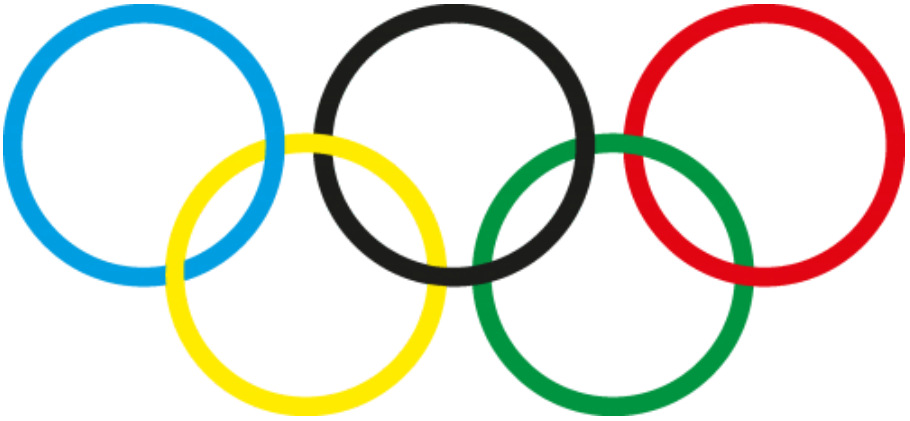
Olympics Games EDA Project



Context:

- The data includes 120 years (1896 to 2016) of Olympic games with information about athletes and medal results.
- This dataset provides an opportunity to ask questions about how the Olympics have evolved over time, including questions about the participation and performance according to genders, different nations, and different sports and events.
- Check out the original source if you are interested in using this data for other purposes (https://www.kaggle.com/heesoo37/120-years-of-olympic-history-athletes-and-results)

Dataset Description:

Each row corresponds to an individual athlete competing in an individual Olympic event.

The columns are:

- **ID**: Unique number for each athlete
- **Name**: Athlete's name
- **Sex**: M or F
- **Age**: Integer
- **Height**: In centimeters
- **Weight**: In kilograms
- **Team**: Team name
- **NOC**: National Olympic Committee 3-letter code
- **Games**: Year and season
- **Year**: Integer
- **Season**: Summer or Winter
- **City**: Host city
- **Sport**: Sport
- **Event**: Event
- **Medal**: Gold, Silver, Bronze, or NA

Objective:

- Examine/clean the dataset
- Explore distributions of single numerical and categorical features via statistics and plots
- Explore relationships of multiple features via statistics and plots

Importing the libraries `Pandas` and `Seaborn`

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

Importing the dataset

In [2]:
```
olympics = pd.read_csv('athlete_events.csv')
```

Getting Basic Information about the Dataset

In [3]:
```
olympics.head()
```

Out[3]:

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Spo
0	1	A Dijiang	M	24.0	180.0	80.0	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketb
1	2	A Lamusi	M	23.0	170.0	60.0	China	CHN	2012 Summer	2012	Summer	London	Ju
2	3	Gunnar Nielsen Aaby	M	24.0	NaN	NaN	Denmark	DEN	1920 Summer	1920	Summer	Antwerpen	Footb
3	4	Edgar Lindenau Aabye	M	34.0	NaN	NaN	Denmark/Sweden	DEN	1900 Summer	1900	Summer	Paris	Tug-O W
4	5	Christine Jacoba Aaftink	F	21.0	185.0	82.0	Netherlands	NED	1988 Winter	1988	Winter	Calgary	Spe Skati

In [4]:
```
olympics.info()
```

```
<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
```

In [5]:
```
olympics.describe()
```

Out[5]:

	ID	Age	Height	Weight	Year
count	271116.000000	261642.000000	210945.000000	208241.000000	271116.000000
mean	68248.954396	25.556898	175.338970	70.702393	1978.378480
std	39022.286345	6.393561	10.518462	14.348020	29.877632
min	1.000000	10.000000	127.000000	25.000000	1896.000000
25%	34643.000000	21.000000	168.000000	60.000000	1960.000000
50%	68205.000000	24.000000	175.000000	70.000000	1988.000000
75%	102097.250000	28.000000	183.000000	79.000000	2002.000000
max	135571.000000	97.000000	226.000000	214.000000	2016.000000

Imputing the missing values in the dataset

Using `IterativeImputer` in `sklearn` to impute based on columns `Year`, `Age`, `Height`, `Weight`

Importing libraries

In [6]:
```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
```

Building a list of columns that will be used for imputation, which are `Year`, `Age`, `Height`, `Weight`

In [7]:
```
impute_cols = olympics[['Year', 'Age', 'Height', 'Weight']].columns
impute_cols
```

Out[7]:
```
Index(['Year', 'Age', 'Height', 'Weight'], dtype='object')
```

Creating an `IterativeImputer` object

In [8]:
```
iter_imp = IterativeImputer(min_value=olympics[impute_cols].min(), max_value=olympics[impute_co
```

Applying the imputer to fit and transform the columns

In [9]:
```
imputed_cols = iter_imp.fit_transform(olympics[impute_cols])
```

Assigning the imputed array back to the original DataFrame's columns

In [10]:
```
olympics[impute_cols] = imputed_cols
```

Checking the columns for missing values

In [11]:
```
olympics.info()
```

```
<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     271116 non-null  float64
 4   Height  271116 non-null  float64
 5   Weight  271116 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  float64
```

```
 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(4), int64(1), object(10)
memory usage: 31.0+ MB
```

Filling the missing values in the column `Medal` with string of 'NA'

In [12]:
```
olympics['Medal'] = olympics['Medal'].fillna('NA')
```

Double checking that all columns are imputed

In [13]:
```
olympics.info()
```

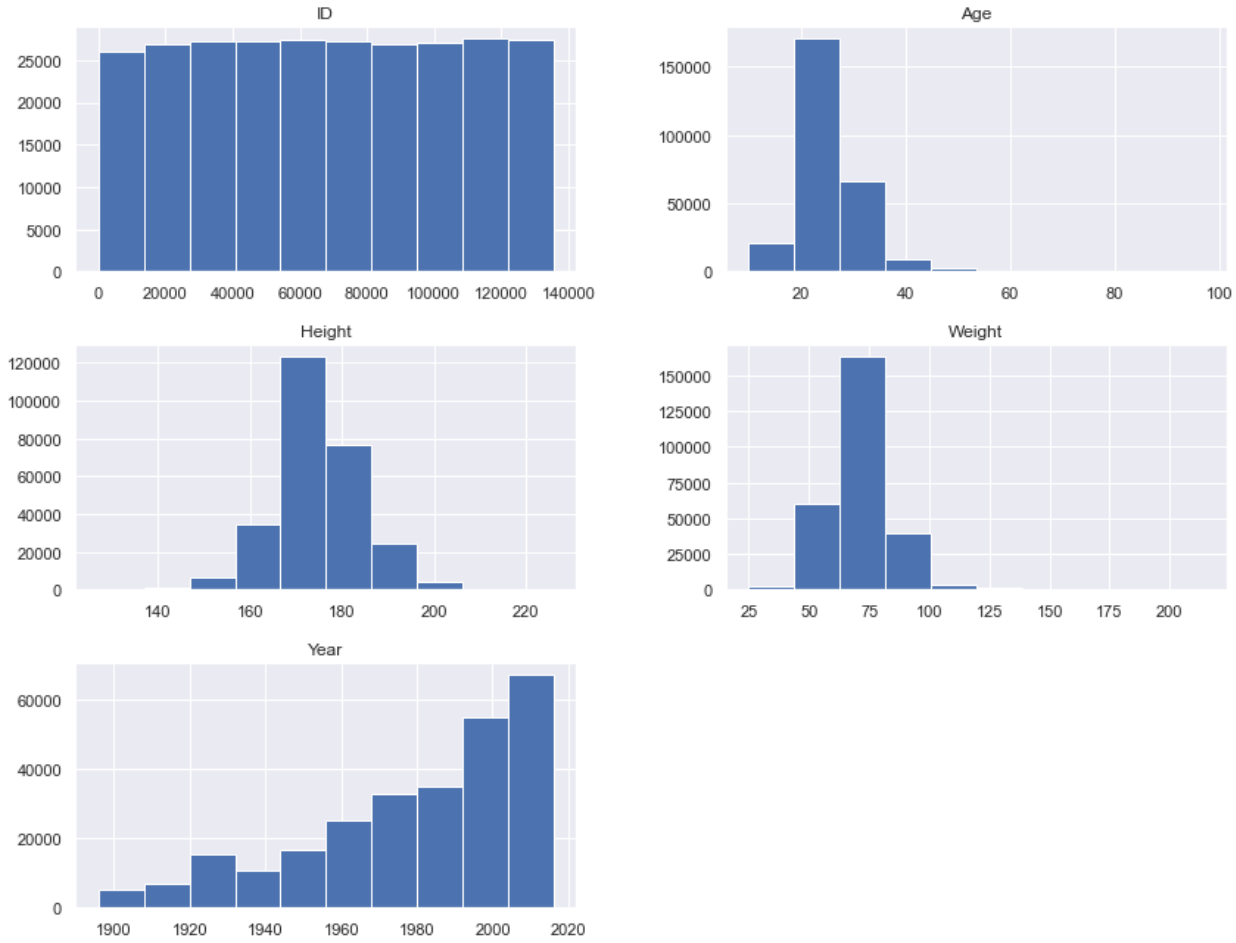
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271116 entries, 0 to 271115
Data columns (total 15 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   ID      271116 non-null  int64  
 1   Name    271116 non-null  object 
 2   Sex     271116 non-null  object 
 3   Age     271116 non-null  float64
 4   Height  271116 non-null  float64
 5   Weight  271116 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  float64
 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   271116 non-null  object 
dtypes: float64(4), int64(1), object(10)
memory usage: 31.0+ MB
```

In [14]:
```
olympics.isnull().sum()
```

Out[14]:
```
ID        0
Name      0
Sex       0
Age       0
Height    0
Weight    0
Team      0
NOC       0
Games     0
Year      0
Season    0
City      0
Sport     0
Event     0
Medal     0
dtype: int64
```

1. Ploting the histograms of the numerical columns using `Pandas`

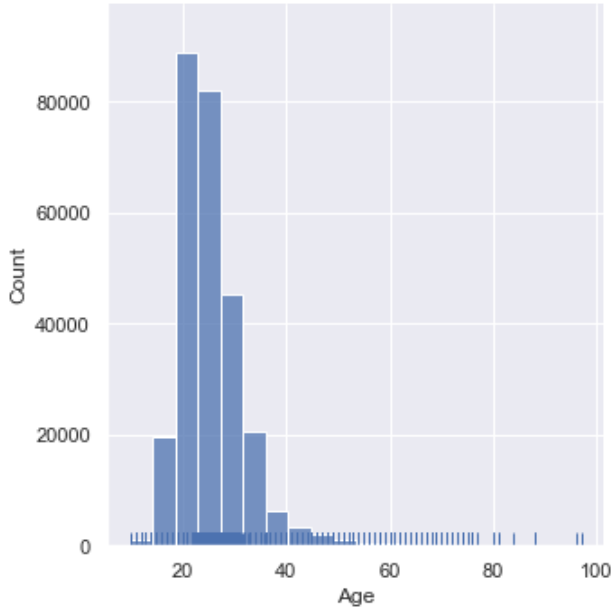
In [15]:
```
olympics.hist(figsize=(14,11))
plt.show()
```



2. Ploting the histogram with a rug plot of the column `Age` using `Seaborn`

In [16]:
```
sns.displot(data=olympics, x='Age', bins=20, rug=True)
```

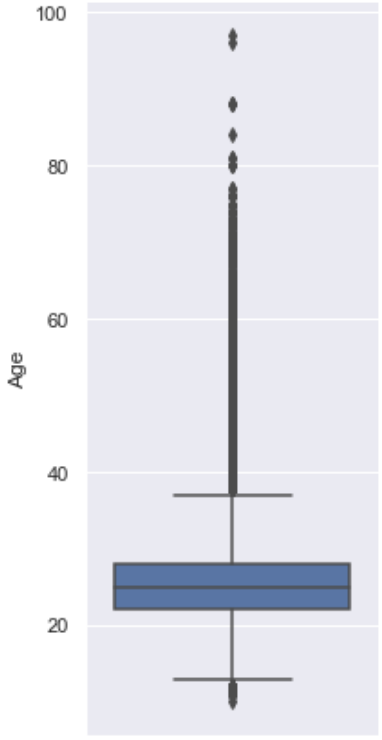
Out[16]:
```
<seaborn.axisgrid.FacetGrid at 0x2156f3afe48>
```



3. Ploting the boxplot of the column `Age` using `Seaborn`

In [17]:
```
sns.catplot(data=olympics, y='Age', kind='box', height=6, aspect=0.5)
```

Out[17]: <seaborn.axisgrid.FacetGrid at 0x2157a1dc208>



4. Calculate the first quartile, third quartile, and IQR of the column `Age`

In [18]:
```
Q1 = olympics['Age'].quantile(0.25)
Q3 = olympics['Age'].quantile(0.75)
IQR = Q3 - Q1
```

In [19]:
```
print(Q1)
print(Q3)
print(IQR)
```

```
22.0
28.0
6.0
```

5. Print out the lower and upper thresholds for outliers based on IQR for the column `Age`

In [20]:
```
print(f'Low age outlier threshold: {Q1 - 1.5*IQR}')
print(f'High age outlier threshold: {Q3 + 1.5*IQR}')
```

```
Low age outlier threshold: 13.0
High age outlier threshold: 37.0
```

Q1. What are the `Sport` for the athletes of really young age

In [21]:
```
olympics[olympics['Age'] < (Q1 - 1.5*IQR)]['Sport'].value_counts()
```

Out[21]:
```
Swimming          25
Figure Skating    15
Gymnastics         5
Rowing             5
Athletics          2
Diving             1
Name: Sport, dtype: int64
```

Q2. What are the `Sport` for the athletes of older age

In [22]:
```
olympics[olympics['Age'] < (Q1 + 1.5*IQR)]['Sport'].value_counts()
```

Out[22]:
```
Athletics        34307
Gymnastics       25039
Swimming         22923
Cycling           9661
Rowing            9471
                 ...
Racquets             8
Jeu De Paume         5
Basque Pelota        2
Roque                1
Aeronautics          1
Name: Sport, Length: 66, dtype: int64
```

Q3. Check for the number of unique values in each column

In [23]:
```
olympics.nunique()
```

Out[23]:
```
ID        135571
Name      134732
Sex            2
Age          785
Height      2475
Weight      3565
Team        1184
NOC          230
Games         51
Year          35
Season         2
City          42
Sport         66
Event        765
Medal          4
dtype: int64
```

Q4. Use the `describe` method to check the non-numerical columns

In [24]:
```
olympics.describe(include=['object'])
```

Out[24]:

	Name	Sex	Team	NOC	Games	Season	City	Sport	Event	Medal
count	271116	271116	271116	271116	271116	271116	271116	271116	271116	271116
unique	134732	2	1184	230	51	2	42	66	765	4
top	Robert Tait McKenzie	M	United States	USA	2000 Summer	Summer	London	Athletics	Football Men's Football	NA
freq	58	196594	17847	18853	13821	222552	22426	38624	5733	231333

Q5. Check the first record within the dataset for each Olympic `Sport`

In [25]:
```
olympics = olympics.sort_values('Year')
```

In [26]:
```
olympics
```

Out[26]:

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City
214333	107607	Fritz Richard Gustav Schuft	M	19.0	172.176206	67.355351	Germany	GER	1896 Summer	1896.0	Summer	Athina

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City
244717	122526	Pierre Alexandre Tuffri	M	19.0	172.176206	67.355351	France	FRA	1896 Summer	1896.0	Summer	Athina
244716	122526	Pierre Alexandre Tuffri	M	19.0	172.176206	67.355351	France	FRA	1896 Summer	1896.0	Summer	Athina
23912	12563	Conrad Helmut Fritz Bcker	M	25.0	173.609502	70.511914	Germany	GER	1896 Summer	1896.0	Summer	Athina
23913	12563	Conrad Helmut Fritz Bcker	M	25.0	173.609502	70.511914	Germany	GER	1896 Summer	1896.0	Summer	Athina
...	...	...	...	...	...	...	...	...	...	...	...	...
142355	71419	Luis Fernando Lpez Erazo	M	37.0	166.000000	60.000000	Colombia	COL	2016 Summer	2016.0	Summer	Rio de Janeiro
47729	24610	Enrico D'Aniello	M	20.0	152.000000	53.000000	Italy	ITA	2016 Summer	2016.0	Summer	Rio de Janeiro
47728	24609	Sabrina D'Angelo	F	23.0	173.000000	71.000000	Canada	CAN	2016 Summer	2016.0	Summer	Rio de Janeiro
47746	24621	Andrea Mitchell D'Arrigo	M	21.0	194.000000	85.000000	Italy	ITA	2016 Summer	2016.0	Summer	Rio de Janeiro
236646	118650	Blair Tarrant	M	26.0	185.000000	83.000000	New Zealand	NZL	2016 Summer	2016.0	Summer	Rio de Janeiro

271116 rows × 15 columns

In [27]:
```
olympics.groupby('Sport').first()
```

Out[27]:

Sport	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Ye
Aeronautics	107506	Hermann Schreiber	M	26.000000	174.589865	71.098729	Switzerland	SUI	1936 Summer	1936
Alpine Skiing	32818	Reat Erce	M	17.000000	172.439922	66.363886	Turkey	TUR	1936 Winter	1936
Alpinism	74134	George Herbert Leigh Mallory	M	37.000000	176.995129	76.867544	Great Britain	GBR	1924 Winter	1924
Archery	67722	Lecomte	M	27.618563	174.310796	71.897322	France	FRA	1900 Summer	1900

Sport	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Ye
Art Competitions	48741	Konrad Hippenmeier	M	31.000000	175.339389	73.692765	Switzerland	SUI	1912 Summer	1912
...	...	...	...	...	...	...	...	...	...	
Tug-Of-War	86368	August Nilsson	M	27.000000	174.161415	71.570173	Denmark/Sweden	SWE	1900 Summer	1900
Volleyball	14306	Georgi Spasov Boyadzhiev	M	21.000000	177.000000	76.000000	Bulgaria	BUL	1964 Summer	1964
Water Polo	74733	Auguste Jean Baptiste Louis Joseph Marc	M	19.000000	172.250355	67.361423	Pupilles de Neptune de Lille #2-1	FRA	1900 Summer	1900
Weightlifting	54456	Alexander Viggo Jensen	M	21.000000	172.653972	68.407539	Denmark	DEN	1896 Summer	1896
Wrestling	122329	Georgios Tsitas	M	27.721559	174.261331	71.945520	Greece	GRE	1896 Summer	1896

66 rows × 14 columns

Q6. What are the average `Age`, `Height`, `Weight` of female versus male Olympic athletes

In [28]:
```
olympics.groupby('Sex')[['Age', 'Height', 'Weight']].mean()
```

Out[28]:

Sex	Age	Height	Weight
F	23.748234	168.476034	60.960650
M	26.302496	177.859161	74.740574

Q7. What are the minimum, average, maximum `Age`, `Height`, `Weight` of athletes in different `Year`

In [29]:
```
olympics.groupby('Year')[['Age', 'Height', 'Weight']].agg(['min', 'mean', 'max'])
```

Out[29]:

	Age			Height			Weight		
	min	mean	max	min	mean	max	min	mean	max
Year									
1896.0	10.0	25.364356	40.0	154.0	173.569607	188.000000	45.000000	70.982411	106.0
1900.0	13.0	28.454101	71.0	153.0	174.664627	191.000000	51.000000	72.502816	102.0
1904.0	14.0	26.867754	71.0	155.0	174.435046	195.000000	43.000000	71.715326	115.0

	Age			Height			Weight		
	min	mean	max	min	mean	max	min	mean	max
Year									
1906.0	13.0	27.272610	54.0	165.0	174.946975	196.000000	52.000000	72.309427	114.0
1908.0	14.0	27.061116	61.0	157.0	175.059627	201.000000	51.000000	72.484058	115.0
1912.0	13.0	27.528582	67.0	157.0	175.149494	200.000000	49.000000	72.354296	125.0
1920.0	13.0	28.861062	72.0	142.0	175.220153	218.770915	33.074491	72.791440	146.0
1924.0	11.0	28.101622	81.0	142.0	174.980222	218.578230	34.356723	72.217506	146.0
1928.0	11.0	28.728275	97.0	147.0	175.282553	211.000000	41.000000	72.568654	125.0
1932.0	11.0	32.002855	96.0	147.0	175.826347	200.000000	40.090648	73.867313	110.0
1936.0	11.0	27.506104	74.0	147.0	175.155241	205.000000	37.000000	71.960859	138.0
1948.0	12.0	28.407495	84.0	140.0	175.567741	213.000000	47.000000	72.408041	125.0
1952.0	12.0	26.165217	65.0	150.0	174.812820	213.000000	42.000000	71.097510	145.0
1956.0	12.0	25.949715	67.0	137.0	174.607226	218.000000	28.000000	71.041599	141.0
1960.0	11.0	25.187182	65.0	137.0	173.316035	218.000000	36.000000	69.480564	141.0
1964.0	12.0	24.948374	60.0	137.0	173.546073	218.000000	38.000000	69.759825	163.0
1968.0	11.0	24.263374	68.0	127.0	173.959439	216.000000	34.000000	69.609517	163.0
1972.0	12.0	24.318900	69.0	130.0	174.562405	223.000000	38.000000	70.003339	182.0
1976.0	12.0	23.850452	70.0	136.0	174.865117	220.000000	30.000000	70.000335	163.0
1980.0	13.0	23.734613	70.0	131.0	175.508742	220.000000	25.000000	70.624903	190.0
1984.0	12.0	23.925675	60.0	132.0	175.524970	218.000000	31.000000	70.257754	150.0
1988.0	11.0	24.085104	70.0	127.0	175.698459	223.000000	32.000000	70.443543	161.0
1992.0	11.0	24.319680	62.0	136.0	175.959574	226.000000	30.000000	70.862030	176.5
1994.0	13.0	24.422594	46.0	148.0	175.158296	200.000000	40.000000	70.972262	113.0
1996.0	12.0	24.915179	63.0	136.0	175.836214	223.000000	30.000000	70.818289	176.5
1998.0	14.0	25.163160	50.0	142.0	174.589636	200.000000	32.000000	70.898616	123.0
2000.0	13.0	25.422476	63.0	136.0	176.085713	226.000000	28.000000	71.106805	180.0
2002.0	15.0	25.916281	48.0	149.0	174.709888	201.000000	42.000000	71.164405	123.0
2004.0	13.0	25.639515	57.0	139.0	175.971393	226.000000	30.000000	71.284135	198.0
2006.0	14.0	25.959151	52.0	147.0	174.628393	206.000000	38.000000	70.512440	127.0
2008.0	12.0	25.733541	67.0	137.0	176.209780	226.000000	28.000000	71.375322	214.0
2010.0	15.0	26.124262	51.0	149.0	174.918873	206.000000	38.000000	70.733917	116.0
2012.0	13.0	25.961378	71.0	140.0	176.247158	221.000000	30.071474	71.173434	214.0
2014.0	15.0	25.987324	55.0	146.0	174.818773	206.000000	38.066881	70.572661	116.0
2016.0	13.0	26.207919	62.0	133.0	176.024469	218.000000	30.000000	70.966084	170.0

Q8. What are the minimum, average, median, maximum `Age` of athletes for different `Season` and `Sex` combinations

In [30]:
```
olympics.groupby(['Season', 'Sex'])['Age'].agg(['min', 'median', 'mean', 'max'])
```

Out[30]:

	min	median	mean	max

Season	Sex	min	median	mean	max
Season	**Sex**				
Summer	**F**	11.0	23.0	23.680377	74.0
	M	10.0	25.0	26.464649	97.0
Winter	**F**	11.0	24.0	24.015736	48.0
	M	12.0	25.0	25.512635	58.0

Q9. What are the average `Age` of athletes, and numbers of unique `Team`, `Sport`, `Event`, for different `Season` and `Sex` combinations

In [31]:
```
olympics.groupby(['Season', 'Sex'])[['Age', 'Team', 'Sport', 'Event']].agg({'Age':'mean', 'Team
```

Out[31]:

Season	Sex	Age	Team	Sport	Event
Summer	**F**	23.680377	352	40	214
	M	26.464649	1118	49	491
Winter	**F**	24.015736	144	14	57
	M	25.512635	214	17	67

Q10. What are the average `Age`, `Height`, `Weight` of athletes, for different `Medal`, `Season`, `Sex` combinations

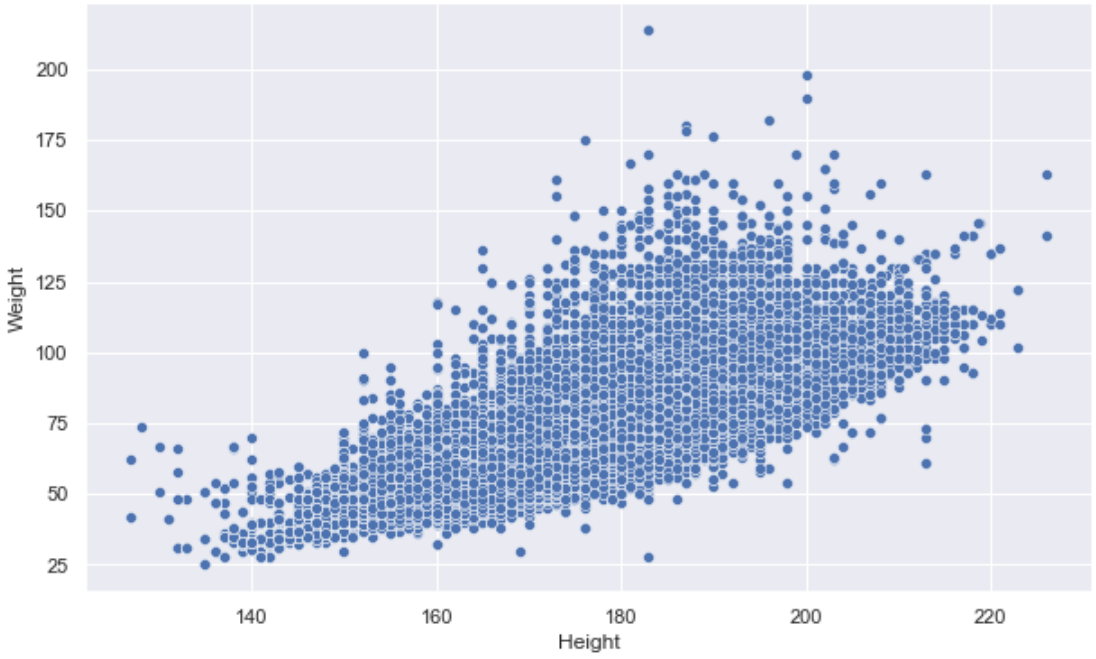
In [32]:
```
olympics.groupby(['Medal', 'Season', 'Sex'])[['Age', 'Height', 'Weight']].mean()
```

Out[32]:

Medal	Season	Sex	Age	Height	Weight
Bronze	**Summer**	**F**	24.637527	171.110734	63.903325
		M	26.382673	179.283138	76.834194
	Winter	**F**	25.115578	167.339043	61.132946
		M	26.387043	178.831087	77.296962
Gold	**Summer**	**F**	24.215093	171.594933	64.252235
		M	26.501307	179.803255	77.426878
	Winter	**F**	25.202636	167.563734	62.006768
		M	26.606436	179.569202	78.047604
NA	**Summer**	**F**	23.547521	168.283937	60.468257
		M	26.454123	177.557129	74.215742
	Winter	**F**	23.855915	167.281733	60.539905
		M	25.386328	177.741840	74.861337
Silver	**Summer**	**F**	24.296089	171.310947	63.903671
		M	26.690500	179.391244	77.017761
	Winter	**F**	25.240527	167.916040	61.921625
		M	26.430566	179.030573	77.429293

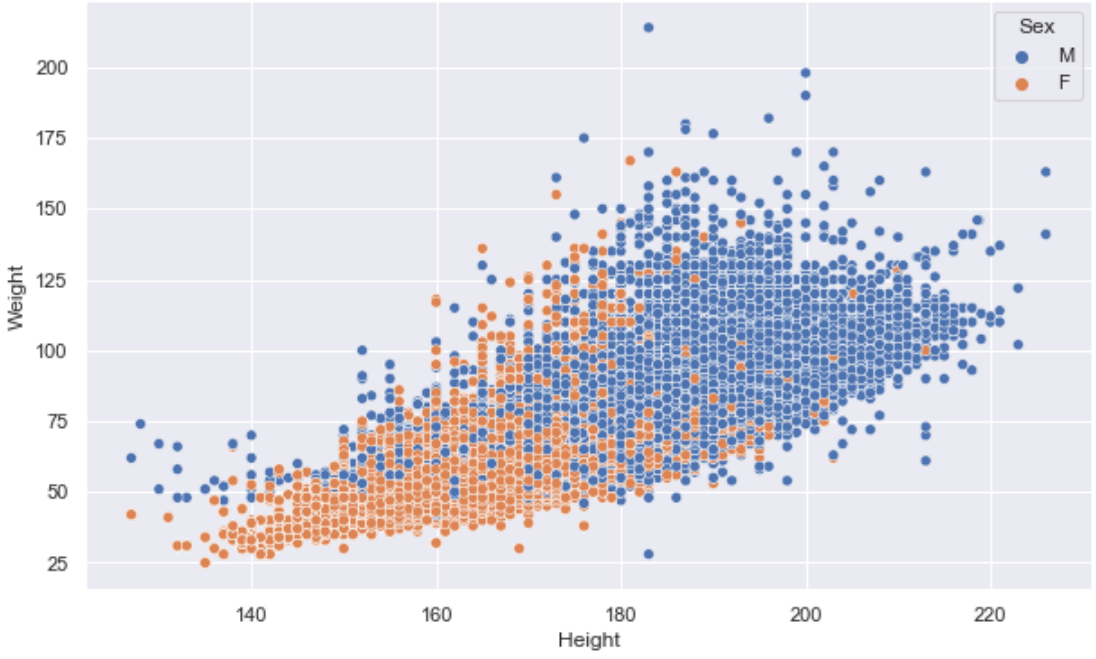
Q11. Plot the scatterplot of `Height` and `Weight`

In [33]:
```
plt.figure(figsize=(10,6))
sns.scatterplot(data=olympics, x='Height', y='Weight')
plt.show()
```



Q12. Plot the scatterplot of `Height` and `Weight`, using different colors and styles of dots for different `Sex`

In [34]:
```
plt.figure(figsize=(10,6))
sns.scatterplot(data=olympics, x='Height', y='Weight', hue='Sex')
plt.show()
```



Q13. Plot the pairwise relationships of `Age`, `Height`, `Weight`

In [35]:
```
sns.pairplot(data=olympics[['Age', 'Height', 'Weight']], height=4, aspect=1.5)
plt.show()
```

Q14. Plot the pairwise relationships of `Age`, `Height`, `Weight`, with different colors for `Sex`

In [36]:
```
sns.pairplot(data=olympics[['Age', 'Height', 'Weight', 'Sex']], hue='Sex', height=4, aspect=1.5
plt.show()
```

Q15. Print out the correlation matrix of `Age`, `Height`, `Weight`

In [37]:
```
olympics[['Age', 'Height', 'Weight']].corr()
```

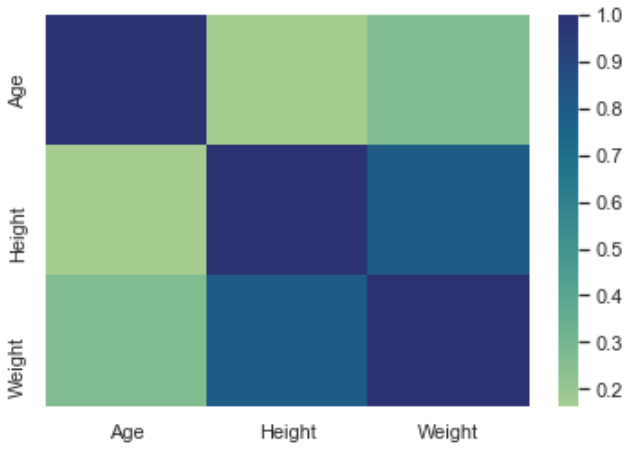
Out[37]:

	Age	Height	Weight
Age	1.000000	0.161797	0.264409
Height	0.161797	1.000000	0.799258
Weight	0.264409	0.799258	1.000000

Q16. Use heatmap to demonstrate the correlation matrix of `Age`, `Height`, `Weight`, use a colormap (`cmap`) of 'crest'

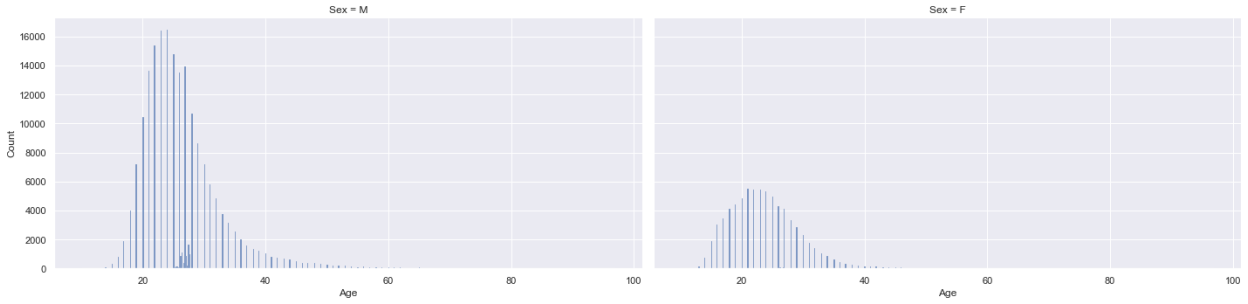
In [38]:
```
sns.heatmap(data=olympics[['Age', 'Height', 'Weight']].corr(), cmap='crest')
```

Out[38]: <AxesSubplot:>



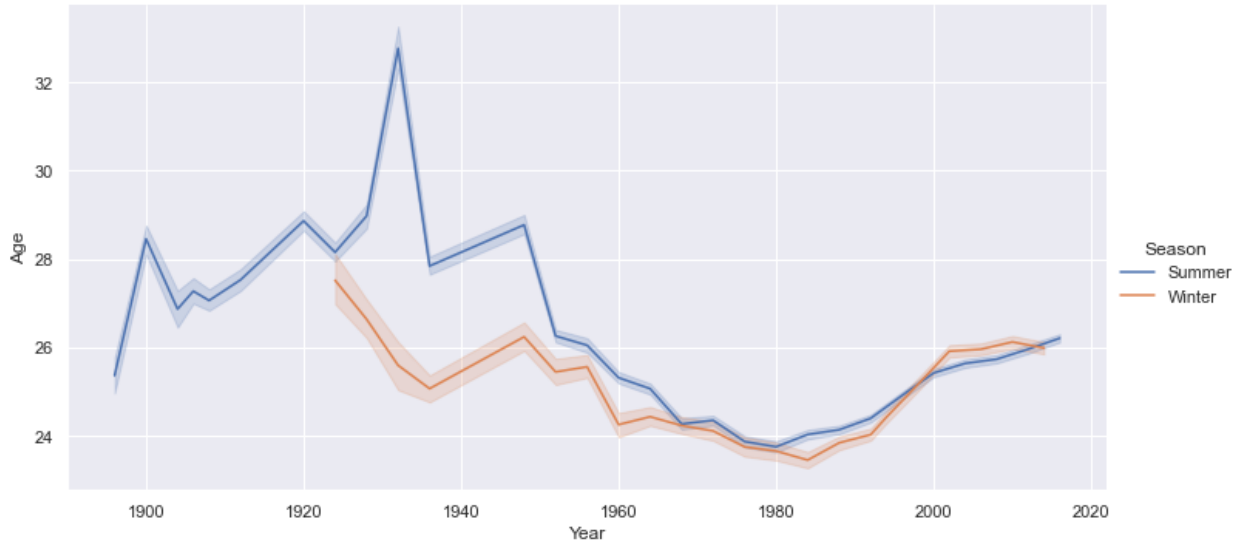
Q17. Plot the histograms of `Age`, on separate plots for different `Sex`

In [39]:
```
sns.displot(data=olympics, x='Age', col='Sex', aspect=2)
plt.show()
```



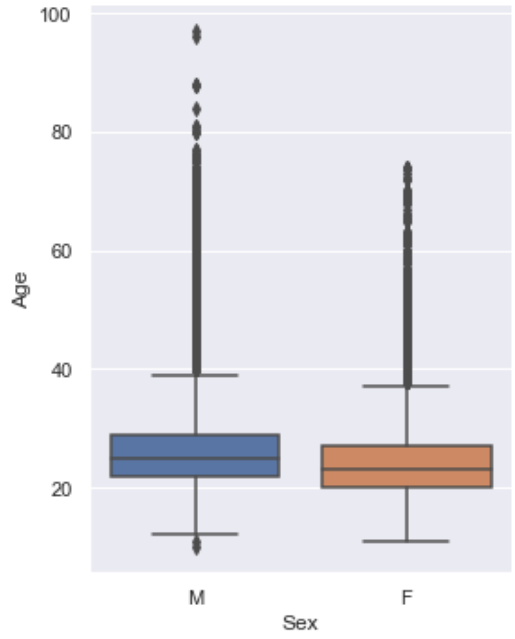
Q18. Look at the changes of average `Age` across `Year` by line charts, with separate lines for different `Season` using different colors

In [40]:
```
sns.relplot(data=olympics, x='Year', y='Age', kind='line', hue='Season', aspect=2)
plt.show()
```



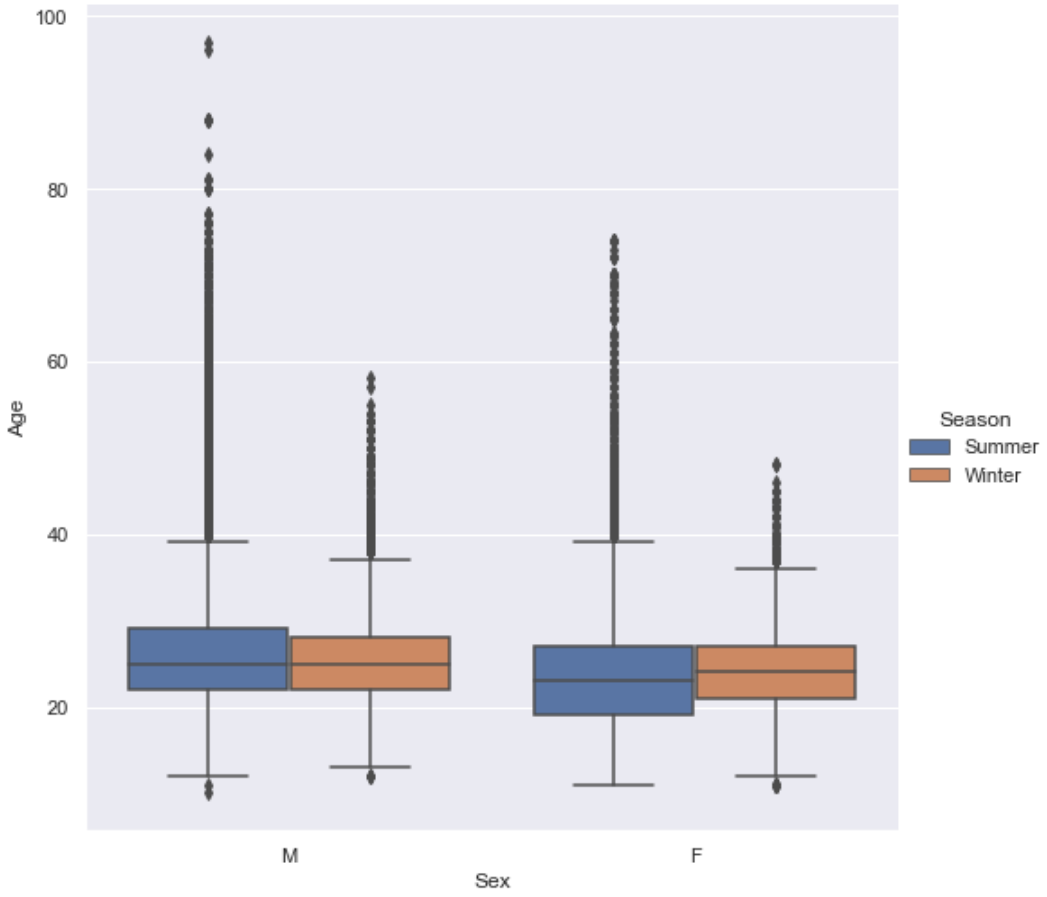
Q19. Look at the distributions of `Age` for different `Sex` using boxplots

In [41]:
```
sns.catplot(data=olympics, y='Age', kind='box', x='Sex', height=5, aspect=0.8)
plt.show()
```



Q20. Look at the distributions of `Age` for different `Sex` using boxplots, with different colors of plots for different `Season`

In [42]:
```
sns.catplot(data=olympics, y='Age', kind='box', x='Sex', hue='Season', height=7, aspect=1)
plt.show()
```



Q21. Use count plots to look at the changes of number of athlete-events across `Year`, for different `Sex` by colors, and different `Season` on separate plots

In [43]:
```
sns.catplot(data=olympics, x='Year', kind='count', hue='Sex', col='Season', col_wrap=1, aspect=
plt.show()
```