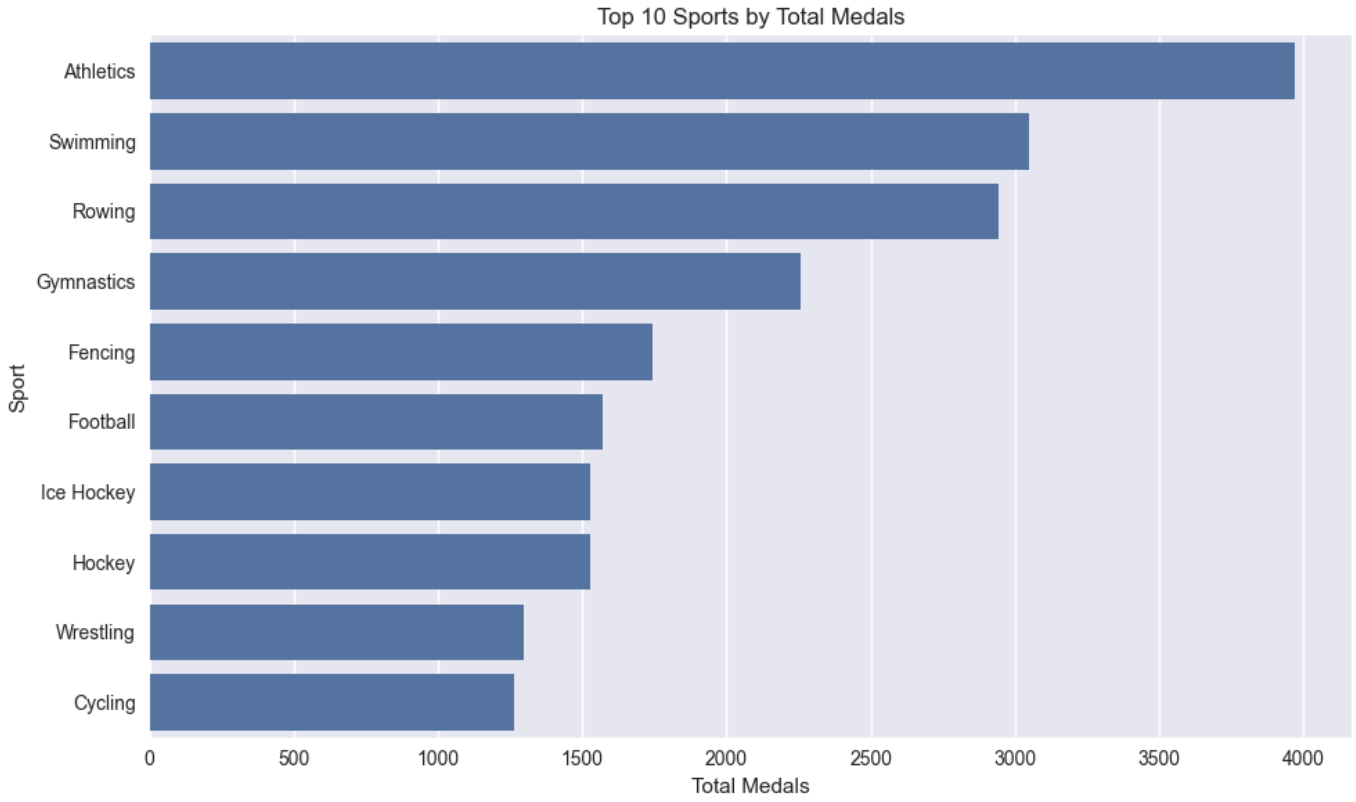
Method:

- Counted medals per `Sport` in `medal_data`.
- Sorted and displayed Top 10.
- This highlights where competition and event density are highest.

Note: Sports with many events (e.g., Athletics, Swimming) naturally accumulate more medals.

In [9]:
```
# --- Top Sports by Total Medals (fixed) ---
```

```
top_sports = (
    medal_data['Sport']
    .value_counts()
    .head(10)
    .reset_index()
)

# Rename columns properly
top_sports.columns = ['Sport', 'Medals']

# Plot
plt.figure(figsize=(10,6))
sns.barplot(data=top_sports, x='Medals', y='Sport')
plt.title('Top 10 Sports by Total Medals')
plt.xlabel('Total Medals')
plt.ylabel('Sport')
plt.tight_layout()
plt.show()
```



In [10]:
```
print(top_sports.head())
print(top_sports.columns)
```

```
        Sport  Medals
0   Athletics    3969
1    Swimming    3048
2      Rowing    2945
3  Gymnastics    2256
4     Fencing    1743
Index(['Sport', 'Medals'], dtype='object')
```

🚹🚺 Step 9: Gender Participation Over Time

Goal: See how athlete participation by gender has evolved over Olympic history.

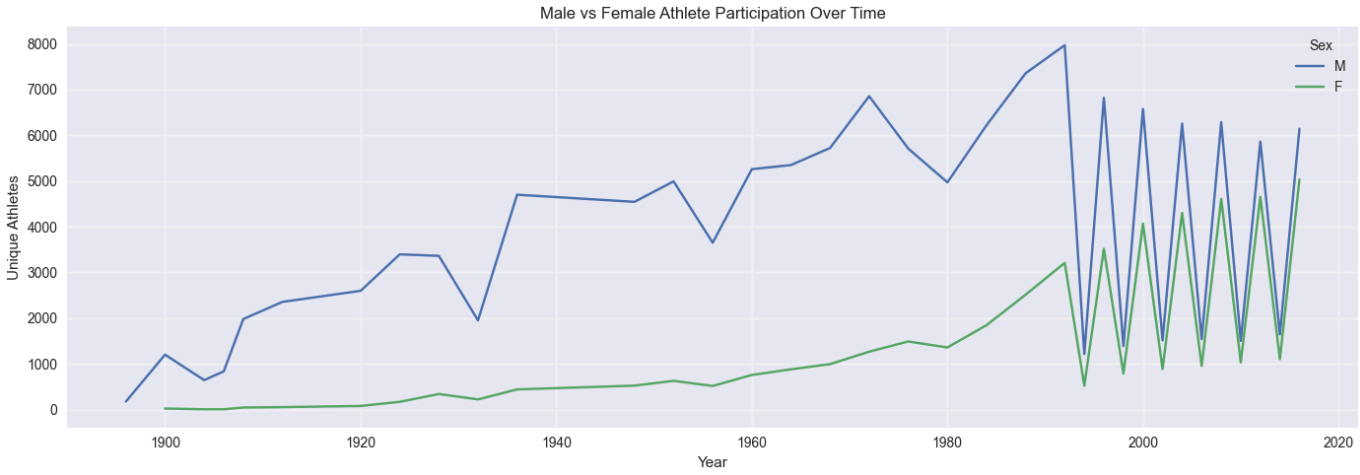
Method:

- Counted **unique athletes** per `Year × Sex`.
- Plotted lines for Male vs Female participation.

Insight to look for:

- Female participation typically rises significantly in late 20th century onward.
- Dips can coincide with global events or canceled editions.

In [11]:
```
# --- Gender Participation Over Time ---

gender_trend = (
    df.groupby(['Year', 'Sex'])['ID']
    .nunique()
    .reset_index(name='Unique_Athletes')
    .sort_values('Year')
)

plt.figure(figsize=(14,5))
sns.lineplot(data=gender_trend, x='Year', y='Unique_Athletes', hue='Sex')
plt.title('Male vs Female Athlete Participation Over Time')
plt.xlabel('Year')
plt.ylabel('Unique Athletes')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```



👴 Step 10: Age Distribution of Olympic Athletes

Goal: Understand the distribution of athlete ages in the Olympics, and how it varies across medalists and non-medalists.

Method:

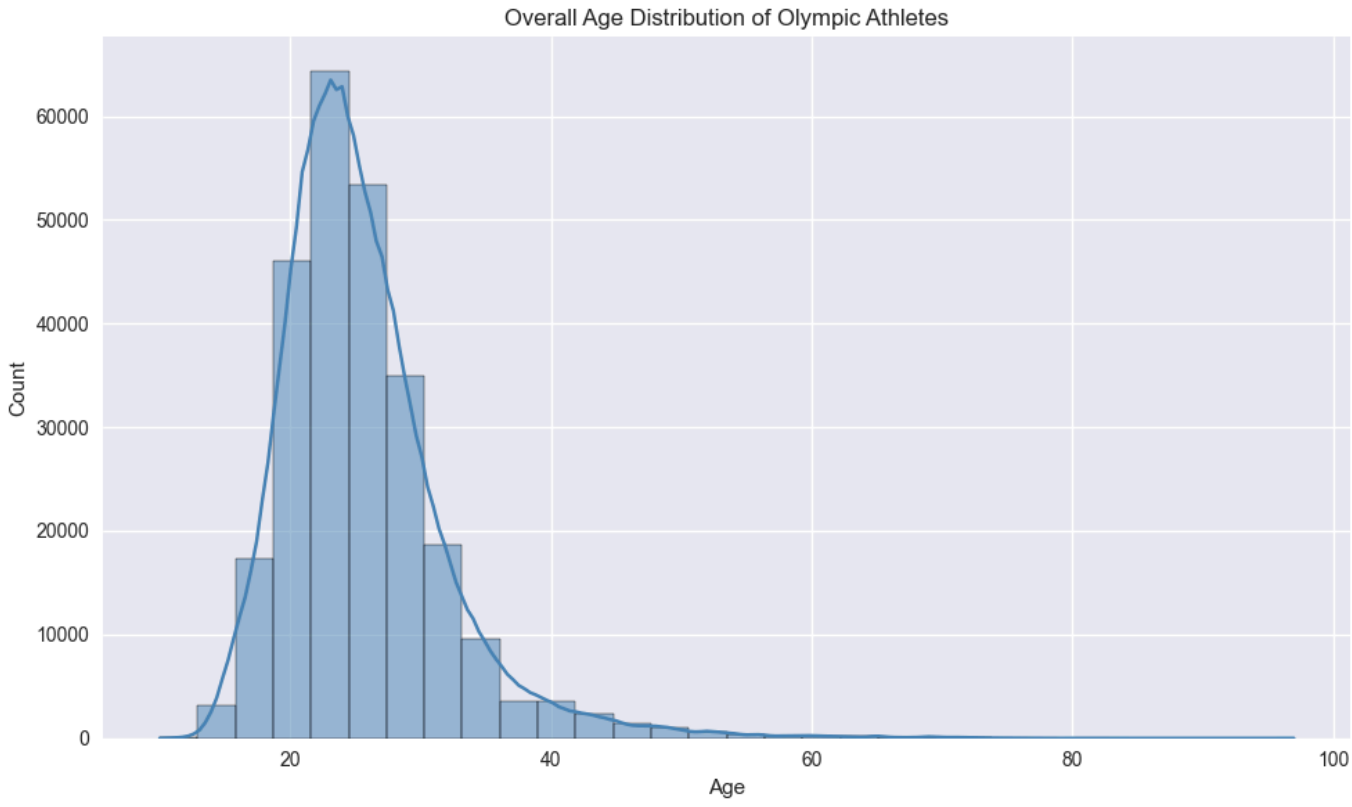
- Used `age_series` (non-null ages).
- Visualized the **overall age distribution** with histogram + KDE curve.
- Compared **medal winners vs. all athletes** to see if medalists cluster around specific ages.
- Optional: Boxplot to summarize distribution and detect outliers.

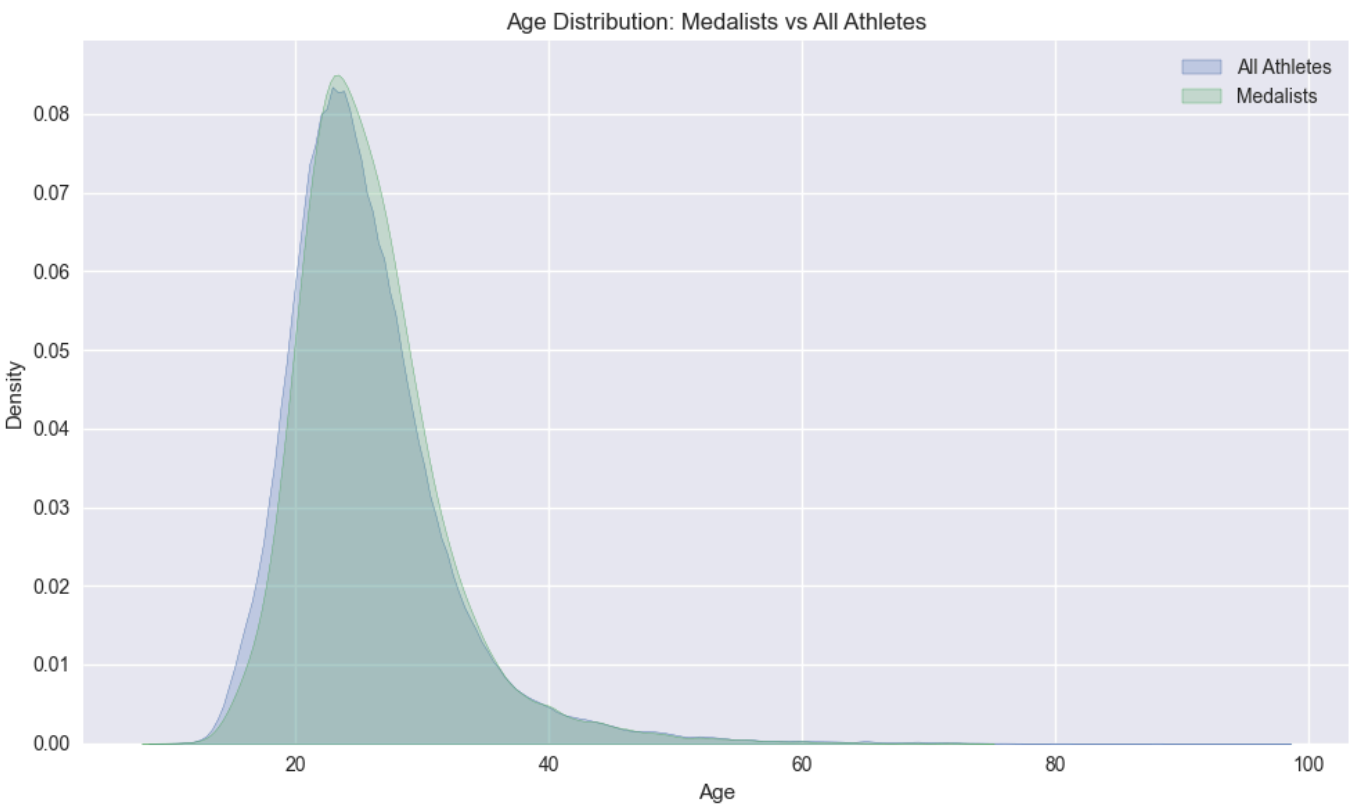
Why this matters:

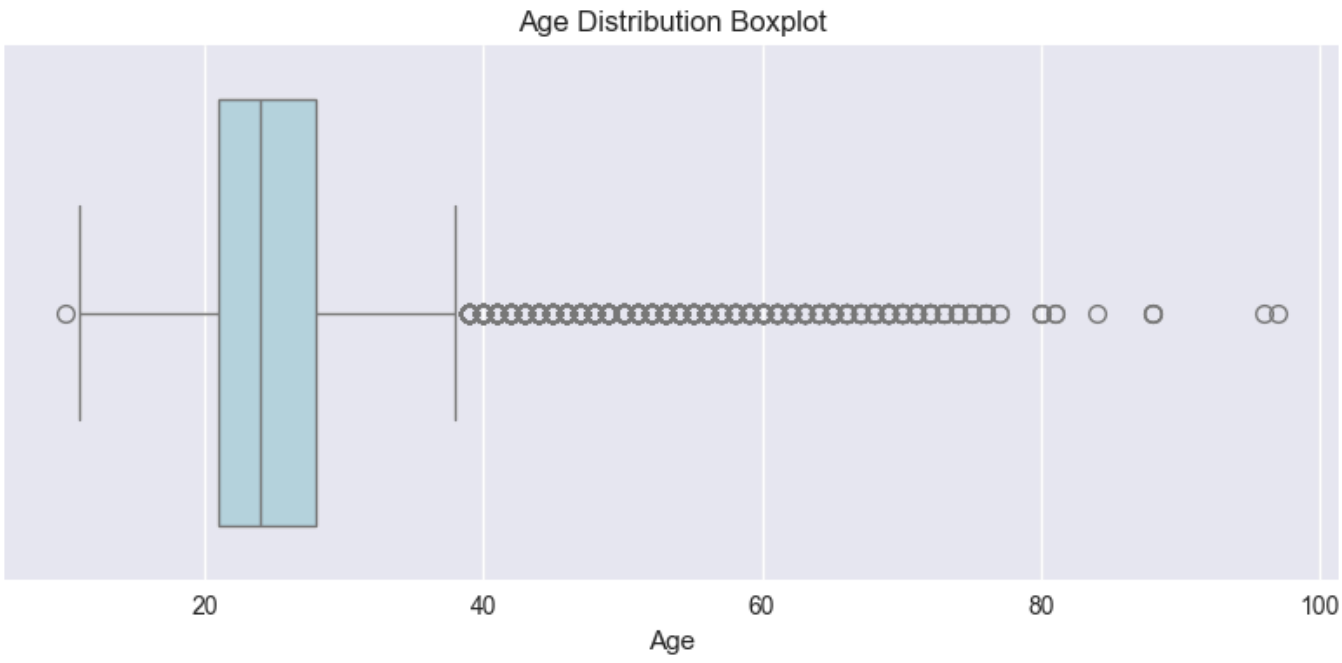
- Highlights typical age ranges of Olympic athletes.
- Can reveal differences in peak performance age by medal status.
- Supports deeper insights for training, scouting, or performance prediction.

In [12]:
```
# --- Age Distribution Analysis ---

# 1) Overall Age Distribution
plt.figure(figsize=(10,6))
sns.histplot(age_series, bins=30, kde=True, color='steelblue')
plt.title('Overall Age Distribution of Olympic Athletes')
```

```
plt.xlabel('Age')
plt.ylabel('Count')
plt.tight_layout()
plt.show()

# 2) Compare Medal Winners vs All Athletes
medal_age = medal_data['Age'].dropna()

plt.figure(figsize=(10,6))
# Replace your two kdeplot lines with these:
sns.kdeplot(x=age_series, label='All Athletes', fill=True)
sns.kdeplot(x=medal_age,  label='Medalists',   fill=True)
plt.title('Age Distribution: Medalists vs All Athletes')
plt.xlabel('Age')
plt.ylabel('Density')
plt.legend()
plt.tight_layout()
plt.show()

# 3) Boxplot for quick summary and outliers
plt.figure(figsize=(8,4))
sns.boxplot(x=age_series, color='lightblue')
plt.title('Age Distribution Boxplot')
plt.xlabel('Age')
plt.tight_layout()
plt.show()
```







🏁 Final Insight Summary — Olympics Data Analysis (1896–2016)

This exploratory data analysis provided meaningful insights into 120 years of Olympic history:

- 🏅 **Medal Trends:**
  - Total medal counts have increased steadily over time, reflecting Olympic expansion.
  - Clear disruptions are visible during major global events (e.g., World Wars).
- 🌍 **Top Performing Nations:**
  - Countries like USA, URS, and Germany have dominated overall medal tallies.
  - Medal distribution reveals strong depth in performance, not just participation.
- 🏊 **Top Sports:**
  - Athletics and Swimming are the top medal-generating sports due to their high event counts.

- 🚹🚺 **Gender Participation:**
  - Male participation has been dominant historically, but **female participation has grown sharply** from the late 20th century.
- 👴 **Age Distribution:**
  - Most athletes compete between **20–30 years**.
  - Medalists show a slightly **narrower and more focused age range**, reflecting peak performance years.

📊 Key Takeaway:

This dataset reflects how the Olympics have grown global