CW - Solutions

Q1(a)

Read in the csv file "Olympic_Games.csv" to create a dataframe. Remove the columns named 'ID','Name','Team','Games' and 'City'. Further, remove all the rows which have 'NA' in the 'Medal' column. In the 'Sex' column, convert all of the 'M's to 0 and all of the 'F's to 1. In the 'Season' column, convert all of the 'Summer's to 0 and all of the 'Winter's to 1. In the 'Medal' column, convert all of the 'Gold's to 3, all of the 'Silver's to 2, all of the 'Bronze's to 1. In the 'NOC' column, convert all of the 'FRG's to 'GER' and convert all of the 'GDR's to 'GER'. Finally remove all the rows which have "NOC" equals "URS" or "RUS". (4 marks)

In [1]:
```
import pandas as pd    #importing pandas library and renaming it to pd

#reads csv file specified in the path
DF_Olympics = pd.read_csv("C:/Users/prano/OneDrive/Documents/DataScienceProgCoursework/Olympic_Games.csv")
print(len(DF_Olympics))

#drops the specified columns in the list from the dataframe "DF_Olympics"
DF_Olympics=DF_Olympics.drop(columns=['ID','Name','Team','Games' ,'City'])
print(DF_Olympics.columns)

#drops all the rows with 'NA' in medal column of the dataframe
DF_Olympics=DF_Olympics.dropna(subset=['Medal'])
print(len(DF_Olympics))
DF_Olympics
```

```
271116
Index(['Sex', 'Age', 'Height', 'Weight', 'NOC', 'Year', 'Season', 'Sport',
       'Event', 'Medal'],
      dtype='object')
39783
```

Out[1]:

	Sex	Age	Height	Weight	NOC	Year	Season	Sport	Event	Medal
3	M	34.0	NaN	NaN	DEN	1900	Summer	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	Gold
37	M	30.0	NaN	NaN	FIN	1920	Summer	Swimming	Swimming Men's 200 metres Breaststroke	Bronze
38	M	30.0	NaN	NaN	FIN	1920	Summer	Swimming	Swimming Men's 400 metres Breaststroke	Bronze
40	M	28.0	184.0	85.0	FIN	2014	Winter	Ice Hockey	Ice Hockey Men's Ice Hockey	Bronze
41	M	28.0	175.0	64.0	FIN	1948	Summer	Gymnastics	Gymnastics Men's Individual All-Around	Bronze
...	...	...	...	...	...	...	...	...	...	...
271078	F	25.0	168.0	80.0	URS	1956	Summer	Athletics	Athletics Women's Shot Put	Silver
271080	F	33.0	168.0	80.0	URS	1964	Summer	Athletics	Athletics Women's Shot Put	Bronze
271082	M	28.0	182.0	82.0	POL	1980	Summer	Fencing	Fencing Men's Foil, Team	Bronze
271102	F	19.0	171.0	64.0	RUS	2000	Summer	Athletics	Athletics Women's 4 x 400 metres Relay	Bronze
271103	F	23.0	171.0	64.0	RUS	2004	Summer	Athletics	Athletics Women's 4 x 400 metres Relay	Silver

39783 rows × 10 columns

In [2]:

```
#Replace the column values in the column "Sex" from M to 0, and F to 1
DF_Olympics['Sex'] = DF_Olympics['Sex'].map({'M': 0, 'F': 1})

#Replace the column values in the column "Season" from Summer to 0, and winter to 1
DF_Olympics['Season'] = DF_Olympics['Season'].map({'Summer': 0, 'Winter': 1})

#Replace the column values in the column "Medal" from gold to 3, silver to 2, and bronze to 1
DF_Olympics['Medal'] = DF_Olympics['Medal'].map({'Gold': 3, 'Silver': 2, 'Bronze':1})

#print(DF_Olympics)
DF_Olympics
```

Out[2]:

	Sex	Age	Height	Weight	NOC	Year	Season	Sport	Event	Medal
3	0	34.0	NaN	NaN	DEN	1900	0	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	3
37	0	30.0	NaN	NaN	FIN	1920	0	Swimming	Swimming Men's 200 metres Breaststroke	1
38	0	30.0	NaN	NaN	FIN	1920	0	Swimming	Swimming Men's 400 metres Breaststroke	1
40	0	28.0	184.0	85.0	FIN	2014	1	Ice Hockey	Ice Hockey Men's Ice Hockey	1
41	0	28.0	175.0	64.0	FIN	1948	0	Gymnastics	Gymnastics Men's Individual All-Around	1
...	...	...	...	...	...	...	...	...	...	...
271078	1	25.0	168.0	80.0	URS	1956	0	Athletics	Athletics Women's Shot Put	2
271080	1	33.0	168.0	80.0	URS	1964	0	Athletics	Athletics Women's Shot Put	1
271082	0	28.0	182.0	82.0	POL	1980	0	Fencing	Fencing Men's Foil, Team	1
271102	1	19.0	171.0	64.0	RUS	2000	0	Athletics	Athletics Women's 4 x 400 metres Relay	1
271103	1	23.0	171.0	64.0	RUS	2004	0	Athletics	Athletics Women's 4 x 400 metres Relay	2

39783 rows × 10 columns

In [3]:

```
#values in column 'NOC' replaced from 'FRG' or 'GDR' to 'GER'
DF_Olympics['NOC']=DF_Olympics['NOC'].replace({'FRG': 'GER', 'GDR': 'GER'})


#drops rows where 'NOC' column has values 'URS' or 'RUS'
DF_Olympics = DF_Olympics[~DF_Olympics.NOC.isin(['URS','RUS'])]

#print(len(DF_Olympics))
DF_Olympics
```

Out[3]:

	Sex	Age	Height	Weight	NOC	Year	Season	Sport	Event	Medal
3	0	34.0	NaN	NaN	DEN	1900	0	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	3
37	0	30.0	NaN	NaN	FIN	1920	0	Swimming	Swimming Men's 200 metres Breaststroke	1
38	0	30.0	NaN	NaN	FIN	1920	0	Swimming	Swimming Men's 400 metres Breaststroke	1
40	0	28.0	184.0	85.0	FIN	2014	1	Ice Hockey	Ice Hockey Men's Ice Hockey	1
41	0	28.0	175.0	64.0	FIN	1948	0	Gymnastics	Gymnastics Men's Individual All-Around	1
...	...	...	...	...	...	...	...	...	...	...
271032	1	22.0	181.0	78.0	NED	1996	0	Judo	Judo Women's Middleweight	1
271046	0	21.0	175.0	70.0	POL	1980	0	Athletics	Athletics Men's 4 x 100 metres Relay	2
271048	0	27.0	197.0	93.0	NED	1992	0	Rowing	Rowing Men's Double Sculls	1
271049	0	31.0	197.0	93.0	NED	1996	0	Rowing	Rowing Men's Coxed Eights	3
271082	0	28.0	182.0	82.0	POL	1980	0	Fencing	Fencing Men's Foil, Team	1

36115 rows × 10 columns

Q1(b)

Remove all 'NOC' which appear less than 25 times. (i) Create a histrogram of the number of countries with medals using 10 bins. (ii) Create a histrogram of the number of countries with gold medals using 10 bins. (iii) Create a histrogram of the number of countries with points using 10 bins, where points equals the sum of the number of bronze medals plus twice the number of silver medals but three times the number of gold medals. Comment on these results. (6 marks)

In [4]:

```
#imports container Counter from the collections module which stores a key value pair of the count of element passed in container
from collections import Counter

#Counts the frequency of each "NOC" and stores the output as a counter Dictionary
dict_NOC_Value_counter=Counter(DF_Olympics['NOC'])
print(dict_NOC_Value_counter,'\n')

#Loops over the dictionary counter to remove countries where count is less then 25 from dataframe DF_Olympics
```

```
for key in dict_NOC_Value_counter:
    if(dict_NOC_Value_counter[key]<25):
        DF_Olympics = DF_Olympics[~DF_Olympics.NOC.eq(key)]

print('Rowcount of DF_Olympics:',len(DF_Olympics)) #prints rowcount after dropping the rows with NOC count <25
```

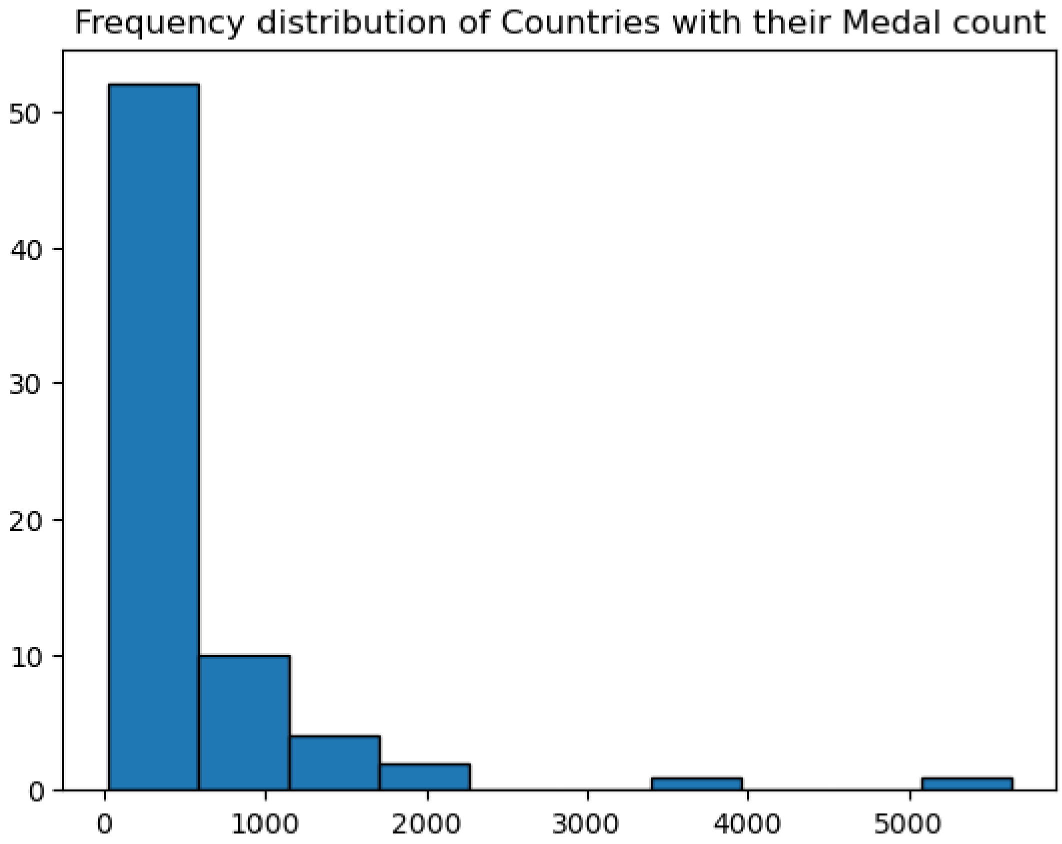
```
Counter({'USA': 5637, 'GER': 3756, 'GBR': 2068, 'FRA': 1777, 'ITA': 1637, 'SWE': 1536, 'CAN': 1352, 'AUS': 1320, 'HUN': 1135, 'N
ED': 1040, 'NOR': 1033, 'CHN': 989, 'JPN': 913, 'FIN': 900, 'SUI': 691, 'ROU': 653, 'KOR': 638, 'DEN': 597, 'POL': 565, 'ESP': 4
89, 'TCH': 488, 'BRA': 475, 'BEL': 468, 'AUT': 450, 'CUB': 409, 'YUG': 390, 'BUL': 342, 'EUN': 279, 'ARG': 274, 'GRE': 255, 'NZ
L': 228, 'UKR': 199, 'IND': 197, 'JAM': 157, 'CRO': 149, 'CZE': 144, 'BLR': 139, 'RSA': 131, 'PAK': 121, 'MEX': 110, 'KEN': 106,
'NGR': 99, 'TUR': 95, 'SRB': 85, 'KAZ': 77, 'IRI': 68, 'PRK': 67, 'SCG': 64, 'URU': 63, 'LTU': 61, 'ETH': 53, 'EST': 50, 'TPE':
49, 'SLO': 48, 'SVK': 47, 'AZE': 44, 'INA': 41, 'POR': 41, 'BAH': 40, 'IRL': 35, 'LAT': 35, 'UZB': 34, 'CHI': 32, 'TTO': 32, 'GE
O': 32, 'THA': 30, 'ANZ': 29, 'COL': 28, 'EGY': 27, 'MGL': 26, 'GHA': 23, 'MAR': 23, 'CMR': 22, 'ZIM': 22, 'ALG': 17, 'ISL': 17,
'PAR': 17, 'ARM': 16, 'MAS': 16, 'VEN': 15, 'PER': 15, 'MNE': 14, 'TUN': 13, 'FIJ': 13, 'BOH': 12, 'PHI': 10, 'PUR': 9, 'ISR':
9, 'SGP': 9, 'LIE': 9, 'LUX': 8, 'MDA': 8, 'UGA': 7, 'HAI': 7, 'DOM': 7, 'KSA': 6, 'QAT': 5, 'IOA': 5, 'WIF': 5, 'TJK': 4, 'LI
B': 4, 'NAM': 4, 'VIE': 4, 'HKG': 4, 'CRC': 4, 'SYR': 3, 'KGZ': 3, 'CIV': 3, 'BRN': 3, 'PAN': 3, 'KUW': 2, 'UAE': 2, 'NIG': 2,
'TAN': 2, 'UAR': 2, 'GRN': 2, 'SRI': 2, 'ZAM': 2, 'MOZ': 2, 'SUR': 2, 'AFG': 2, 'BDI': 2, 'ECU': 2, 'JOR': 1, 'BOT': 1, 'GUY':
1, 'IRQ': 1, 'GUA': 1, 'AHO': 1, 'TOG': 1, 'NEP': 1, 'SEN': 1, 'BER': 1, 'ISV': 1, 'MKD': 1, 'SUD': 1, 'MRI': 1, 'KOS': 1, 'CY
P': 1, 'MON': 1, 'GAB': 1, 'DJI': 1, 'ERI': 1, 'BAR': 1, 'TGA': 1})

Rowcount of DF_Olympics: 35669
```

In [5]:

```
import matplotlib.pyplot as plt #imports pyplot collection from matplotlib library and renaming it to plt

countries_wrt_medal_count=DF_Olympics.groupby('NOC')['Medal'].count() #creates a series with unique NOCs and their medal counts
plt.hist(countries_wrt_medal_count,bins=10,edgecolor='black') #plots a histogram with the frequency of countries with 10 bins
plt.title('Frequency distribution of Countries with their Medal count') #gives title to the plot
countries_wrt_medal_count
```

Out[5]:

```
NOC
ANZ      29
ARG     274
AUS    1320
AUT     450
AZE      44
       ...
UKR     199
URU      63
USA    5637
UZB      34
YUG     390
Name: Medal, Length: 70, dtype: int64
```



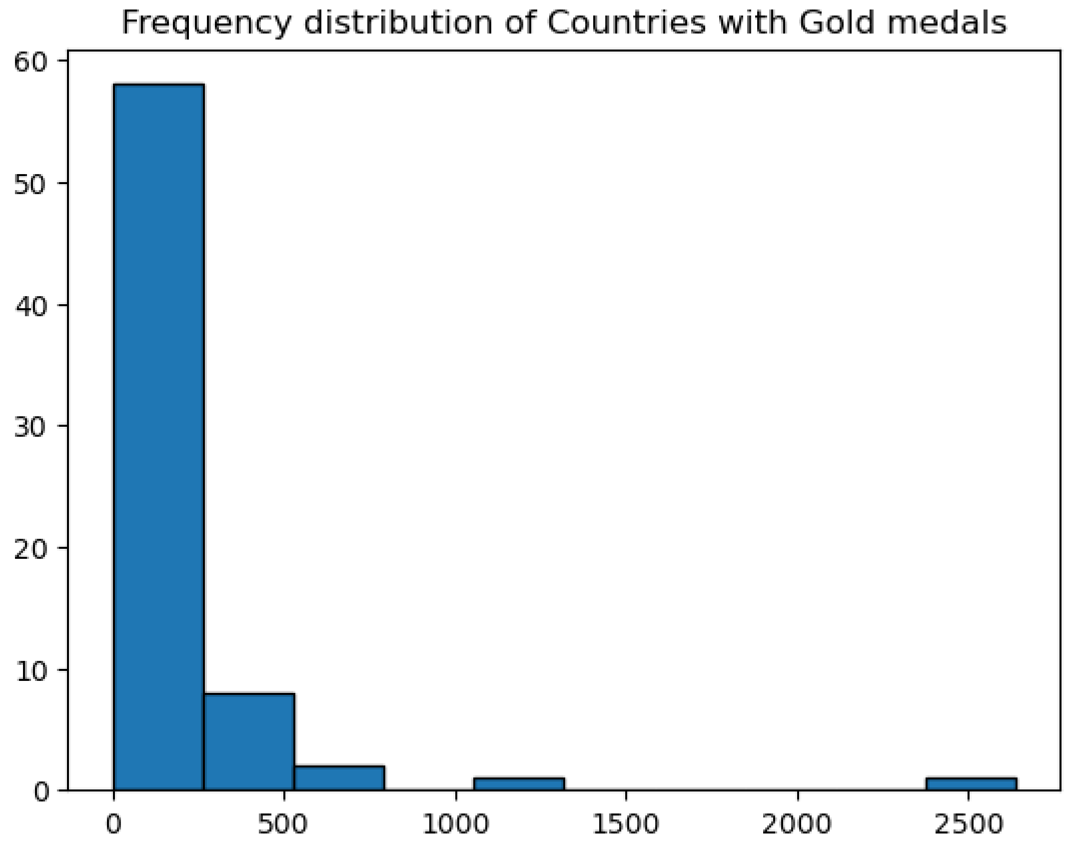
As shown in the above graph, it can be inferred that more than 50 countries have their medal counts upto approximately 500 to 600. Moreover, 10 countries shows medal count between approximately 600 to nearly 1100, and less than 5 countries with their medal counts above an approximate value of 1700 with just 1 or 2 countries above 3000 or 5000.

In [6]:

```
# histogram of number of countries with gold medals

#creates a new dataframe with filtered data where medal column in DF_Olympics only has value as 3(gold medal)
DF_gold_medals=DF_Olympics[DF_Olympics['Medal'].eq(3)]

#plots histogram with frequency of countries in new dataframe DF_gold_medals
plt.hist(DF_gold_medals['NOC'].value_counts(),bins=10,edgecolor='black')
plt.title('Frequency distribution of Countries with Gold medals') #gives title to the plot
```

Out[6]:

```
Text(0.5, 1.0, 'Frequency distribution of Countries with Gold medals')
```



As shown in the above graph, more than 55 countries have upto 250 gold medals. Nearly 8-10 countries shows gold medal count between approximately 250 to 500, and less than 5 countries with their gold medal count above 500. Additionally, 1 country has gold medal count 2500.

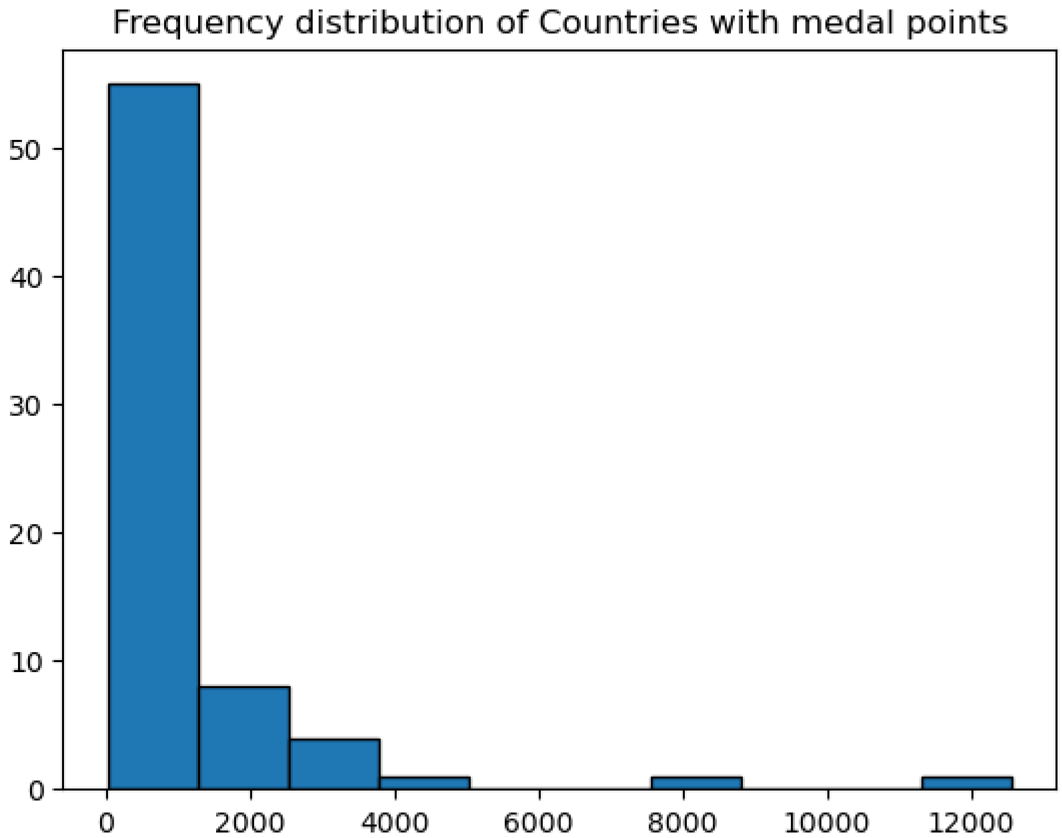
In [7]:
```
# countries against number of points

#creates a series with each NOC and their medal points for each medal type (gold, silver, bronze)
po3_gold=DF_Olympics[DF_Olympics['Medal'].eq(3)].groupby('NOC')['Medal'].count()*3
po2_silver=DF_Olympics[DF_Olympics['Medal'].eq(2)].groupby('NOC')['Medal'].count()*2
po1_bronze=DF_Olympics[DF_Olympics['Medal'].eq(1)].groupby('NOC')['Medal'].count()

#merges all the three series to get a series with NOC and their points
df_NOC_points=po3_gold+po2_silver+po1_bronze
print(df_NOC_points)
```

```
plt.hist(df_NOC_points,bins=10,edgecolor='black')#plots histogram for NOC and its points
plt.title('Frequency distribution of Countries with medal points') #gives title to the plot
```

```
NOC
ANZ       73
ARG      548
AUS     2471
AUT      852
AZE       70
       ...
UKR      345
URU      127
USA    12554
UZB       61
YUG      817
Name: Medal, Length: 70, dtype: int64
```

Out[7]:

```
Text(0.5, 1.0, 'Frequency distribution of Countries with medal points')
```



The plot shows that most of the countries have upto 1000 points. Less than 10 countries have points between 1000 and 2200 approximately and less than 5 countries have more than 22000 points with just 2 to 3 countries with 4000, 8000 and 12000 points.

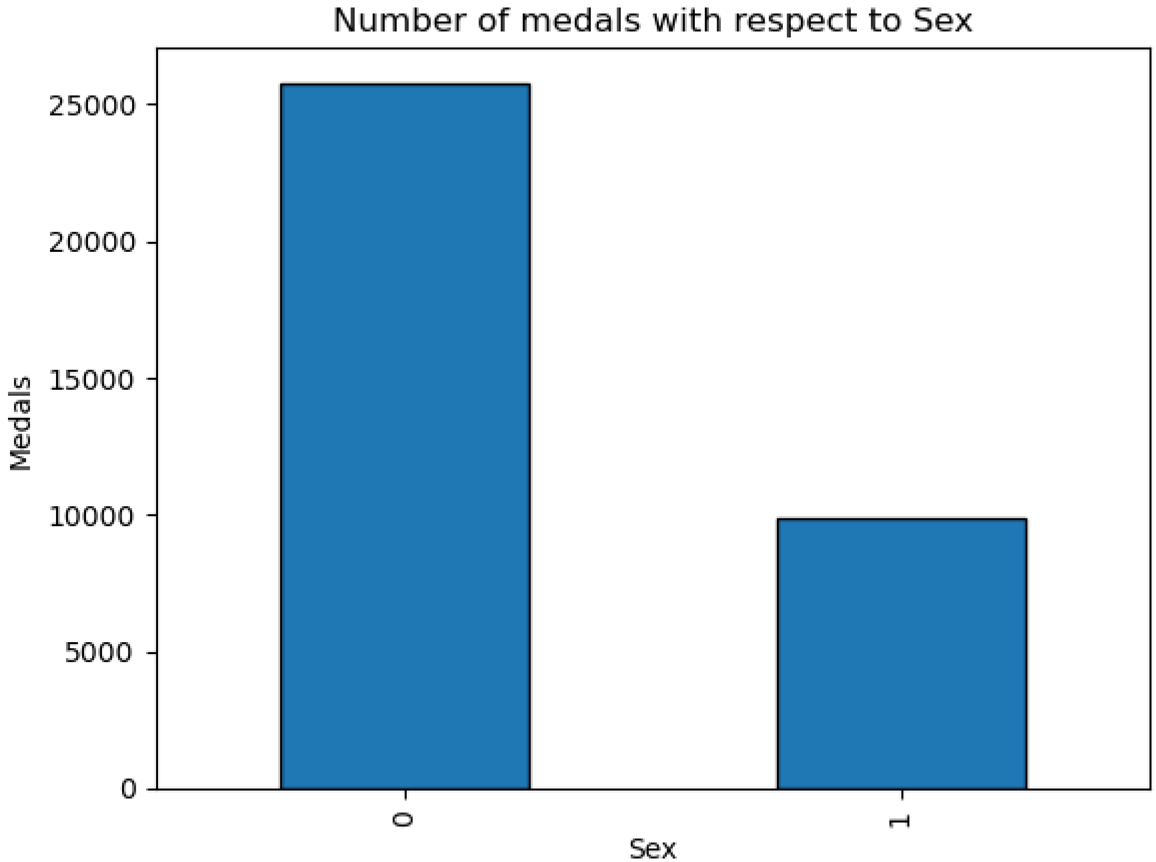
Q1(c)

(i) Create a bar chart of the number of medals for each sex. (ii) Create a bar chart of the number of medals for each season. (iii) Create a histogram of the frequency of the age of the athletes. (iv) Create a histogram of the frequency of the height of the athletes. (v) Create a histogram of the frequency of the weight of the athletes. Comment on these results. (6 marks)

In [8]:
```
#Creating Bar chart for medal count by Sex
medals_wrt_sex=DF_Olympics.groupby('Sex')['Medal'].count() #creates series with each sex(0&1) and their medal counts
```

```
print(medals_wrt_sex)

medals_wrt_sex.plot(kind='bar', edgecolor='black') #plots bar graph for each sex and its medal counts
plt.title('Number of medals with respect to Sex') #gives title to the plot
#labels the x and y axis
plt.xlabel('Sex')
plt.ylabel('Medals')
plt.show()
```

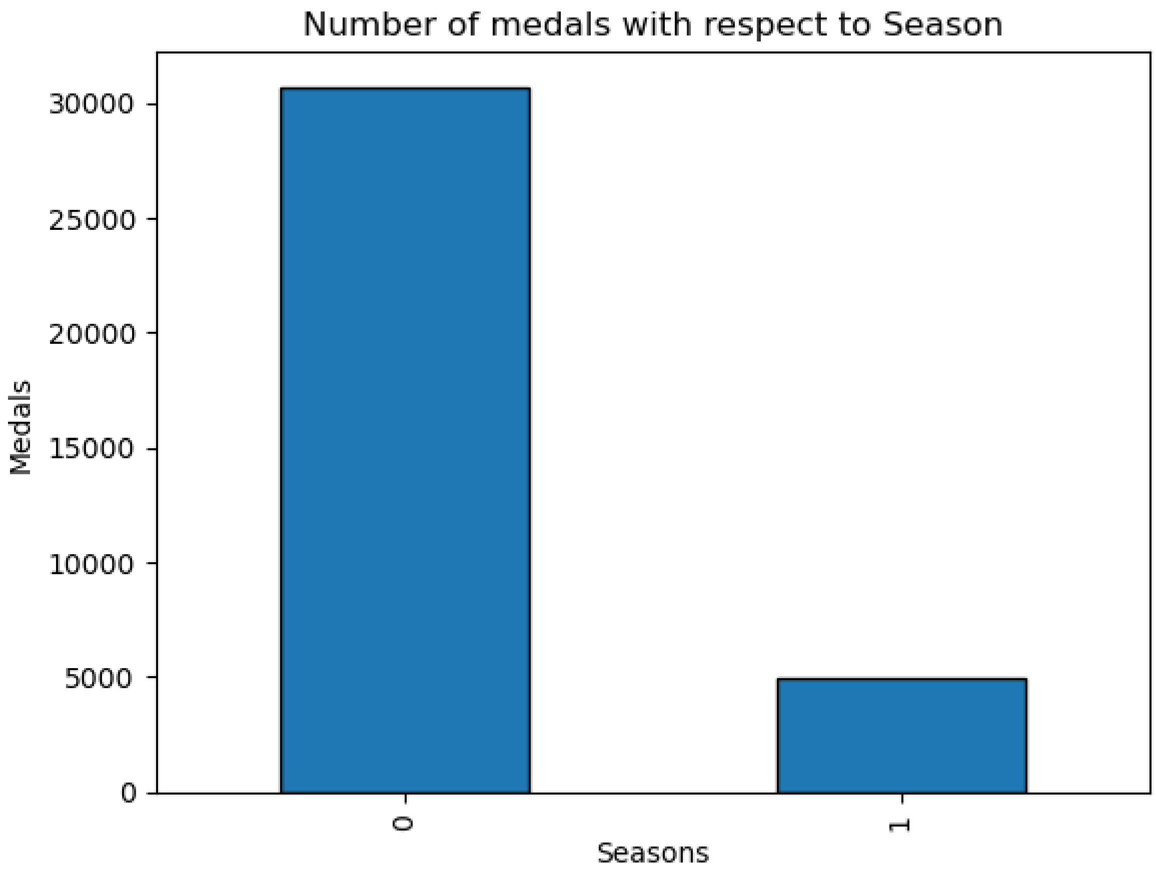
```
Sex
0    25765
1     9904
Name: Medal, dtype: int64
```



The above bar graph shows that the male athletes (sex=0) have won a total of 25000 medals whereas the female atheletes have won a total of 10000 medals

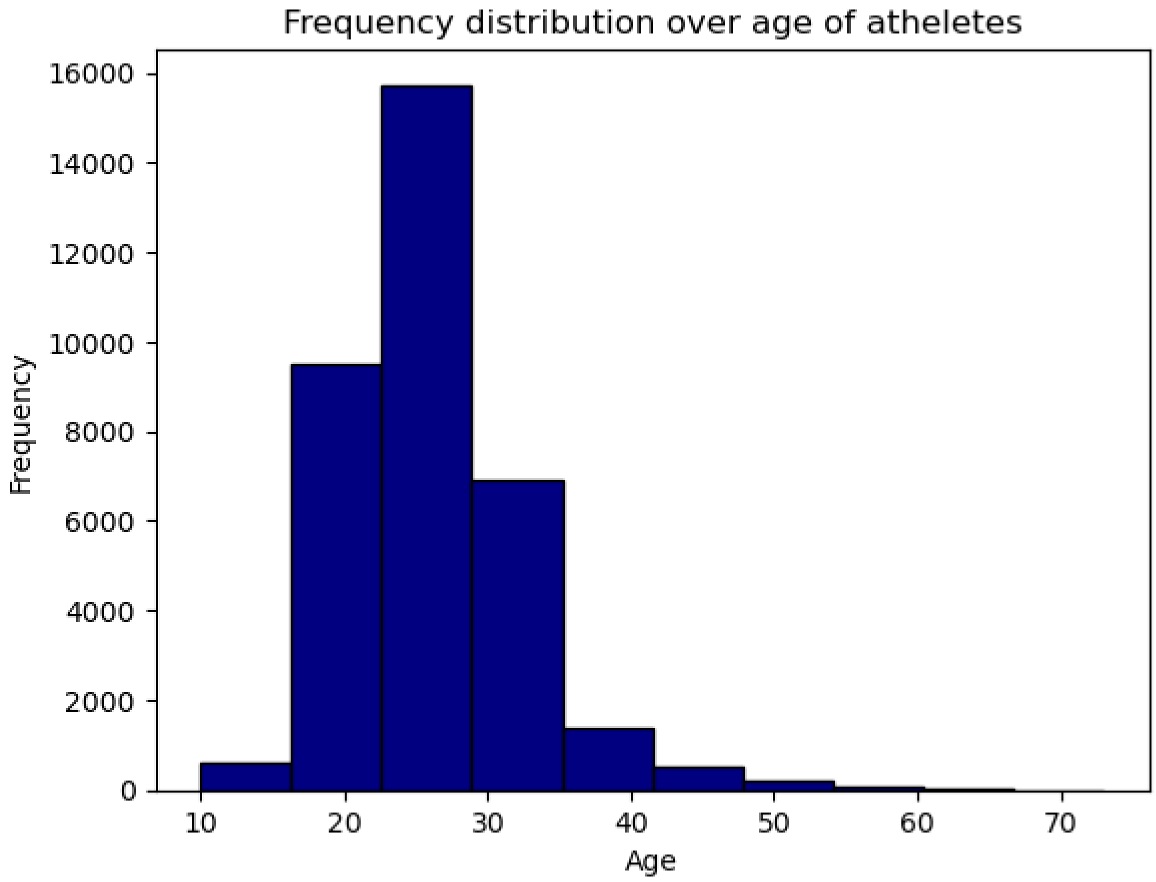
In [9]:
```
#Creating Bar chart for medal count by Sex
medals_wrt_season = DF_Olympics.groupby('Season')['Medal'].count() #creates pandas series with the medal counts for each season
print(medals_wrt_season)

medals_wrt_season.plot(kind='bar', edgecolor='black') #plots bar graph for each season and its medal counts using the series
plt.title('Number of medals with respect to Season') #gives title to the plot
#labels the x and y axis
plt.xlabel('Seasons')
plt.ylabel('Medals')
plt.show()
```

```
Season
0    30686
1     4983
Name: Medal, dtype: int64
```



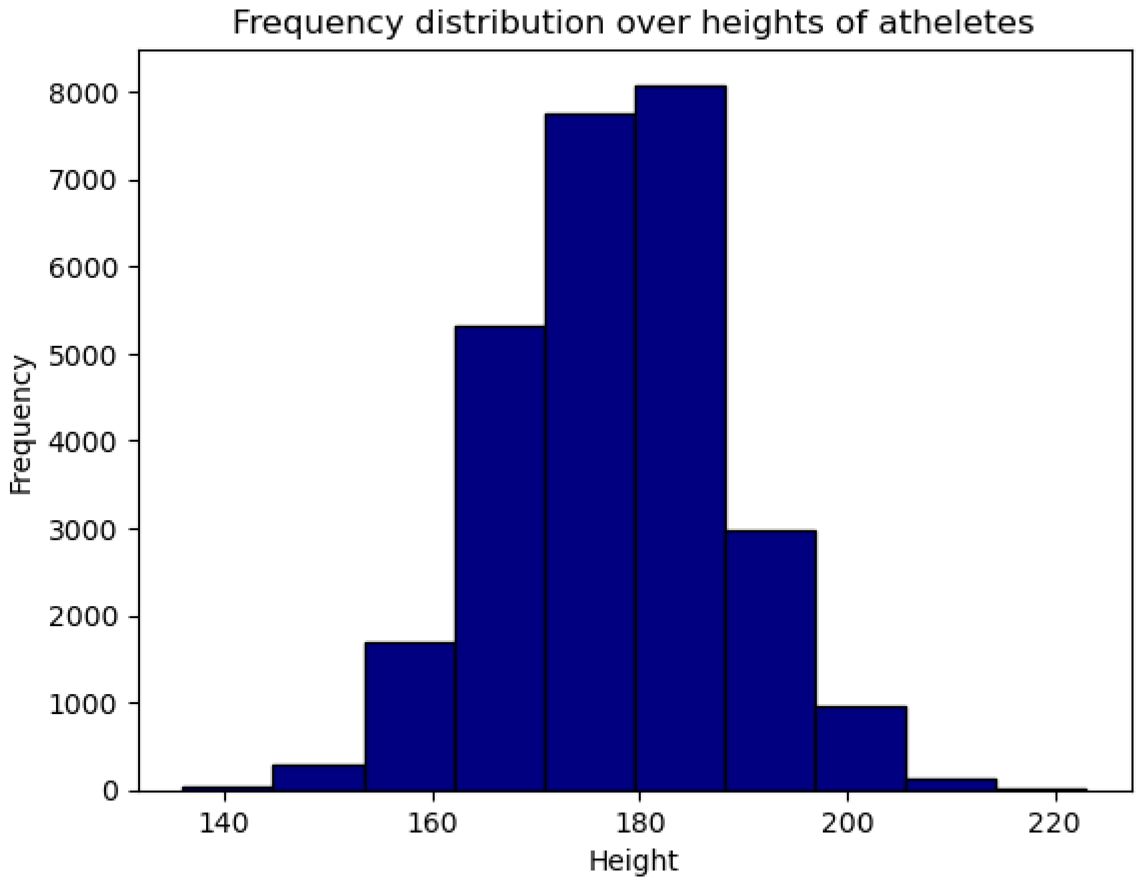
The above bar graph shows that a total of 30000 medals were won during summer(season=0) while in winters the atheletes won only 5000 medals

In [10]:
```
#Histogram of freq of age of atheletes
#plots histogram with frequency of age of the atheletes
plt.hist(DF_Olympics['Age'], edgecolor='black', color='navy') # default bins are 10
plt.title('Frequency distribution over age of atheletes') #gives title to the plot
#labels the x and y axis
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



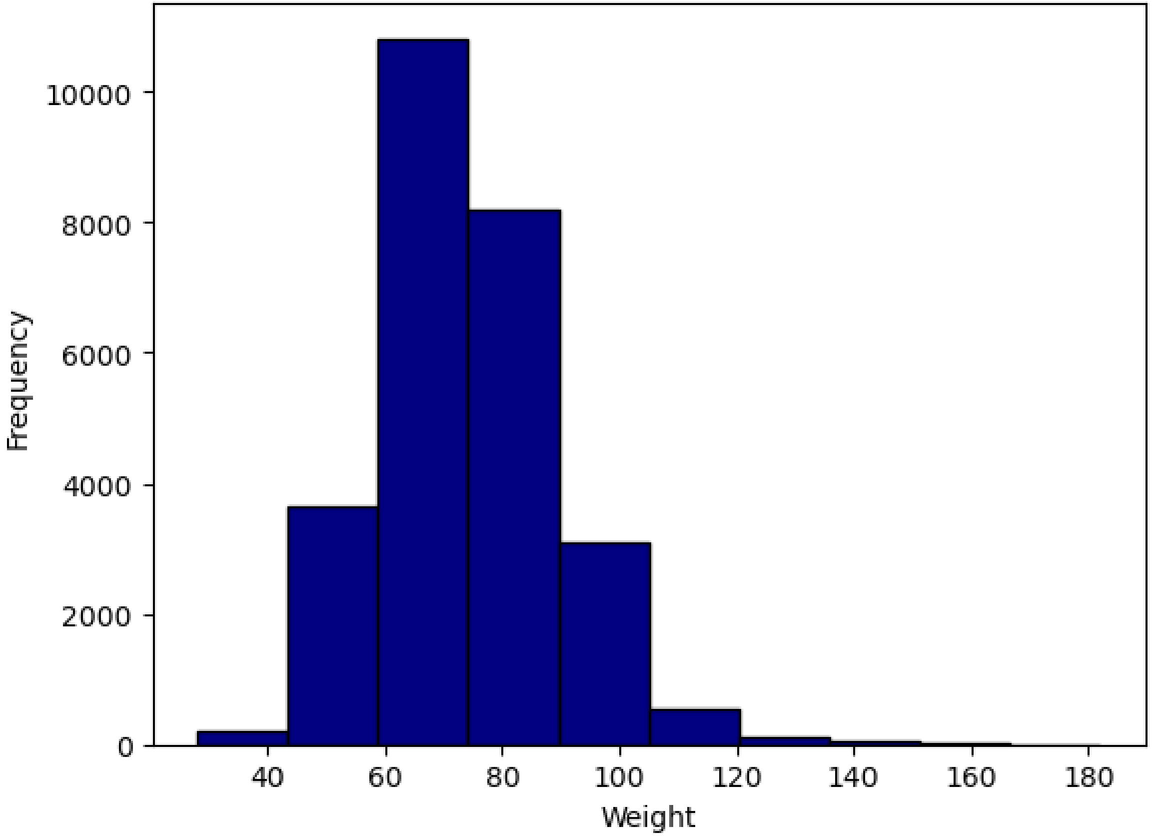
The above frequency distribution shows the number of atheletes in different age groups. The maximum number of atheletes are in the age group 22 to 29, following nearly more than 9000 atheletes in their late teenage and in their early 20s. Approximately 7000 atheletes have their ages between 30 to 35. There are less than 1000 young junior atheletes aged 10 to nearly 16 years. And very few players are above 42 years old

In [11]:
```
#Histogram of freq of height of atheletes
plt.hist(DF_Olympics['Height'], edgecolor='black', color='navy') #plots histogram for column(height) in the dataframe
plt.title('Frequency distribution over heights of atheletes')
#labels the x and y axis
plt.xlabel('Height')
plt.ylabel('Frequency')
plt.show()
```



The above frequency distribution shows that among all the atheletes most of the players have a height between 170 to 190 cms. Nearly 5000 players have a height between approx. 164 to 170. 3000 players have their height between 190 to 200 and less than 500 players are of height more than 200. with a maximum height of players being 220.

In [12]:
```
#Histogram of freq of age of atheletes
plt.hist(DF_Olympics['Weight'], edgecolor='black', color='navy') #plots histogram using column(weight) in the dataframe
plt.xlabel('Weight')
plt.ylabel('Frequency')
plt.show()
```



The above frequency distribution shows the weight distribution of atheletes. Maximum weight any athelete has is between 150 to 160. Few players weight more than 100 and very few weigh less than 40. Nearly 3000 players weigh between 90 to 110 kgs. and approximately 3500 players has their weight between 40 to 60. Maximum number of players has their weight between 60 to 90..

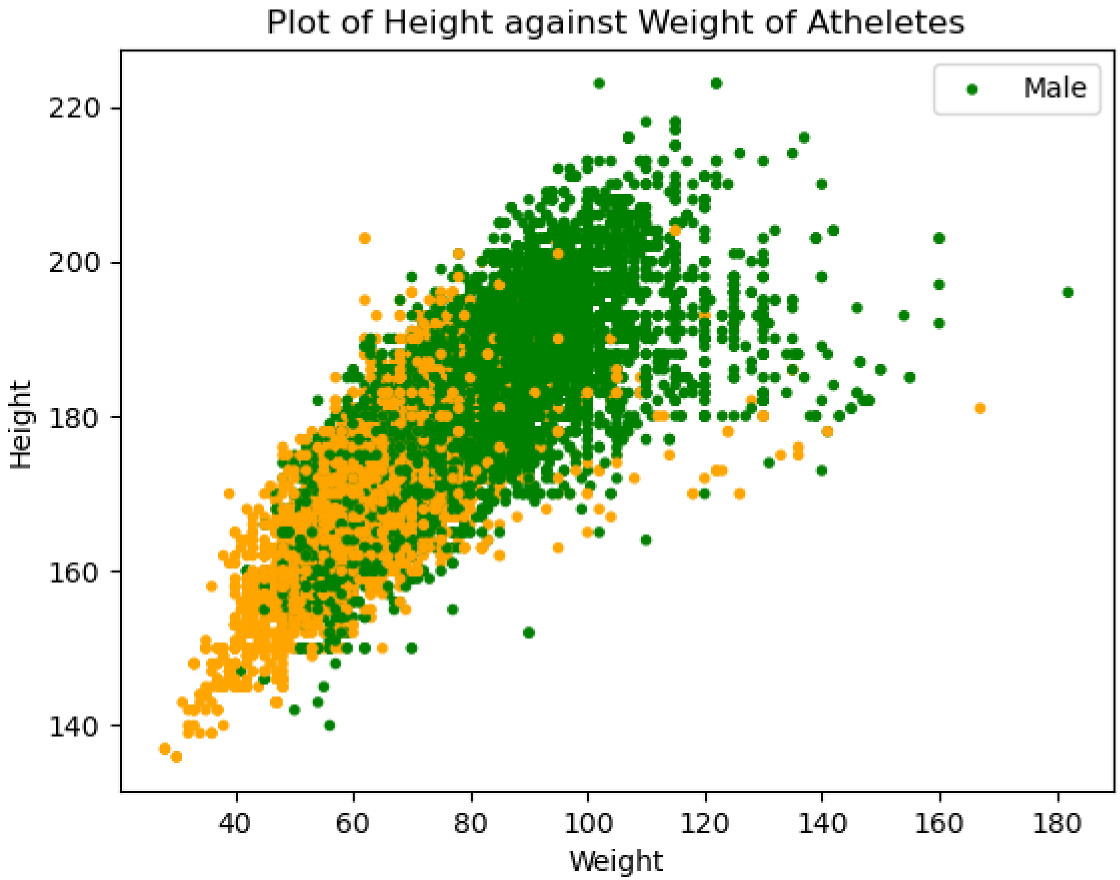
Q1(d)

Remove all the rows which have 'NaN' in the 'Height' or 'Weight' columns. Make a scatter plot of "Weight" and "Height" and colour the points using "Sex". Create a new column called "BMI" equal to the "Weight" divided by the ("Height"/100)^2. Make a scatter plot of "BMI" and "Weight" and colour the points using "Sex". Create a correlation matrix and discuss your results. (6 marks)

In [13]:
```
#Removes the rows having 'NaN' in height and weight columns

DF_Olympics=DF_Olympics.dropna(subset=['Height','Weight'])
print(len(DF_Olympics))
```

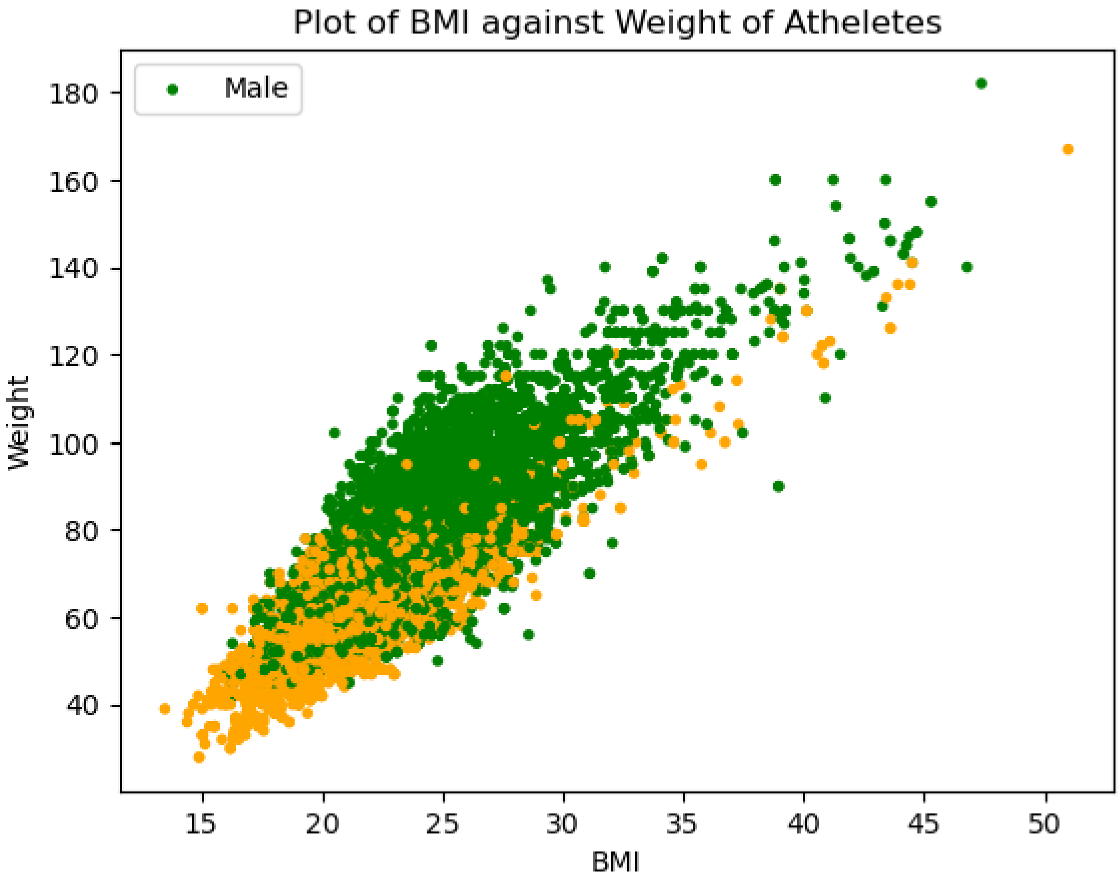
26437

In [14]:
```
color_code={0: 'green', 1:'orange'} #defines dictionary for color coding
legend_map={'Male':'green', 'Female':'orange'} #dictionary to show legend as per the points colored based on sex

#Plots scatter plot of height and weight
plt.scatter(DF_Olympics['Weight'], DF_Olympics['Height'], c=DF_Olympics['Sex'].map(color_code), marker=".") #colored on basis of
plt.title('Plot of Height against Weight of Atheletes')
plt.xlabel('Weight')
plt.ylabel('Height')
plt.legend(legend_map)
plt.show()
```



In [15]:
```
#adds BMI column
DF_Olympics['BMI']=DF_Olympics['Weight']/pow((DF_Olympics['Height']/100),2)
print(DF_Olympics.columns)

#Plots scatter plot of BMI and weight
plt.scatter(DF_Olympics['BMI'], DF_Olympics['Weight'], c=DF_Olympics['Sex'].map(color_code), marker=".") #colored on basis of se
plt.title('Plot of BMI against Weight of Atheletes')
plt.xlabel('BMI')
plt.ylabel('Weight')
plt.legend(legend_map)
plt.show()
```

```
Index(['Sex', 'Age', 'Height', 'Weight', 'NOC', 'Year', 'Season', 'Sport',
       'Event', 'Medal', 'BMI'],
      dtype='object')
```



In [16]:
```
#creates correlation matrix

DF_Olympics.corr()
```

Out[16]:

	Sex	Age	Height	Weight	Year	Season	Medal	BMI
Sex	1.000000	-0.118879	-0.470853	-0.502817	0.252534	0.010973	-0.009598	-0.364814
Age	-0.118879	1.000000	0.096922	0.158269	0.112662	0.043224	-0.013592	0.170008
Height	-0.470853	0.096922	1.000000	0.803173	0.025421	-0.080258	0.037632	0.330621
Weight	-0.502817	0.158269	0.803173	1.000000	0.015696	-0.027479	0.023186	0.821515
Year	0.252534	0.112662	0.025421	0.015696	1.000000	0.107219	-0.033639	-0.011399
Season	0.010973	0.043224	-0.080258	-0.027479	0.107219	1.000000	-0.009207	0.039627
Medal	-0.009598	-0.013592	0.037632	0.023186	-0.033639	-0.009207	1.000000	-0.000261
BMI	-0.364814	0.170008	0.330621	0.821515	-0.011399	0.039627	-0.000261	1.000000

The above correlation matrix shows the best and worst relationships between two columns in the Olympics dataframe. Correlation being 1 is the relationship of a column to itself is the best/perfect linear relationship. The columns weight and BMI have a very good relationship followed by columns Height and Weight with 0.821 and 0.803 as thir correlation coefficients. The column Season has a bad linear relationship with a lowest correlation score of 0.01 with the column sex. The negative values denotes a negative correation between corresponding columns. Highest negative correlation is between Sex and Weight column.

Q1(e)

When "Sex" equals 0, remove all rows with "Age" greater than 43, or "Age" less than 18, or "Height" less than 158, or "Height" greater than 205, or "Weight" less than 52 or "Weight" greater than 120. When "Sex" equals 1, remove all rows with "Age" greater than 39, or "Age" less than 15, or "Height" less than 148, or "Height" greater than 193, or "Weight" less than 40 or "Weight" greater than 94. Remove rows with "Year"'s less than 1948 and remove "Season"'s equal to 1. Remove the "Season" column. Now remove "Sports" which have less than 200 medals. Now make a bar chart of the number of medals in each "Sport". Comment on this. Now remove the "Events" column. Name your dataframe "dfe". (6 marks)

In [17]:

```
#function to check if sex=0, if yes, it removes the rows as per conditions specified.
def remove_rows_if_sex_0(df_local): #function takes 1 argument as a local dataframe
    df_local= df_local.drop(df_local[(df_local['Sex']==0) & ((df_local['Age']>43) | (df_local['Age']<18) | (df_local['Height']<1
    return df_local #returns the updated dataframe with filtered rows
```

In [18]:
```
#function to check if sex=1, if yes, it removes the rows as per conditions specified.
def remove_rows_if_sex_1(df_local): #function takes 1 argument as a local dataframe
    df_local= df_local.drop(df_local[(df_local['Sex']==1) & ((df_local['Age']>39) | (df_local['Age']<15) | (df_local['Height']<14
    return df_local #returns the updated dataframe with filtered rows
```

In [19]:
```
#function call with dataframe as an input to the function
DF_Olympics= remove_rows_if_sex_0(DF_Olympics)
DF_Olympics= remove_rows_if_sex_1(DF_Olympics)
print('Number of rows after removing rows based on sex', len(DF_Olympics))
```

```
Number of rows after removing rows based on sex 25163
```

In [20]:
```
#drop rows with year less than 1948
DF_Olympics= DF_Olympics.drop(DF_Olympics[DF_Olympics['Year']<1948].index)
print('Number of rows after removing data when year<1948    ', len(DF_Olympics))

#drop rows with Season =1
DF_Olympics= DF_Olympics.drop(DF_Olympics[DF_Olympics['Season']==1].index)
print('Rowcount when filtered on season=1    ', len(DF_Olympics))

#drops column season in the dataframe
DF_Olympics=DF_Olympics.drop(columns=['Season'])
print(DF_Olympics.columns)
```

```
Number of rows after removing data when year<1948     23845
Rowcount when filtered on season=1     19736
Index(['Sex', 'Age', 'Height', 'Weight', 'NOC', 'Year', 'Sport', 'Event',
       'Medal', 'BMI'],
      dtype='object')
```

In [21]:
```
#removes sports with less than 200 medals

#creates series with unique sports and their corresponding medal counts
medalCount_per_sport= DF_Olympics.groupby('Sport')['Medal'].count()
print('Medal Count for each sport')
print(medalCount_per_sport)
print('\n')
#removes sports with medal count less than 200 from series
medalCount_per_sport= medalCount_per_sport[medalCount_per_sport>200]
print('Number of Sports with medal counts less than 200   ', len(medalCount_per_sport))

#drops rows where which has medal count below 200
```

```
DF_Olympics= DF_Olympics.drop(DF_Olympics[~DF_Olympics['Sport'].isin(medalCount_per_sport.index)].index)
DF_Olympics
```

```
Medal Count for each sport
Sport
Archery                  190
Art Competitions           1
Athletics               2297
Badminton                140
Baseball                 332
Basketball               670
Beach Volleyball          66
Boxing                   487
Canoeing                 932
Cycling                  739
Diving                   251
Equestrianism            473
Fencing                  867
Football                 937
Golf                       4
Gymnastics               690
Handball                 760
Hockey                  1114
Judo                     422
Modern Pentathlon        122
Rhythmic Gymnastics       75
Rowing                  1806
Rugby Sevens              61
Sailing                  610
Shooting                 418
Softball                 162
Swimming                2156
Synchronized Swimming    158
Table Tennis             153
Taekwondo                127
Tennis                   157
Trampolining              25
Triathlon                 30
Volleyball               705
Water Polo               623
Weightlifting            304
Wrestling                672
Name: Medal, dtype: int64


Number of Sports with medal counts less than 200   22
```

Out[21]:

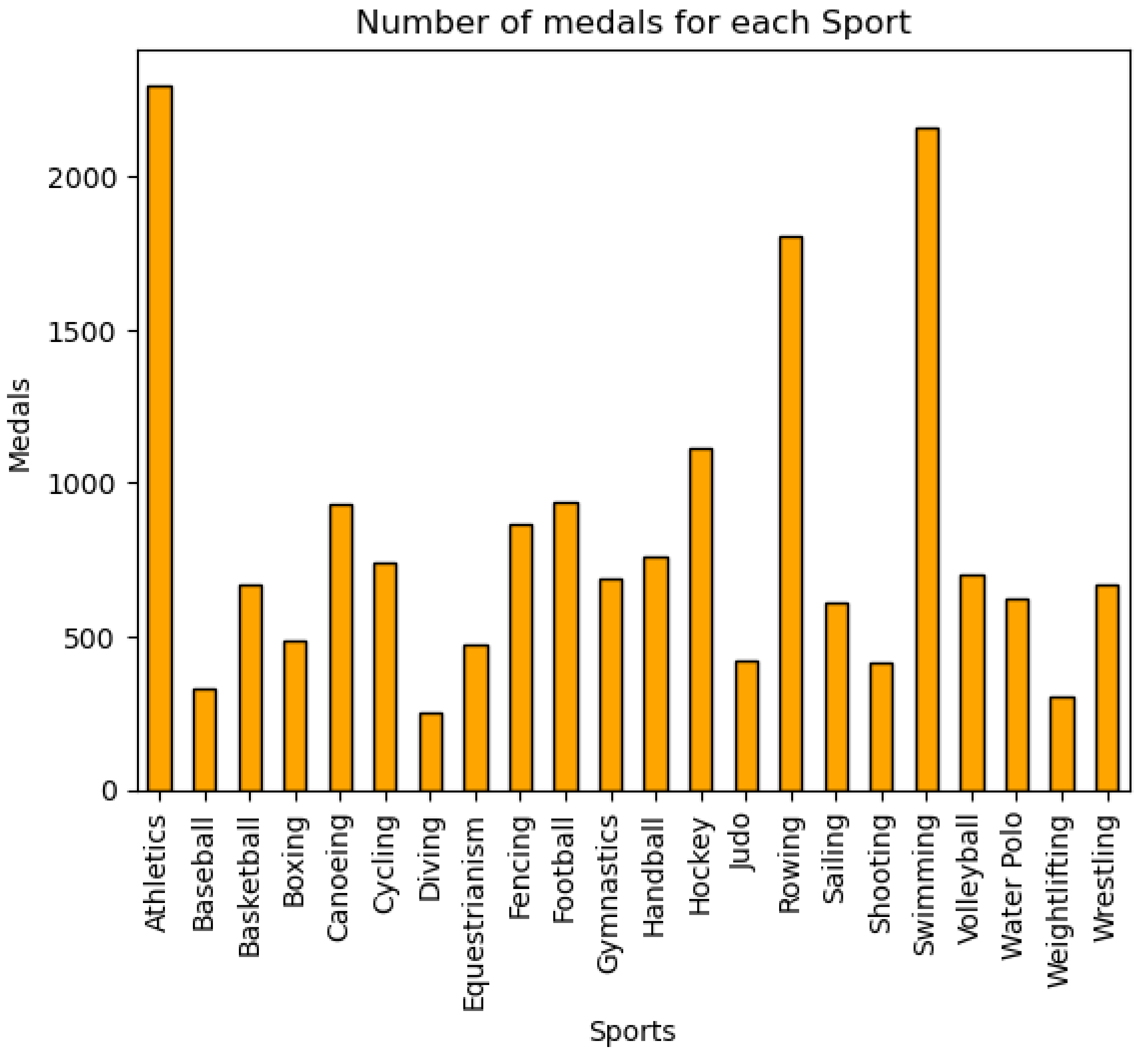
	Sex	Age	Height	Weight	NOC	Year	Sport	Event	Medal	BMI
41	0	28.0	175.0	64.0	FIN	1948	Gymnastics	Gymnastics Men's Individual All-Around	1	20.897959
42	0	28.0	175.0	64.0	FIN	1948	Gymnastics	Gymnastics Men's Team All-Around	3	20.897959
44	0	28.0	175.0	64.0	FIN	1948	Gymnastics	Gymnastics Men's Horse Vault	3	20.897959
48	0	28.0	175.0	64.0	FIN	1948	Gymnastics	Gymnastics Men's Pommelled Horse	3	20.897959
50	0	32.0	175.0	64.0	FIN	1952	Gymnastics	Gymnastics Men's Team All-Around	1	20.897959
...	...	...	...	...	...	...	...	...	...	...
271032	1	22.0	181.0	78.0	NED	1996	Judo	Judo Women's Middleweight	1	23.808797
271046	0	21.0	175.0	70.0	POL	1980	Athletics	Athletics Men's 4 x 100 metres Relay	2	22.857143
271048	0	27.0	197.0	93.0	NED	1992	Rowing	Rowing Men's Double Sculls	1	23.963514
271049	0	31.0	197.0	93.0	NED	1996	Rowing	Rowing Men's Coxed Eights	3	23.963514
271082	0	28.0	182.0	82.0	POL	1980	Fencing	Fencing Men's Foil, Team	1	24.755464

18265 rows × 10 columns

In [22]:

```
#creates a series with Unique sports and their medal counts from the updated dataframe having sports with medal count more than 2
medals_per_sport= DF_Olympics.groupby('Sport')['Medal'].count()
print(medals_per_sport)

#plots a bar graph showing medals won by atheletes in each sport using plot function.
medals_per_sport.plot(kind='bar', edgecolor='black', color='orange')
plt.title('Number of medals for each Sport')
plt.xlabel('Sports')
plt.ylabel('Medals')
plt.show()
```

```
Sport
Athletics        2297
Baseball          332
Basketball        670
Boxing            487
Canoeing          932
Cycling           739
Diving            251
Equestrianism     473
Fencing           867
Football          937
Gymnastics        690
Handball          760
Hockey           1114
Judo              422
Rowing           1806
Sailing           610
Shooting          418
Swimming         2156
Volleyball        705
Water Polo        623
Weightlifting     304
Wrestling         672
Name: Medal, dtype: int64
```



In the above bar graph, the count of medals won is shown for each sport. The maximum number of medals are for Atheletics, followed by swimming and rowing with medals more than 1500. Sports like Baseball, Diving have the least number of medals with thier medal counts below 500. Remaining all the sports have their medal counts close to 500 and between 500 and 1000.

In [23]:
```
#removes events column and name dataframe dfe
dfe=DF_Olympics.drop(columns=['Event'])
print(dfe)
```

```
        Sex   Age  Height  Weight  NOC  Year       Sport  Medal        BMI
41        0  28.0   175.0    64.0  FIN  1948  Gymnastics      1  20.897959
42        0  28.0   175.0    64.0  FIN  1948  Gymnastics      3  20.897959
44        0  28.0   175.0    64.0  FIN  1948  Gymnastics      3  20.897959
48        0  28.0   175.0    64.0  FIN  1948  Gymnastics      3  20.897959
50        0  32.0   175.0    64.0  FIN  1952  Gymnastics      1  20.897959
...     ...   ...     ...     ...  ...   ...         ...    ...        ...
271032    1  22.0   181.0    78.0  NED  1996        Judo      1  23.808797
271046    0  21.0   175.0    70.0  POL  1980   Athletics      2  22.857143
271048    0  27.0   197.0    93.0  NED  1992      Rowing      1  23.963514
271049    0  31.0   197.0    93.0  NED  1996      Rowing      3  23.963514
271082    0  28.0   182.0    82.0  POL  1980     Fencing      1  24.755464

[18265 rows x 9 columns]
```

Q1(f)

Normalise the "Age", "Height", "Weight", "Year", "Medal" and "BMI" columns. In this part of the question just consider the case when the "Sport" equals "Baseball". You should split the data into (80%) training data and (20%) test data. Use an appropriate linear model from sklearn to predict "NOC" using the normalised values "Sex", "Age", "Height", "Weight", "Year", "Medal" and "BMI". Test your model using the test data set. Discuss your results. (6 marks)

In [24]:

```
#creates a dataframe to normalise columns #['Age','Height','Year','Medal','BMI']
normalise_df=dfe.copy()

#function to normalise column using min max normalisation
def normalise_column(normalise_df,col_name):
    normalise_df[col_name]=(normalise_df[col_name]-normalise_df[col_name].min())/(normalise_df[col_name].max()-normalise_df[col_
    return normalise_df[col_name]

#calls function with dataframe and column  to be normalised as arguments and returns in the column value of the original datafran
normalise_df['Age']= normalise_column(normalise_df,'Age')
normalise_df['Height']= normalise_column(normalise_df,'Height')
normalise_df['Weight']= normalise_column(normalise_df,'Weight')
normalise_df['Year']= normalise_column(normalise_df,'Year')
normalise_df['Medal']= normalise_column(normalise_df,'Medal')
normalise_df['BMI']= normalise_column(normalise_df,'BMI')


normalise_df
```

Out[24]:

	Sex	Age	Height	Weight	NOC	Year	Sport	Medal	BMI
41	0	0.464286	0.473684	0.3000	FIN	0.000000	Gymnastics	0.0	0.205268
42	0	0.464286	0.473684	0.3000	FIN	0.000000	Gymnastics	1.0	0.205268
44	0	0.464286	0.473684	0.3000	FIN	0.000000	Gymnastics	1.0	0.205268
48	0	0.464286	0.473684	0.3000	FIN	0.000000	Gymnastics	1.0	0.205268
50	0	0.607143	0.473684	0.3000	FIN	0.058824	Gymnastics	0.0	0.205268
...	...	...	...	...	...	...	...	...	...
271032	1	0.250000	0.578947	0.4750	NED	0.705882	Judo	0.0	0.317432
271046	0	0.214286	0.473684	0.3750	POL	0.470588	Athletics	0.5	0.280762
271048	0	0.428571	0.859649	0.6625	NED	0.647059	Rowing	0.0	0.323394
271049	0	0.571429	0.859649	0.6625	NED	0.705882	Rowing	1.0	0.323394
271082	0	0.464286	0.596491	0.5250	POL	0.470588	Fencing	0.0	0.353911

18265 rows × 9 columns

In [25]:

```
#dataframe with sport = baseball

df__Baseball= normalise_df.drop(normalise_df[~normalise_df['Sport'].eq('Baseball')].index) #drops column where sport is other th
df__Baseball= df__Baseball.reset_index(drop=True) #resets the index of the dataframe and drops the previousy created index
print(df__Baseball)
```

```
     Sex       Age    Height  Weight  NOC      Year     Sport  Medal       BMI
0      0  0.250000  0.596491  0.5500  USA  0.764706  Baseball    1.0  0.377177
1      0  0.464286  0.614035  0.5625  JPN  0.823529  Baseball    0.0  0.378033
2      0  0.250000  0.684211  0.5500  USA  0.764706  Baseball    1.0  0.325620
3      0  0.428571  0.473684  0.6625  CUB  0.647059  Baseball    1.0  0.570155
4      0  0.571429  0.473684  0.6625  CUB  0.705882  Baseball    1.0  0.570155
..   ...       ...       ...     ...  ...       ...       ...    ...       ...
327    0  0.607143  0.614035  0.5500  AUS  0.823529  Baseball    0.5  0.366526
328    0  0.500000  0.736842  0.5750  USA  0.764706  Baseball    1.0  0.317969
329    0  0.500000  0.596491  0.5000  TPE  0.647059  Baseball    0.5  0.330644
330    0  0.571429  0.596491  0.8250  USA  0.764706  Baseball    1.0  0.633104
331    0  0.392857  0.421053  0.4625  USA  0.764706  Baseball    1.0  0.402930

[332 rows x 9 columns]
```

In [26]:

```
from sklearn.svm import LinearSVC  #imports class linear svc from sklearn library
from sklearn.pipeline import make_pipeline #imports make_pipeline fuction from sklearn
from sklearn.preprocessing import StandardScaler #imports StandardScaler which is an object that performs scaling on data
from sklearn.linear_model import LogisticRegression #imports Logistic Regression Model from sk learn

x_features = ["Sex", "Age", "Height", "Weight", "Year", "Medal","BMI"] #creates a feature list of column names
x_=df__Baseball[x_features] #creates a dataframe with selected columns
y_=df__Baseball["NOC"] #column value to be predicted

x__trn, x__tst =x_[:80], x_[:20]  #divides data from feature columns into 80% training and remaining as testing data
y__trn,y__tst = y_[:80], y_[:20]  #divides data from y as 80% training and remaining as testing data

#Using Logistic Regression
clsf_model_1= LogisticRegression(max_iter=1000) #use LogisticRegression as classification model
clsf_model_1.fit(x__trn,y__trn)  #fits the training data into the model and train the model
y_predicted = clsf_model_1.predict(x__tst) #test the data using the x_test dataset to predict outputs as y_predicted which is pre
print(y__tst) #prints test data
print(y_predicted) #prints data predicted by model
print("Accuracy using Logistic Regression: ", clsf_model_1.score(x__trn,y__trn)) #prints the accuracy score against the data resu


#Using Linear SVC

#pipeline to train the model to classify the results using LinearSVC
clsf_model_2= make_pipeline(StandardScaler(), LinearSVC(random_state=1)) #StandardScalar() to scale the input to improve the per
clsf_model_2.fit(x__trn,y__trn) #fits the training data into the model and train the model
y_predicted_2=clsf_model_2.predict(x__tst)  #test the data using the x_test dataset to predict outputs as y_predicted which is pr
print("Accuracy using Linear SVC: ", clsf_model_2.score(x__trn,y__trn))
```

```
0     USA
1     JPN
2     USA
3     CUB
4     CUB
5     CUB
6     USA
7     USA
8     AUS
9     JPN
10    CUB
11    USA
12    CUB
13    USA
14    CUB
15    USA
16    CUB
17    KOR
18    USA
19    CUB
Name: NOC, dtype: object
['CUB' 'USA' 'CUB' 'CUB' 'CUB' 'CUB' 'USA' 'USA' 'USA' 'USA' 'CUB' 'USA'
 'CUB' 'USA' 'CUB' 'USA' 'CUB' 'CUB' 'CUB' 'CUB']
Accuracy using Logistic Regression:  0.6
Accuracy using Linear SVC:  0.6375
```

While trying to use a model to classify the countries basis on the feature data, Linear SVC performed slightly better than Linear Regression. Comparing to the test data, it seems to be very limited and any significant conclusion is difficult to make

Q1(g)

You are now going to try to predict "NOC" when the "Sport" column is "Baseball". Now split the data into (80%) training data and (20%) test data. Create any regression model you like using PyTorch; select an appropriate criterion, optimisation algorithm, and learning rate. Train the model and report the training error. Comment on the testing error. (6 marks)

In [27]:
```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn  #Importing nn to use functions to train models
```

```
from sklearn.model_selection import train_test_split #to split the data into training and testing
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder #to encode outr string values to int



# Load data
input_cols = ["Sex", "Age", "Height", "Weight", "Year", "Medal", "BMI"]
output_col = "NOC"
X_data = df__Baseball[input_cols]
Y_data = df__Baseball[output_col]

# Convert String target values to numerical values
Y_data = LabelEncoder().fit_transform(Y_data)

# Preprocess the input X_data using scaling to improve performance
X_data = StandardScaler().fit_transform(X_data)

# Split data
X_trn, X_tst, Y_trn, Y_tst = train_test_split(X_data, Y_data, test_size=0.2, random_state=42)

# Convert data to PyTorch tensors
X_trn = torch.Tensor(X_trn)
X_tst = torch.Tensor(X_tst)
# Convert target values to PyTorch tensors
Y_trn = torch.Tensor(Y_trn)
Y_tst = torch.Tensor(Y_tst)
#torch.LongTensor

# Define logistic regression model
class Basic(nn.Module): #creating class basic
    def __init__(self, input_size, output_size): #initialising the model
        #super.__init__()
        super(Basic, self).__init__() #nn module is getting cslled by super
        self.linear = nn.Linear(input_size, output_size) #calls self to make a linear layer

    def forward(self, x):
        out = self.linear(x)
        return out

# Initializing model
input_dim_ = len(input_cols) #input is lengths of input columns
output_dim_ = len(df__Baseball[output_col].unique()) #ouput is number of unique NOCs
clf_model = Basic(input_dim_, output_dim_) #defining the model with input and output dimensions
```

```
# Define loss function, optimization algorithm, and learning rate
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(clf_model.parameters(), lr=0.01)    #Adam is Stocastic Gradient Descent method optimisation techniqu
#optimizer = torch.optim.SGD(clf_model.parameters(), lr=learning_rate)     #giving decreased accuracy


# Train the model
total_iter = 1000
for i in range(total_iter):
    optimizer.zero_grad() #clears gradient based on parameters
    outputs = clf_model(X_trn) # Forward pass to get the output
    loss = criterion(outputs, Y_trn.long()) #calculates loss(cross entropy loss)

    # Backward and optimize
    loss.backward() #gets gradient with parameters
    optimizer.step() #updating the parameters

    if (i+1) % 100 == 0:
        print("Iter [{}/{}], Loss: {:.4f}".format(i+1, total_iter, loss.item())) #print loss for each iteration

# Evaluate training error
with torch.no_grad():
    outputs = clf_model(X_trn)  #forward pass to get outputs from training data
    _, predicted = torch.max(outputs.data, 1) #gets predictions with the maximum value
    train_accuracy = 100*((predicted == Y_trn.long()).sum().item() / Y_trn.size(0)) #calculates accuracy from the total correct p
    print("Training Accuracy: {:.4f}".format(train_accuracy)) #prints the training Saccuracy

# Evaluate testing error
with torch.no_grad():
    outputs = clf_model(X_tst) #forward pass to get outputs from testing data
    _, predicted = torch.max(outputs.data, 1) #gets predictions with the maximum value
    test_accuracy = 100*((predicted == Y_tst.long()).sum().item() / Y_tst.size(0)) #calculates accuracy from the total correct pr
    print("Testing Accuracy: {:.4f}".format(test_accuracy)) #prints the test accuracy
```

```
Iter [100/1000], Loss: 1.2159
Iter [200/1000], Loss: 1.1604
Iter [300/1000], Loss: 1.1455
Iter [400/1000], Loss: 1.1384
Iter [500/1000], Loss: 1.1338
Iter [600/1000], Loss: 1.1304
Iter [700/1000], Loss: 1.1277
Iter [800/1000], Loss: 1.1255
Iter [900/1000], Loss: 1.1237
Iter [1000/1000], Loss: 1.1221
Training Accuracy: 54.7170
Testing Accuracy: 59.7015
```

Based on the above output, training accuracy is less than the testing accuracy. Ideally it should be similar but in this case the accuracy itself is very low suggesting that the model is not able to capture all the relevent data. The model might not be able to get all the data patterns due to which it cannot classify new data properly.

Q1(h)

Using your dataset obtained in Q1(e), create a new dataframe called "new_df". Create a list called "Rownames" using the 69 "NOC" values which you will use as the row names. Create a list called "Columnames" using the 22 "Sport" values which you will use as the column names. The values in "new_df" should represent the total numer of points (i.e. the total of the "Medal" number) each NOC "gets" in each "Sport". Obtain the correleation matrix of "new_df". Discuss your results. Write "new_df" to a csv file and read this into R. In R, create 4 different plots to illustrate some statistical properties of "new_df". Discuss any significant results. (10 marks)

In [28]:
```
#creates new dataframe new_df

Rownames=dfe.NOC.unique().tolist() #creates list Rownames of unique NOCs from dataframe dfe
Columnames=dfe.Sport.unique().tolist() #creates list Columnnames of unique Sports from dataframe dfe

print(Rownames)
print(len(Rownames))

print(Columnames)
print(len(Columnames))
```

```
['FIN', 'NED', 'NOR', 'ITA', 'ESP', 'BLR', 'FRA', 'USA', 'HUN', 'AUS', 'IRI', 'CAN', 'PAK', 'UZB', 'AZE', 'GER', 'ETH', 'TUR', 
'BUL', 'EGY', 'GBR', 'SWE', 'JPN', 'ROU', 'MEX', 'SUI', 'NZL', 'ARG', 'CUB', 'POL', 'NGR', 'BRA', 'LTU', 'CHI', 'UKR', 'CRO', 'S
RB', 'IND', 'TTO', 'COL', 'KOR', 'PRK', 'YUG', 'DEN', 'TCH', 'EUN', 'KAZ', 'GEO', 'KEN', 'JAM', 'GRE', 'CHN', 'CZE', 'SVK', 'BA
H', 'POR', 'SCG', 'AUT', 'RSA', 'URU', 'BEL', 'THA', 'EST', 'SLO', 'IRL', 'TPE', 'MGL', 'LAT', 'INA']
69
['Gymnastics', 'Rowing', 'Football', 'Fencing', 'Canoeing', 'Handball', 'Water Polo', 'Wrestling', 'Sailing', 'Athletics', 'Hock
ey', 'Swimming', 'Boxing', 'Basketball', 'Diving', 'Baseball', 'Cycling', 'Judo', 'Volleyball', 'Equestrianism', 'Shooting', 'We
ightlifting']
22
```

In [29]:

```
#creates a series of unique NOC and Sports with thier total Medal as points
medals_per_sport_per_NOC= dfe.groupby(['NOC','Sport'])['Medal'].sum()
medals_per_sport_per_NOC
```

Out[29]:

```
NOC  Sport
ARG  Athletics          7
     Basketball        36
     Boxing             3
     Cycling            6
     Equestrianism      2
                     ... 
YUG  Rowing            10
     Shooting           7
     Swimming           5
     Water Polo       167
     Wrestling         30
Name: Medal, Length: 651, dtype: int64
```

In [30]:

```
new_df = pd.DataFrame(index=Rownames, columns=Columnames)

# Iterate over the row and column names, and assign the values from the grouped DataFrame
for j, rowname in enumerate(Rownames): #iterates over each NOC value
    for k, colname in enumerate(Columnames): #iterates over each Sport value
        NOC_sport = (rowname, colname)
        if NOC_sport in medals_per_sport_per_NOC.index:  #checks if the paired value existed in the series index
            new_df.iloc[j, k] = medals_per_sport_per_NOC[NOC_sport] #takes the corresponding values from series and stores in rov
        else:
            new_df.iloc[j, k] = 0     #if the index keys not in series assign 0 in the dataframe

new_df
```

Out[30]:

	Gymnastics	Rowing	Football	Fencing	Canoeing	Handball	Water Polo	Wrestling	Sailing	Athletics	...	Boxing	Basketball	Diving	Baseball	Cyclin
FIN	31	24	0	0	25	0	0	35	16	64	...	15	0	0	0	
NED	6	188	0	0	9	0	46	0	32	27	...	7	0	0	0	9
NOR	0	55	66	2	40	125	0	1	26	24	...	0	0	0	0	
ITA	21	134	17	366	46	0	206	10	22	79	...	49	38	22	0	15
ESP	8	4	43	1	49	58	78	1	88	23	...	0	82	0	0	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
IRL	0	4	0	0	0	0	0	0	6	7	...	13	0	0	0	
TPE	0	0	0	0	0	0	0	0	0	3	...	0	0	0	40	
MGL	0	0	0	0	0	0	0	13	0	0	...	9	0	0	0	
LAT	5	0	0	0	4	0	0	0	0	6	...	0	0	0	0	
INA	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

69 rows × 22 columns

In [31]:

```
print(new_df.dtypes)
new_df=new_df.astype('int64') #converting values to int to make a proper correlation matrix as the dataframe has
new_df                         #NaN which makes the datatype of dataframe to object
new_df.index.name='NOC' #giving header to index values NOC
```

```
Gymnastics       object
Rowing           object
Football         object
Fencing          object
Canoeing         object
Handball         object
Water Polo       object
Wrestling        object
Sailing          object
Athletics        object
Hockey           object
Swimming         object
Boxing           object
Basketball       object
Diving           object
Baseball         object
Cycling          object
Judo             object
Volleyball       object
Equestrianism    object
Shooting         object
Weightlifting    object
dtype: object
```

In [32]:

```
#correlation matrix of new_df
new_df.corr()
```

Out[32]:

	Gymnastics	Rowing	Football	Fencing	Canoeing	Handball	Water Polo	Wrestling	Sailing	Athletics	...	Boxing	Basketball	Divi
Gymnastics	1.000000	0.441603	0.403329	0.223251	0.318911	0.117766	0.189947	0.636960	0.255858	0.457389	...	0.323995	0.366238	0.5521
Rowing	0.441603	1.000000	0.536838	0.354127	0.711184	0.296926	0.189761	0.299019	0.606528	0.633865	...	0.457667	0.332043	0.3861
Football	0.403329	0.536838	1.000000	0.366670	0.578363	0.318146	0.360821	0.396639	0.452534	0.555574	...	0.408233	0.455326	0.3344
Fencing	0.223251	0.354127	0.366670	1.000000	0.595000	0.376585	0.518885	0.200270	0.154740	0.192989	...	0.341329	0.080004	0.1643
Canoeing	0.318911	0.711184	0.578363	0.595000	1.000000	0.454370	0.331422	0.267021	0.307191	0.298708	...	0.258637	0.009352	0.1355
Handball	0.117766	0.296926	0.318146	0.376585	0.454370	1.000000	0.187025	0.153400	0.248995	0.059989	...	0.084229	0.032760	-0.0000
Water Polo	0.189947	0.189761	0.360821	0.518885	0.331422	0.187025	1.000000	0.317459	0.293894	0.405908	...	0.337429	0.536284	0.2653
Wrestling	0.636960	0.299019	0.396639	0.200270	0.267021	0.153400	0.317459	1.000000	0.213190	0.504977	...	0.473697	0.509217	0.3116
Sailing	0.255858	0.606528	0.452534	0.154740	0.307191	0.248995	0.293894	0.213190	1.000000	0.664086	...	0.372963	0.586793	0.4077
Athletics	0.457389	0.633865	0.555574	0.192989	0.298708	0.059989	0.405908	0.504977	0.664086	1.000000	...	0.647497	0.860915	0.5752
Hockey	0.113596	0.574576	0.256229	0.090730	0.427038	0.136793	0.057933	-0.040611	0.406414	0.190558	...	0.074323	0.024906	0.1211
Swimming	0.468396	0.569148	0.511245	0.145319	0.233183	0.037173	0.473933	0.529293	0.678802	0.926913	...	0.576954	0.923494	0.6252
Boxing	0.323995	0.457667	0.408233	0.341329	0.258637	0.084229	0.337429	0.473697	0.372963	0.647497	...	1.000000	0.554457	0.3925
Basketball	0.366238	0.332043	0.455326	0.080004	0.009352	0.032760	0.536284	0.509217	0.586793	0.860915	...	0.554457	1.000000	0.5726
Diving	0.552145	0.386117	0.334440	0.164330	0.135560	-0.000079	0.265306	0.311640	0.407771	0.575228	...	0.392500	0.572689	1.0000
Baseball	0.247619	0.064768	0.085594	-0.011181	-0.044456	0.005797	0.106254	0.355298	0.141164	0.305988	...	0.716334	0.323489	0.1555
Cycling	0.249696	0.805753	0.390332	0.540054	0.590409	0.321235	0.221318	0.098145	0.682839	0.462675	...	0.366009	0.194316	0.2761
Judo	0.669744	0.255885	0.293280	0.377595	0.220459	0.250841	0.040718	0.479179	0.171292	0.174775	...	0.339860	0.088453	0.1606
Volleyball	0.631395	0.215251	0.578146	0.175510	0.033600	-0.085267	0.259682	0.465976	0.330016	0.455666	...	0.482764	0.499469	0.4916
Equestrianism	0.370433	0.910944	0.574817	0.426420	0.654414	0.275850	0.244008	0.283711	0.685484	0.704715	...	0.485976	0.422709	0.3904
Shooting	0.607225	0.629337	0.488698	0.519419	0.460108	0.253015	0.393592	0.456321	0.464016	0.636187	...	0.546793	0.533123	0.8207
Weightlifting	0.453560	0.198680	0.228531	0.199056	0.186316	0.001003	0.108195	0.425806	0.070079	0.237280	...	0.312305	0.228290	0.7140

22 rows × 22 columns

In [33]:
```
new_df.to_csv('new_df.csv') #downloads the dataframe as csv
```

Q2(a)

Read in the csv file "Gross_Domestic_Product.csv" to create the dataframe "gdp_df". Create a column named "NOC" which takes the values of "Code". Now remove the columns named "Entity" and "Code". In the 'NOC' column, change all 'DEU's to 'GER' and remove all the rows which have "NOC" equals "RUS". Now make sure that your dataframes "gdp_df" and "dfe" have the same values of "NOC" and "Year" and remove rows with different values of "NOC" or different values of "Year". Remove the columns 'Sex', 'Sport', 'Age', 'Height', 'Weight' and 'BMI' from "dfe". (6 marks)

In [34]:
```
#reads csv file
gdp_df = pd.read_csv("C:/Users/prano/OneDrive/Documents/DataScienceProgCoursework/Gross_Domestic_Product.csv")
print(gdp_df)

#creates column named NOC with values from column code
gdp_df['NOC']=gdp_df['Code']

#drops columns 'Code' and 'Entity'
gdp_df=gdp_df.drop(columns=['Entity','Code'])

#in NOC column replace 'DEU' to 'GER'
gdp_df['NOC']=gdp_df['NOC'].replace({'DEU': 'GER'})

#removes rows with NOC= 'RUS'
gdp_df = gdp_df[~gdp_df.NOC.eq('RUS')]
print(gdp_df)
```

```
            Entity Code  Year           GDP
0      Afghanistan  AFG  2002  7.228792e+09
1      Afghanistan  AFG  2003  7.867259e+09
2      Afghanistan  AFG  2004  7.978511e+09
3      Afghanistan  AFG  2005  8.874476e+09
4      Afghanistan  AFG  2006  9.349917e+09
...            ...  ...   ...           ...
10452     Zimbabwe  ZWE  2016  2.011402e+10
10453     Zimbabwe  ZWE  2017  2.106128e+10
10454     Zimbabwe  ZWE  2018  2.207733e+10
10455     Zimbabwe  ZWE  2019  2.072084e+10
10456     Zimbabwe  ZWE  2020  1.942605e+10

[10457 rows x 4 columns]
       Year           GDP  NOC
0      2002  7.228792e+09  AFG
1      2003  7.867259e+09  AFG
2      2004  7.978511e+09  AFG
3      2005  8.874476e+09  AFG
4      2006  9.349917e+09  AFG
...     ...           ...  ...
10452  2016  2.011402e+10  ZWE
10453  2017  2.106128e+10  ZWE
10454  2018  2.207733e+10  ZWE
10455  2019  2.072084e+10  ZWE
10456  2020  1.942605e+10  ZWE

[10425 rows x 3 columns]
```

In [35]:

```
#List of NOC values in dfe to be compared with gdf_df
dfe_noc=list(dfe.NOC.unique()) #creates a list of unique NOC
dfe_year=list(dfe.Year.unique()) #creates a list of unique Year

#removes columns from gdp_df with diff values  of NOC and Year compared to dfe
gdp_df = gdp_df[gdp_df.NOC.isin(dfe_noc)]
gdp_df = gdp_df[gdp_df.Year.isin(dfe_year)]
gdp_df=gdp_df.reset_index(drop=True)
print(gdp_df)
```

```
     Year           GDP  NOC
0    1960  1.510000e+11  ARG
1    1964  1.640000e+11  ARG
2    1968  1.950000e+11  ARG
3    1972  2.370000e+11  ARG
4    1976  2.520000e+11  ARG
..    ...           ...  ...
546  2000  3.147265e+10  UZB
547  2004  3.817499e+10  UZB
548  2008  5.236266e+10  UZB
549  2012  7.010685e+10  UZB
550  2016  9.130956e+10  UZB

[551 rows x 3 columns]
```

In [36]:
```
gdp_noc=list(gdp_df.NOC.unique()) #creates a list of unique NOC
gdp_year=list(gdp_df.Year.unique()) #creates a list of unique Year

#removes columns from dfe with diff values  of NOC and Year compared to gdp_df
dfe = dfe[dfe.NOC.isin(gdp_noc)]
dfe = dfe[dfe.Year.isin(gdp_year)]
dfe=dfe.reset_index(drop=True)
print(dfe)
```

```
       Sex   Age  Height  Weight  NOC  Year      Sport  Medal        BMI
0        1  23.0   182.0    64.0  NOR  1996   Football      1  19.321338
1        0  21.0   198.0    90.0  ITA  2016     Rowing      1  22.956841
2        0  30.0   194.0    87.0  ESP  2008    Fencing      1  23.116165
3        0  28.0   180.0    83.0  BLR  2008   Canoeing      3  25.617284
4        0  23.0   182.0    86.0  FRA  2008   Handball      3  25.963048
...    ...   ...     ...     ...  ...   ...        ...    ...        ...
15052    0  23.0   182.0    90.0  GEO  2004       Judo      3  27.170632
15053    1  28.0   167.0    60.0  GER  2004     Hockey      3  21.513859
15054    0  29.0   175.0    64.0  GER  2016     Hockey      1  20.897959
15055    0  21.0   175.0    70.0  POL  1980  Athletics      2  22.857143
15056    0  28.0   182.0    82.0  POL  1980    Fencing      1  24.755464

[15057 rows x 9 columns]
```

In [37]:
```
#removes specified columns from gdp_df
dfe=dfe.drop(columns=['Sex', 'Sport', 'Age', 'Height', 'Weight', 'BMI'])
dfe=dfe.reset_index(drop=True) #reset index for dataframe and drop previous index
print(dfe)
```

```
         NOC  Year  Medal
0        NOR  1996      1
1        ITA  2016      1
2        ESP  2008      1
3        BLR  2008      3
4        FRA  2008      3
...      ...   ...    ...
15052    GEO  2004      3
15053    GER  2004      3
15054    GER  2016      1
15055    POL  1980      2
15056    POL  1980      1

[15057 rows x 3 columns]
```

Q2(b)

Read in the csv file "Demographic_Indicators.csv" to create the dataframe "dmg_df". Rename the columns "ISO3_code" as "NOC", "Time" as "Year" and "TPopulation1July" as "Population". In the 'NOC' column, change all 'DEU's to 'GER' and remove all the rows which have "NOC" equals "RUS". Now make sure that your dataframes "dmg_df" and "dfe" have the same values of "NOC" and "Year" and remove rows with different values of "NOC" or different values of "Year". Combine "dfe", "gdp_df" and "dmg_df" into a single dataframe called "com_df". Remove any rows with missing values. Don't forget to reset the index. (7 marks)

In [38]:
```
#reads csv file
dmg_df = pd.read_csv("C:/Users/prano/OneDrive/Documents/DataScienceProgCoursework/Demographic_Indicators.csv")

#rename column "ISO3_code" as "NOC", "Time" as "Year" and "TPopulation1July" as "Population"
dmg_df=dmg_df.rename(columns={"ISO3_code":"NOC", "Time":"Year" ,"TPopulation1July":"Population"})

#in NOC column replace 'DEU' to 'GER'
dmg_df['NOC']=dmg_df['NOC'].replace({'DEU': 'GER'})

#removes rows with NOC= 'RUS'
dmg_df = dmg_df[~dmg_df.NOC.eq('RUS')]


#removes rows from dmg_df with diff values  of NOC & Year compared to dfe
dmg_df = dmg_df[dmg_df.NOC.isin(dfe_noc)]
dmg_df = dmg_df[dmg_df.Year.isin(dfe_year)]
```

```
dmg_df=dmg_df.reset_index(drop=True)
print(dmg_df)
```

```
     SortOrder  NOC     Location  Year  TPopulation1Jan  Population  \
0           33  ETH     Ethiopia  1952        18299.061   18496.797
1           33  ETH     Ethiopia  1956        19912.477   20127.942
2           33  ETH     Ethiopia  1960        21476.715   21739.710
3           33  ETH     Ethiopia  1964        23754.287   24073.696
4           33  ETH     Ethiopia  1968        26409.808   26778.653
..         ...  ...          ...   ...              ...         ...
794        266  NZL  New Zealand  2000         3843.356    3855.266
795        266  NZL  New Zealand  2004         4054.696    4081.408
796        266  NZL  New Zealand  2008         4240.473    4260.239
797        266  NZL  New Zealand  2012         4395.630    4410.284
798        266  NZL  New Zealand  2016         4629.139    4668.081

     TPopulationMale1July  TPopulationFemale1July  PopDensity  PopSexRatio  \
0                9173.931                9322.866     18.4968      98.4025
1                9977.948               10149.994     20.1279      98.3050
2               10771.242               10968.468     21.7397      98.2019
3               11927.690               12146.006     24.0737      98.2026
4               13271.850               13506.804     26.7787      98.2605
..                    ...                     ...         ...          ...
794              1892.614                1962.652     14.5535      96.4315
795              2000.841                2080.568     15.4072      96.1680
796              2083.936                2176.302     16.0823      95.7559
797              2156.372                2253.913     16.6487      95.6723
798              2292.434                2375.647     17.6219      96.4972

     ...  Q0060Male  Q0060Female     Q1550  Q1550Male  Q1550Female     Q1560  \
0    ...   698.5519     631.8291  317.9938   341.7289     293.3906  456.5342
1    ...   682.4952     611.9509  309.8361   335.3421     283.4197  447.2425
2    ...   661.5705     584.8050  299.1512   327.2324     270.1749  434.5690
3    ...   638.9675     558.0422  286.6173   316.0235     256.4305  419.3332
4    ...   630.8221     554.0253  285.1459   312.4993     257.3013  417.4996
..   ...        ...          ...       ...        ...          ...       ...
794  ...   113.7750      74.9784   39.5214    51.7338      27.5436   85.6649
795  ...   100.8973      70.0544   36.9036    46.6374      27.4171   77.9816
796  ...    95.5151      64.1264   33.9391    43.6679      24.5633   72.8382
797  ...    88.4809      58.5083   32.3354    40.5164      24.4755   67.2621
798  ...    86.1389      57.9579   30.7575    38.5345      23.2462   66.4963

     Q1560Male  Q1560Female  NetMigrations    CNMR
0     493.5175     420.4676         -1.468  -0.079
1     485.8067     408.4089        -10.640  -0.528
2     475.6081     391.9841          1.698   0.078
3     462.0942     374.8705         10.745   0.446
```

```
4     457.8608     375.9665          9.992   0.373
..         ...          ...            ...     ...
794   104.3004      67.0746         -6.080  -1.576
795    93.1336      63.0169         24.480   6.016
796    87.7490      58.3014          4.679   1.099
797    82.0020      53.0226         -1.539  -0.349
798    80.4770      53.0021         50.012  10.772

[799 rows x 58 columns]
```

In [39]:

```
dmg_noc=list(dmg_df.NOC.unique()) #creates a unique list of values in NOC
dmg_year=list(dmg_df.Year.unique()) #creates a unique list of values in Year

#removes columns from gdp_df with diff values  of NOC and Year compared to dfe
dfe = dfe[dfe.NOC.isin(dmg_noc)] #compares values from dfe and the created list of unique values NOC in dmg_df
dfe = dfe[dfe.Year.isin(dmg_year)] #compares values from dfe and the created list of \nunique values Year in dmg_df
dfe=dfe.reset_index(drop=True) #reset index for dataframe and drop previous index
print(dfe)
```

```
       NOC  Year  Medal
0      NOR  1996      1
1      ITA  2016      1
2      ESP  2008      1
3      BLR  2008      3
4      FRA  2008      3
...    ...   ...    ...
15052  GEO  2004      3
15053  GER  2004      3
15054  GER  2016      1
15055  POL  1980      2
15056  POL  1980      1

[15057 rows x 3 columns]
```

In [40]:

```
#combines 3 dataframes
com_df= pd.merge(dfe,gdp_df,on=['NOC', 'Year'] ,how='inner' ) #merge the dataframe on common columns using inner join
com_df=pd.merge(com_df,dmg_df,on=['NOC', 'Year'] ,how='inner')
com_df=com_df.dropna() #removes rows where values are Nan
print('com_df after removing Nan')
print(com_df)
```

```
com_df after removing Nan
       NOC  Year  Medal           GDP  SortOrder             Location  \
0      NOR  1996      1  2.660000e+11        172               Norway
1      NOR  1996      1  2.660000e+11        172               Norway
2      NOR  1996      1  2.660000e+11        172               Norway
3      NOR  1996      1  2.660000e+11        172               Norway
4      NOR  1996      1  2.660000e+11        172               Norway
...    ...   ...    ...           ...        ...                  ...
13424  COL  2000      3  1.570000e+11        247             Colombia
13426  TTO  2016      1  2.356165e+10        230  Trinidad and Tobago
13427  IRL  1980      2  5.644437e+10        167              Ireland
13428  IRL  1980      2  5.644437e+10        167              Ireland
13429  MEX  1972      2  3.060000e+11        239               Mexico

       TPopulation1Jan  Population  TPopulationMale1July  \
0             4370.136    4381.510              2166.567
1             4370.136    4381.510              2166.567
2             4370.136    4381.510              2166.567
3             4370.136    4381.510              2166.567
4             4370.136    4381.510              2166.567
...                ...         ...                   ...
13424        38901.889   39215.135             19469.885
13426         1464.853    1469.330               723.726
13427         3372.333    3391.387              1702.594
13428         3372.333    3391.387              1702.594
13429        52710.838   53543.436             26811.805

       TPopulationFemale1July  ...  Q0060Male  Q0060Female     Q1550  \
0                    2214.943  ...   113.5521      68.7246   36.7890
1                    2214.943  ...   113.5521      68.7246   36.7890
2                    2214.943  ...   113.5521      68.7246   36.7890
3                    2214.943  ...   113.5521      68.7246   36.7890
4                    2214.943  ...   113.5521      68.7246   36.7890
...                       ...  ...        ...          ...       ...
13424               19745.251  ...   257.8678     123.6142  105.0326
13426                 745.604  ...   212.3154     126.8777   72.1482
13427                1688.793  ...   191.2425     113.6134   50.8989
13428                1688.793  ...   191.2425     113.6134   50.8989
13429               26731.631  ...   384.0713     294.4447  154.7332

       Q1550Male  Q1550Female     Q1560  Q1560Male  Q1560Female  \
0        47.4570      25.4993   85.7187   106.7067      63.7288
1        47.4570      25.4993   85.7187   106.7067      63.7288
2        47.4570      25.4993   85.7187   106.7067      63.7288
```

```
3        47.4570      25.4993   85.7187   106.7067      63.7288
4        47.4570      25.4993   85.7187   106.7067      63.7288
...          ...          ...       ...        ...          ...
13424   158.5779      49.2377  167.6963   232.1554     100.1840
13426    94.0263      49.7558  152.1490   193.6665     110.2198
13427    63.8224      37.1619  138.8647   175.5694     100.6296
13428    63.8224      37.1619  138.8647   175.5694     100.6296
13429   183.7038     125.6222  255.6998   300.3069     209.8183

       NetMigrations   CNMR
0              5.756  1.315
1              5.756  1.315
2              5.756  1.315
3              5.756  1.315
4              5.756  1.315
...              ...    ...
13424        -31.116 -0.793
13426         -0.103 -0.070
13427         -2.183 -0.643
13428         -2.183 -0.643
13429       -140.810 -2.626

[8431 rows x 60 columns]
```

In [41]:

```python
#each NOC and Year pair to be present only once
com_df= pd.DataFrame(com_df.groupby(['NOC', 'Year', 'GDP', 'Population'])['Medal'].sum()) #drop unwanted columns by grouping on 
#reset index of dataframe com_df
com_df=com_df.reset_index(drop=False) #reset index)
print(com_df)
```

```
     NOC  Year           GDP  Population  Medal
0    ARG  1960  1.510000e+11   20349.744      3
1    ARG  1964  1.640000e+11   21708.487      2
2    ARG  1968  1.950000e+11   23112.971      2
3    ARG  1972  2.370000e+11   24612.794      2
4    ARG  1988  2.940000e+11   31690.792     12
..   ...   ...           ...         ...    ...
278  UZB  2000  3.147265e+10   24925.554      7
279  UZB  2004  3.817499e+10   26234.923     10
280  UZB  2008  5.236266e+10   27726.810      7
281  UZB  2012  7.010685e+10   29503.051      5
282  UZB  2016  9.130956e+10   31453.574     20

[283 rows x 5 columns]
```

Q2(c)

Replace each value in the columns "GDP", "Population" and "Medal" by its logarithm to base e. Then normalise the columns "GDP", "Population" and "Medal" for each "Year". Now split the data into (80%) training data and (20%) test data. Use an appropriate linear model from sklearn to predict the number of points given in the column "Medal" using the normalised values "GDP" and "Population". Test your model using the test data set. Discuss your results. (7 marks)

In [42]:
```
#replacing column values with its logarithmic base
import numpy as np

#taking log of three columns
com_df['GDP']=np.log(com_df['GDP']) #using log function in numpy
com_df['Population']=np.log(com_df['Population'])
com_df['Medal']=np.log(com_df['Medal'])


#calls function created already to normalise column in a dataframe
com_df['GDP']= normalise_column(com_df,'GDP')
com_df['Population']= normalise_column(com_df,'Population')
com_df['Medal']= normalise_column(com_df,'Medal')

print(com_df)
```

```
     NOC  Year       GDP  Population     Medal
0    ARG  1960  0.415791    0.423062  0.165719
1    ARG  1964  0.425814    0.431872  0.104557
2    ARG  1968  0.446826    0.440418  0.104557
3    ARG  1972  0.470500    0.448988  0.104557
4    ARG  1988  0.496656    0.483442  0.374833
..   ...   ...       ...         ...       ...
278  UZB  2000  0.225470    0.450709  0.293529
279  UZB  2004  0.248901    0.457688  0.347331
280  UZB  2008  0.287254    0.465227  0.293529
281  UZB  2012  0.322672    0.473691  0.242774
282  UZB  2016  0.354741    0.482418  0.451888

[283 rows x 5 columns]
```

In [43]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split #to split training ans test data
from sklearn.linear_model import LinearRegression #imports model
from sklearn.preprocessing import StandardScaler #to scale the input data to improve accuracy
from sklearn.metrics import mean_squared_error, mean_absolute_error #imports the functions to calculate error in the model

features = ["GDP", "Population"] #inilialiseing feature columns
x=com_df[features]
y=com_df["Medal"] #column values to be predicted
x=StandardScaler().fit_transform(x)  #scaling the input for improved performance
x_trn, x_tst, y_trn, y_tst = train_test_split(x, y,    # creates 4 data sets using to split as training and esting
                                               random_state = 1,  # A random split that we can repeat each time we run
                                               test_size = 0.2)

# creating a regression model
model = LinearRegression()

# fitting the model
model.fit(x_trn,y_trn)

# making predictions
y_predict = model.predict(x_tst)
print(y_predict[0:5])
print(y_tst[0:5])

print('mean_squared_error : ', mean_squared_error(y_tst, y_predict))  #calculates and prints mean squared error
print('mean_absolute_error : ', mean_absolute_error(y_tst, y_predict)) #calculates and prints absolute error
print('Accuracy score: ', model.score(x_tst,y_tst)) #print accuracy score
```

```
[0.24694749 0.50187082 0.42377864 0.39702561 0.48281068]
99     0.398086
260    0.331438
62     0.728322
102    0.386907
259    0.386907
Name: Medal, dtype: float64
mean_squared_error :  0.039842183581804365
mean_absolute_error :  0.1571995106903354
Accuracy score:  0.2506193722959885
```

Looking at the error and accuracy score, it can be inferred that the model is not the best fit for this problem. Furthermore, with increased data, the model might work more effectively. However, looking at the current figures for error and accuracy, the data might not have proper linear relationship to predict the values or might have less dependency over the feature columns.

References and Citations

https://towardsdatascience.com/

https://scikit-learn.org/