# Olympics Data Analysis

**By: Purnima Agarwal**

## 1) Data set used

We have taken the data set from Kaggle. It is information on the Olympic Games, from Athens 1896 to Rio 2016.
The data set has two files:

- athlete_events.csv
- noc_regions.csv

## 2) Understanding the data

i) We first import the needed libraries.

```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
```

ii) Read the files

```
ath=pd.read_csv('athlete_events.csv')
```

```
ath.head()
```

| | ID | Name | Sex | Age | Height | Weight | Team | NOC | Games | Year | Season | City | Sport | Event | Medal |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | A Dijiang | M | 24.0 | 180.0 | 80.0 | China | CHN | 1992 Summer | 1992 | Summer | Barcelona | Basketball | Basketball Men's Basketball | NaN |
| 1 | 2 | A Lamusi | M | 23.0 | 170.0 | 60.0 | China | CHN | 2012 Summer | 2012 | Summer | London | Judo | Judo Men's Extra-Lightweight | NaN |
| 2 | 3 | Gunnar Nielsen Aaby | M | 24.0 | NaN | NaN | Denmark | DEN | 1920 Summer | 1920 | Summer | Antwerpen | Football | Football Men's Football | NaN |
| 3 | 4 | Edgar Lindenau Aabye | M | 34.0 | NaN | NaN | Denmark/Sweden | DEN | 1900 Summer | 1900 | Summer | Paris | Tug-Of-War | Tug-Of-War Men's Tug-Of-War | Gold |
| 4 | 5 | Christine Jacoba Aaftink | F | 21.0 | 185.0 | 82.0 | Netherlands | NED | 1988 Winter | 1988 | Winter | Calgary | Speed Skating | Speed Skating Women's 500 metres | NaN |

```python
regions=pd.read_csv('noc_regions.csv')
```

```python
regions.head()
```

| | NOC | region | notes |
|---|---|---|---|
| 0 | AFG | Afghanistan | NaN |
| 1 | AHO | Curacao | Netherlands Antilles |
| 2 | ALB | Albania | NaN |
| 3 | ALG | Algeria | NaN |
| 4 | AND | Andorra | NaN |

## iii) Exploring the data

- Number of rows and columns in the **athlete table**- 271116, 15 and column names in the table.

```python
ath.shape
```

```
(271116, 15)
```

```python
ath.columns
```

```
Index(['ID', 'Name', 'Sex', 'Age', 'Height', 'Weight', 'Team', 'NOC', 'Games',
       'Year', 'Season', 'City', 'Sport', 'Event', 'Medal'],
      dtype='object')
```

- Different datatypes in the table:

```python
ath.dtypes
```

```
ID          int64
Name       object
Sex        object
Age       float64
Height    float64
Weight    float64
Team       object
NOC        object
Games      object
Year        int64
Season     object
City       object
Sport      object
Event      object
Medal      object
dtype: object
```

- Finding the number of null values:

```
ath.isnull().sum()
```

```
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
```

- Using Describe Function:

```
ath.describe()
```

| | 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 |

- Using info() function:

```
ath.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
```

- Now for the regions table

```
regions.shape
```

```
(230, 3)
```

```
regions.columns
```

```
Index(['NOC', 'region', 'notes'], dtype='object')
```

- Describing the data

```
regions.describe()
```

| | NOC | region | notes |
|---|---|---|---|
| count | 230 | 227 | 21 |
| unique | 230 | 206 | 21 |
| top | ANT | Germany | Virgin Islands |
| freq | 1 | 4 | 1 |

```
regions.isnull().sum()
```

```
NOC         0
region      3
notes     209
dtype: int64
```

```
regions.dtypes
```

```
NOC       object
region    object
notes     object
dtype: object
```

- Using info() function:

```
regions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 230 entries, 0 to 229
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   NOC     230 non-null    object
 1   region  227 non-null    object
 2   notes   21 non-null     object
dtypes: object(3)
memory usage: 5.5+ KB
```

## 3) Merging the data set

We will merge both the tables for easy analysis.
We will join the two data frames using as key the NOC column with the Pandas merge() function.

```
df=ath.merge(regions,how='left',on='NOC')
```

```
df.columns
```

```
Index(['ID', 'Name', 'Sex', 'Age', 'Height', 'Weight', 'Team', 'NOC', 'Games',
       'Year', 'Season', 'City', 'Sport', 'Event', 'Medal', 'region', 'notes'],
      dtype='object')
```

```
df.head()
```

| | ID | Name | Sex | Age | Height | Weight | Team | NOC | Games | Year | Season | City | Sport | Event | Medal | region | notes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | A Dijiang | M | 24.0 | 180.0 | 80.0 | China | CHN | 1992 Summer | 1992 | Summer | Barcelona | Basketball | Basketball Men's Basketball | NaN | China | NaN |
| 1 | 2 | A Lamusi | M | 23.0 | 170.0 | 60.0 | China | CHN | 2012 Summer | 2012 | Summer | London | Judo | Judo Men's Extra-Lightweight | NaN | China | NaN |
| 2 | 3 | Gunnar Nielsen Aaby | M | 24.0 | NaN | NaN | Denmark | DEN | 1920 Summer | 1920 | Summer | Antwerpen | Football | Football Men's Football | NaN | Denmark | NaN |
| 3 | 4 | Edgar Lindenau Aabye | M | 34.0 | NaN | NaN | Denmark/Sweden | DEN | 1900 Summer | 1900 | Summer | Paris | Tug-Of-War | Tug-Of-War Men's Tug-Of-War | Gold | Denmark | NaN |

```
[16]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 271116 entries, 0 to 271115
Data columns (total 17 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
 15  region  270746 non-null  object
 16  notes   5039 non-null    object
dtypes: float64(3), int64(2), object(12)
memory usage: 37.2+ MB
```

# 4) Cleaning the data set

- **Checking duplicate values**

```
## duplicate values

df.duplicated().sum()
```

```
1385
```

This just shows there are multiple participations

```
df.isnull().sum()
```

```
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
region        370
notes      266077
dtype: int64
```

We saw earlier that the data is mostly clean with some null values. There are null values in Age, Height, Columns, notes and Medals columns. The medal column null values are not to be removed as it represents players who haven't won a medal and notes column is not important. We can drop the other rows.

(We will use the table with dropped values in the ML part.)

- Dropping the null values:

```
df1 = df.dropna(axis=0, subset=['Age','Height','Weight','region'])
```

```
df1.isnull().sum()
```

```
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      175723
region          0
notes      202418
dtype: int64
```

```
df1.shape
```

```
(205895, 17)
```

The number of rows is now reduced to 205895.

# 5) Analysing the data set

## • Analyse for India

### Taking out information about the Indian Athletes

```
df.query('Team=="India"').head(20)
```

| | ID | Name | Sex | Age | Height | Weight | Team | NOC | Games | Year | Season | City | Sport | Event | Medal | region | notes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **505** | 281 | S. Abdul Hamid | M | NaN | NaN | NaN | India | IND | 1928 Summer | 1928 | Summer | Amsterdam | Athletics | Athletics Men's 110 metres Hurdles | NaN | India | NaN |
| **506** | 281 | S. Abdul Hamid | M | NaN | NaN | NaN | India | IND | 1928 Summer | 1928 | Summer | Amsterdam | Athletics | Athletics Men's 400 metres Hurdles | NaN | India | NaN |
| **895** | 512 | Shiny Kurisingal Abraham-Wilson | F | 19.0 | 167.0 | 53.0 | India | IND | 1984 Summer | 1984 | Summer | Los Angeles | Athletics | Athletics Women's 800 metres | NaN | India | NaN |
| **896** | 512 | Shiny Kurisingal Abraham-Wilson | F | 19.0 | 167.0 | 53.0 | India | IND | 1984 Summer | 1984 | Summer | Los Angeles | Athletics | Athletics Women's 4 x 400 metres Relay | NaN | India | NaN |
| **897** | 512 | Shiny Kurisingal Abraham-Wilson | F | 23.0 | 167.0 | 53.0 | India | IND | 1988 Summer | 1988 | Summer | Seoul | Athletics | Athletics Women's 800 metres | NaN | India | NaN |

## • Countries with most medals

```
## Countries with most medals
medal_rank=df.groupby("Team")['Medal'].apply(lambda x: x.notnull().sum()).reset_index(name='Medal')
medal_rank
```

| | Team | Medal |
|---|---|---|
| **0** | 30. Februar | 0 |
| **1** | A North American Team | 4 |
| **2** | Acipactli | 0 |
| **3** | Acturus | 0 |
| **4** | Afghanistan | 2 |
| **...** | ... | ... |
| **1179** | Zambia | 2 |
| **1180** | Zefyros | 0 |
| **1181** | Zimbabwe | 22 |
| **1182** | Zut | 3 |
| **1183** | rn-2 | 0 |

1184 rows × 2 columns

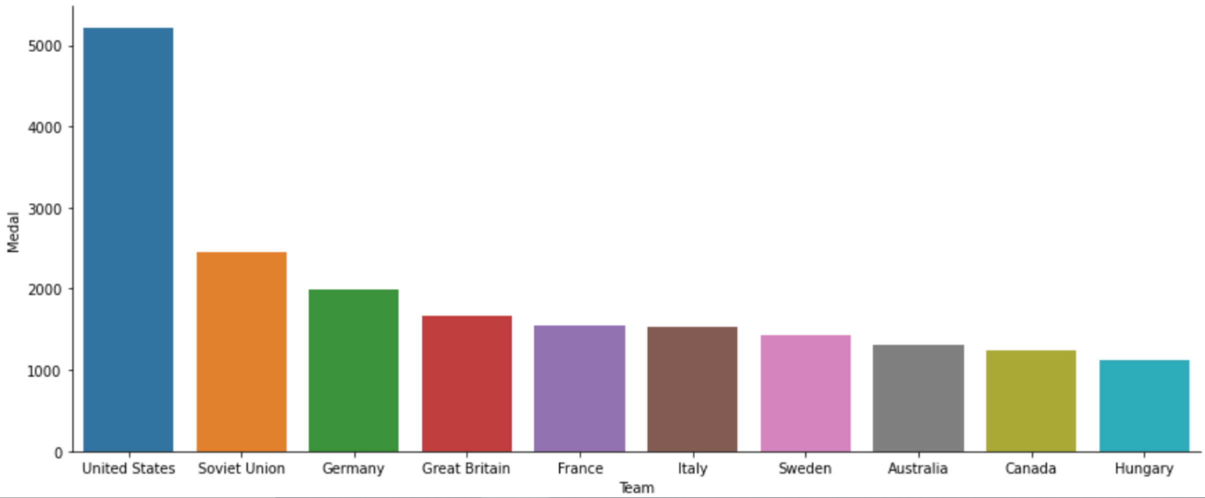
```
medal_rank=medal_rank.sort_values("Medal",ascending=False)
medal_rank.head(10).reset_index()
```

| | index | Team | Medal |
|---|---|---|---|
| 0 | 1095 | United States | 5219 |
| 1 | 976 | Soviet Union | 2451 |
| 2 | 398 | Germany | 1984 |
| 3 | 412 | Great Britain | 1673 |
| 4 | 361 | France | 1550 |
| 5 | 506 | Italy | 1527 |
| 6 | 1010 | Sweden | 1434 |
| 7 | 65 | Australia | 1306 |
| 8 | 173 | Canada | 1243 |
| 9 | 476 | Hungary | 1127 |

- **Plotting the table using seaborn**

```
medal=medal_rank.head(10)
sns.catplot(x="Team", y="Medal", kind="bar", data=medal,aspect=20/8.27)
```

```
<seaborn.axisgrid.FacetGrid at 0x17e564b7520>
```




- **Teams with lowest medal count**

```
medal=medal_rank.tail(10)
medal
##sns.catplot(x="Team", y="Medal", kind="bar", data=medal,aspect=20/8.27)
```

| | Team | Medal |
|---|---|---|
| 487 | India-1 | 0 |
| 488 | India-2 | 0 |
| 492 | Indonesia-2 | 0 |
| 493 | Inga-Lill XXXXIII | 0 |
| 494 | Ingegerd | 0 |
| 498 | Ireland-1 | 0 |
| 503 | Israel-1 | 0 |
| 504 | Israel-2 | 0 |
| 509 | Italy-3 | 0 |
| 1183 | rn-2 | 0 |

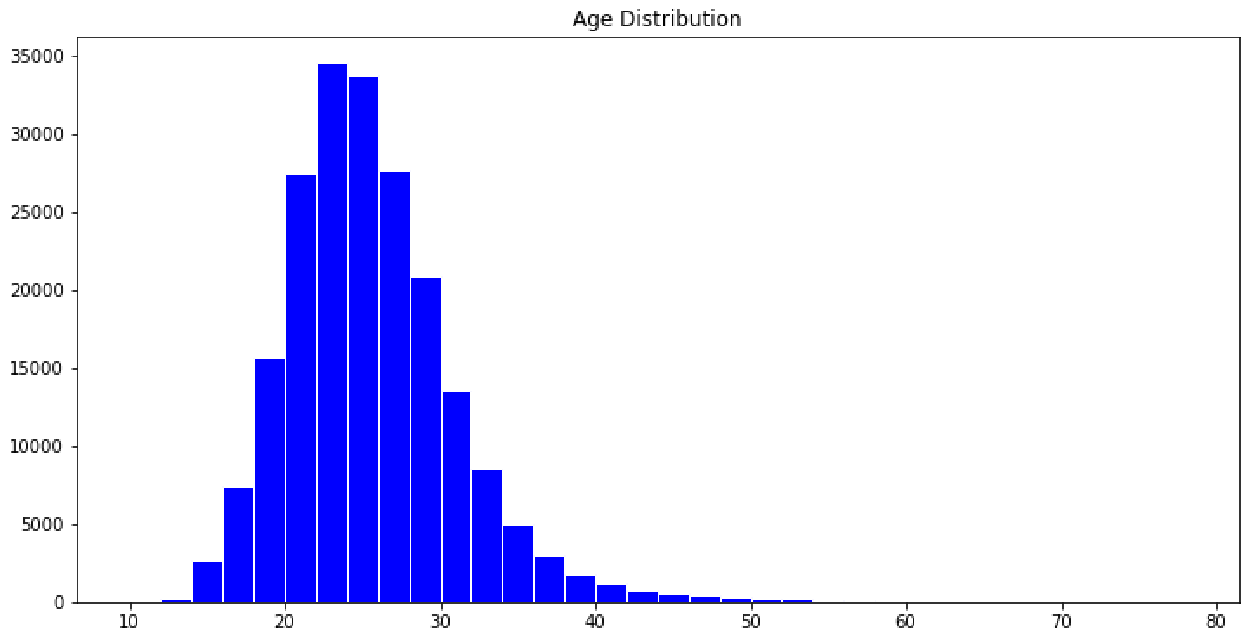
- **Total medals for India (These values count for individual players in team also)**

```
### Medals India
medal_rank.loc[medal_rank['Team'] == 'India']
```

| | Team | Medal |
|---|---|---|
| 249 | India | 96 |

- **Analysing for Age of the athletes**

```
### Analysis of age
plt.figure(figsize=(12, 6))
plt.tight_layout()
plt.title('Age Distribution')
plt.hist(df.Age,bins=np.arange(10,80,2),color='blue',edgecolor='white')
```




Here, we see that most participants are between age 20-30.

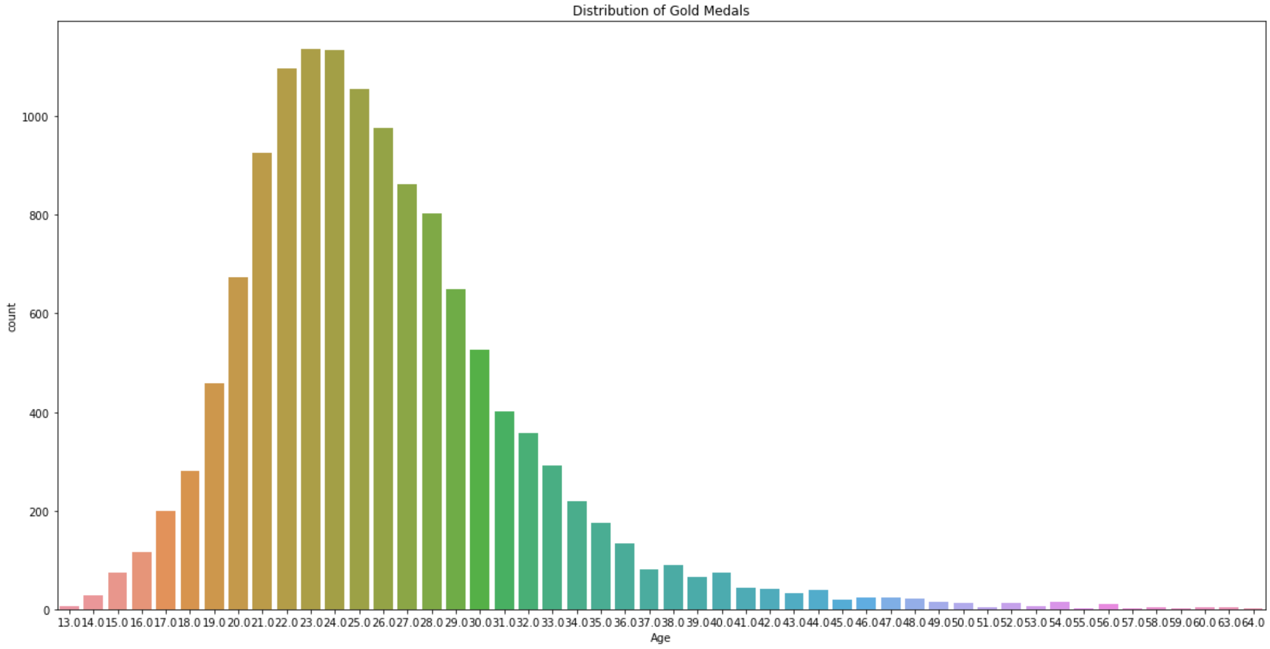
### We also see there are athletes who are above 50. Let's see the count:

```
df['ID'][df['Age'] > 50].count()
```

```
1938
```

- ## Distribution of gold with age

```
### Age distribution for gold medal
goldMedals = df[(df.Medal == 'Gold')]
goldMedals = goldMedals[np.isfinite(goldMedals['Age'])]
plt.figure(figsize=(20, 10))
plt.tight_layout()
sns.countplot(goldMedals['Age'])
plt.title('Distribution of Gold Medals')
```




- **Total teams**

```
## Number of Teams
count_team=len(pd.unique(df['Team']))
count_team
```

```
1184
```

There are 1184 total teams.

- **Medals for India information:**

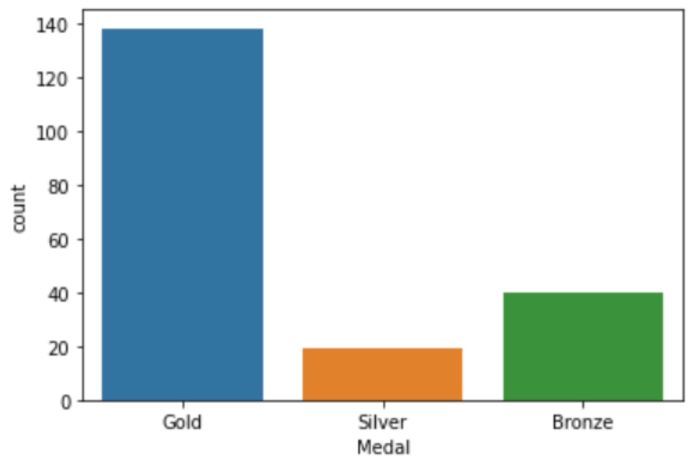
```
### medals for india

india=df.loc[(df1['Team'] == 'India') & (df['Medal'].notnull())]
india
```

| | ID | Name | Sex | Age | Height | Weight | Team | NOC | Games | Year | Season | City | Sport | Event | Medal | region | notes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **4736** | 2703 | Syed Mushtaq Ali | M | 22.0 | 165.0 | 61.0 | India | IND | 1964 Summer | 1964 | Summer | Tokyo | Hockey | Hockey Men's Hockey | Gold | India | NaN |
| **8192** | 4518 | Joseph Anthony "Joe" Antic | M | 29.0 | 168.0 | 59.0 | India | IND | 1960 Summer | 1960 | Summer | Roma | Hockey | Hockey Men's Hockey | Silver | India | NaN |
| **21208** | 11197 | Vasudevan Bhaskaran | M | 29.0 | 174.0 | 68.0 | India | IND | 1980 Summer | 1980 | Summer | Moskva | Hockey | Hockey Men's Hockey | Gold | India | NaN |
| **21815** | 11520 | Govinda Billimogaputtaswamy | M | 20.0 | 171.0 | 60.0 | India | IND | 1972 Summer | 1972 | Summer | Munich | Hockey | Hockey Men's Hockey | Bronze | India | NaN |
| **22004** | 11601 | Abhinav Bindra | M | 25.0 | 173.0 | 70.0 | India | IND | 2008 Summer | 2008 | Summer | Beijing | Shooting | Shooting Men's Air Rifle, 10 metres | Gold | India | NaN |

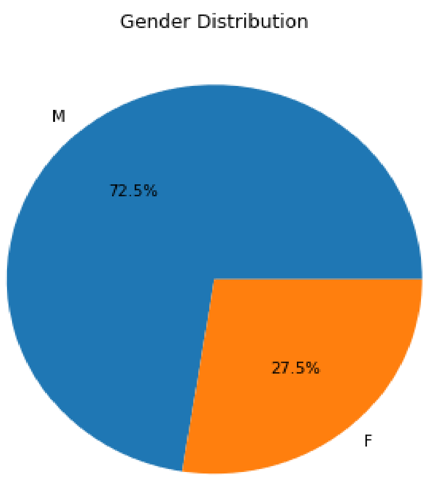
- **Plotting for Gold, Silver, Bronze:**

```
sns.countplot(india['Medal'])
```




- **Gender Distribution**

```
plt.figure(figsize=(12, 6))
plt.tight_layout()
plt.title('Gender Distribution')
plt.pie(gender_count,labels=gender_count.index,autopct='%1.1f%%')
```




- Medal won by female participants

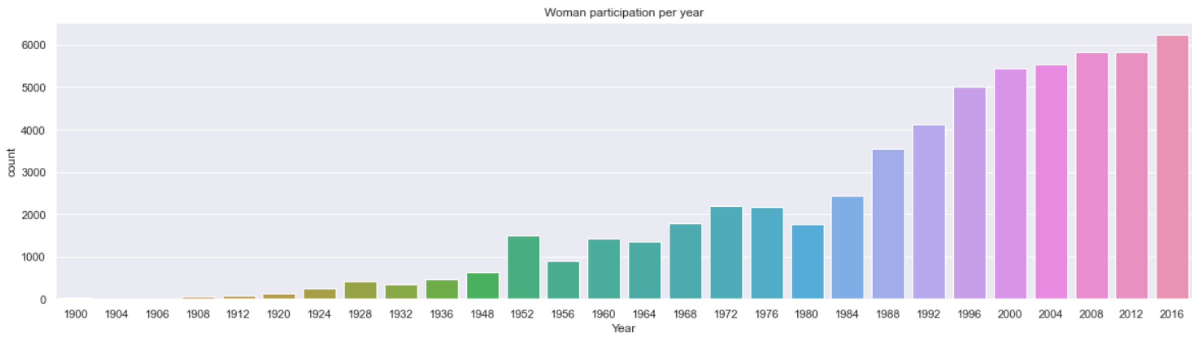
```python
### Medal won by female Participants
female_medal=df[(df.Sex=='F')]
female_medal.Medal.value_counts()
```

```
Bronze    3771
Gold      3747
Silver    3735
Name: Medal, dtype: int64
```

- **Female participation in each Summer Olympics**

```python
female=df[(df.Sex=='F') & (df.Season=='Summer')][['Sex','Year']]
female=female.groupby('Year').count().reset_index()
female.tail()
```

| | Year | Sex |
|---|---|---|
| **23** | 2000 | 5431 |
| **24** | 2004 | 5546 |
| **25** | 2008 | 5816 |
| **26** | 2012 | 5815 |
| **27** | 2016 | 6223 |




In the graph we can see that woman participation has increased in the recent years.

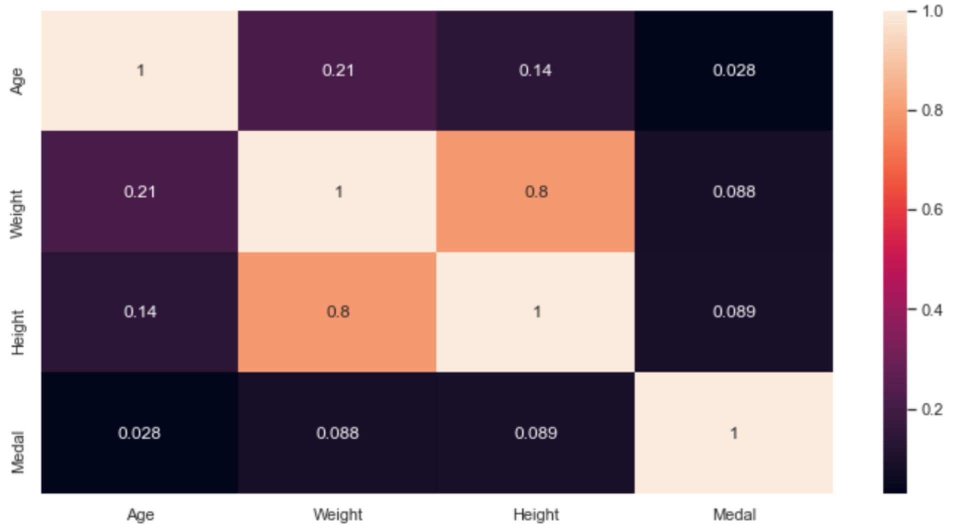
- **Correlation among the data**

- Plotting the heatmap

```
c=win.corr()
c
```

| | Age | Weight | Height | Medal |
|---|---|---|---|---|
| **Age** | 1.000000 | 0.211951 | 0.141736 | 0.028036 |
| **Weight** | 0.211951 | 1.000000 | 0.796652 | 0.088426 |
| **Height** | 0.141736 | 0.796652 | 1.000000 | 0.089117 |
| **Medal** | 0.028036 | 0.088426 | 0.089117 | 1.000000 |

```
plt.figure(figsize=(12,6))
sns.heatmap(c,annot=True)
plt.show()
```




## Machine learning model to Predict winning

- Creating a new data frame

  ## 1-Winning a medal
  ## 0- Losing

```
### Taking out the numerical Values using the table with non null values
win=df1[['Age','Weight','Height','Medal']]
win.loc[df1['Medal'].notnull(), 'Medal'] = 1
win.loc[df1['Medal'].isnull(), 'Medal'] = 0
win
```

| | Age | Weight | Height | Medal |
|---|---|---|---|---|
| **0** | 24.0 | 80.0 | 180.0 | 0 |
| **1** | 23.0 | 60.0 | 170.0 | 0 |
| **4** | 21.0 | 82.0 | 185.0 | 0 |
| **5** | 21.0 | 82.0 | 185.0 | 0 |
| **6** | 25.0 | 82.0 | 185.0 | 0 |
| **...** | ... | ... | ... | ... |
| **271111** | 29.0 | 89.0 | 179.0 | 0 |
| **271112** | 27.0 | 59.0 | 176.0 | 0 |
| **271113** | 27.0 | 59.0 | 176.0 | 0 |
| **271114** | 30.0 | 96.0 | 185.0 | 0 |
| **271115** | 34.0 | 96.0 | 185.0 | 0 |

205895 rows × 4 columns

```
win['Medal'].value_counts()
```

```
0     175723
1      30172
Name: Medal, dtype: int64
```

```
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
%matplotlib inline
```

```
X = np.asarray(win[['Age','Weight','Height']]).astype('int')
X[0:5]
```

```
array([[ 24,  80, 180],
       [ 23,  60, 170],
       [ 21,  82, 185],
       [ 21,  82, 185],
       [ 25,  82, 185]])
```

```
y = np.asarray(win['Medal']).astype('int')
y [0:5]
```

```
array([0, 0, 0, 0, 0])
```

```
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

```
array([[-0.19,  0.65,  0.44],
       [-0.38, -0.75, -0.51],
```

## normalizing the values

- **Logistic Regression**

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
yhat = LR.predict(X_test)
```

```
yhat
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

```
array([[0.85, 0.15],
       [0.83, 0.17],
       [0.85, 0.15],
       ...,
       [0.82, 0.18],
       [0.84, 0.16],
       [0.86, 0.14]])
```

- Let's try Root mean Squared error for error calculation

```
## root mean squared error
from sklearn.metrics import mean_squared_error

np.sqrt(mean_squared_error(y_test, yhat))
```

```
0.3851653093161678
```

- Plotting confusion matrix

```
from sklearn.metrics import classification_report, confusion_matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)
```

```
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
print(confusion_matrix(y_test, yhat, labels=[1,0]))
```

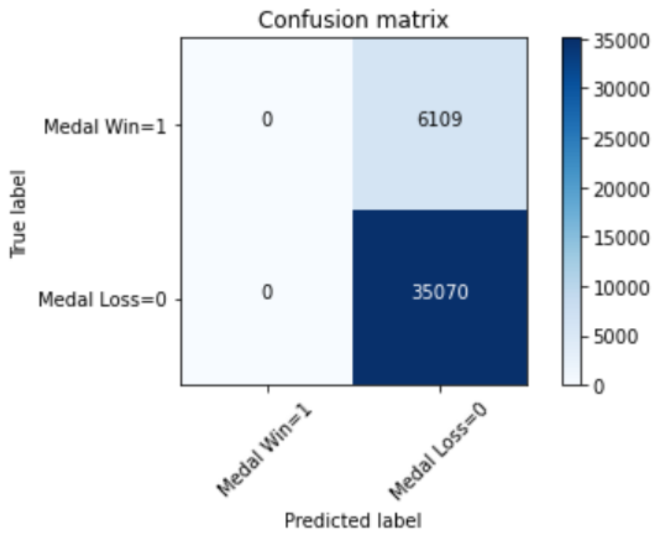
```
[[    0  6109]
 [    0 35070]]
```

We are plotting a confusion matrix to evaluate the performance.

```
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['Medal Win=1','Medal Loss=0'],normalize= False,  title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[    0  6109]
 [    0 35070]]
```




Based on the count of each section, we can calculate precision and recall of each label:

- **Precision** is a measure of the accuracy provided that a class label has been predicted.
- **Recall** is the true positive rate.

So, we can calculate the precision and recall of each class.

**F1 score:** Now we are in the position to calculate the F1 scores for each label based on the precision and recall of that label.

The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. It is a good way to show that a classifier has a good value for both recall and precision.

```
print (classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92     35070
           1       0.00      0.00      0.00      6109

    accuracy                           0.85     41179
   macro avg       0.43      0.50      0.46     41179
weighted avg       0.73      0.85      0.78     41179
```

# We see here accuracy of the model is 0.85

- ## Decision Tree Classification

```
from sklearn.tree import DecisionTreeClassifier
```

```
## We will first create an instance of the DecisionTreeClassifier called Tree
Tree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
Tree.fit(X_train,y_train)
## predict the value
predTree =Tree.predict(X_test)
```

```
predTree[0:10]
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
## defining the accuracy of the model
from sklearn import metrics
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_test, predTree))
```

```
DecisionTrees's Accuracy:  0.8516476844993808
```

- Let's try Root mean Squared error for error calculation

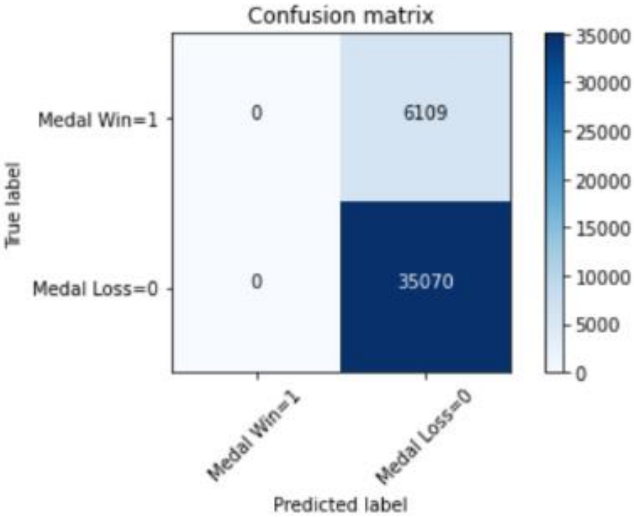
```
np.sqrt(mean_squared_error(y_test, predTree))
```

```
0.3851653093161678
```

```
cnf_matrix = confusion_matrix(y_test, predTree, labels=[1,0])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['Medal Win=1','Medal Loss=0'],normalize= False,  title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[    0  6109]
 [    0 35070]]
```




```
print (classification_report(y_test, predTree))
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92     35070
           1       0.00      0.00      0.00      6109

    accuracy                           0.85     41179
   macro avg       0.43      0.50      0.46     41179
weighted avg       0.73      0.85      0.78     41179
```

**# We see here accuracy of the model is 0.85**

- **K-Nearest Neighbour**

Import the files and predict the values and then calculate root mean squared error

```
### k nearest neighbour

from sklearn.neighbors import KNeighborsClassifier
k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat = neigh.predict(X_test)
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.8548896282085529
Test set Accuracy:  0.8427596590495156
```

```
##RMSE
np.sqrt(mean_squared_error(y_test, yhat))
```

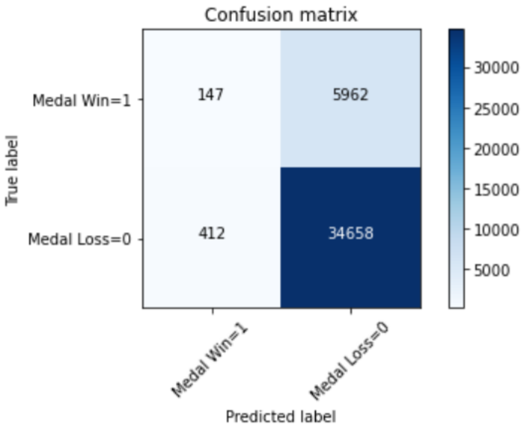
```
0.39343059676883585
```

- ## Confusion matrix

```
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['Medal Win=1','Medal Loss=0'],normalize= False,  title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[  147  5962]
 [  412 34658]]
```




- ## Calculate the accuracy

# We see here accuracy of the model is 0.85

```
print (classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

           0       0.85      0.99      0.92     35070
           1       0.26      0.02      0.04      6109

    accuracy                           0.85     41179
   macro avg       0.56      0.51      0.48     41179
weighted avg       0.77      0.85      0.79     41179
```

K in KNN, is the number of nearest neighbours to examine. It is supposed to be specified by the user. So, how can we choose right value for K? The general solution is to reserve a part of your data for testing the accuracy of the model. Then choose k =1, use the training part for modelling, and calculate the accuracy of prediction using all samples in your test set. Repeat this process, increasing the k, and see which k is the best for your model.

We can calculate the accuracy of KNN for different values of k.

```
## We can calculate the accuracy of KNN for different values of k.
```

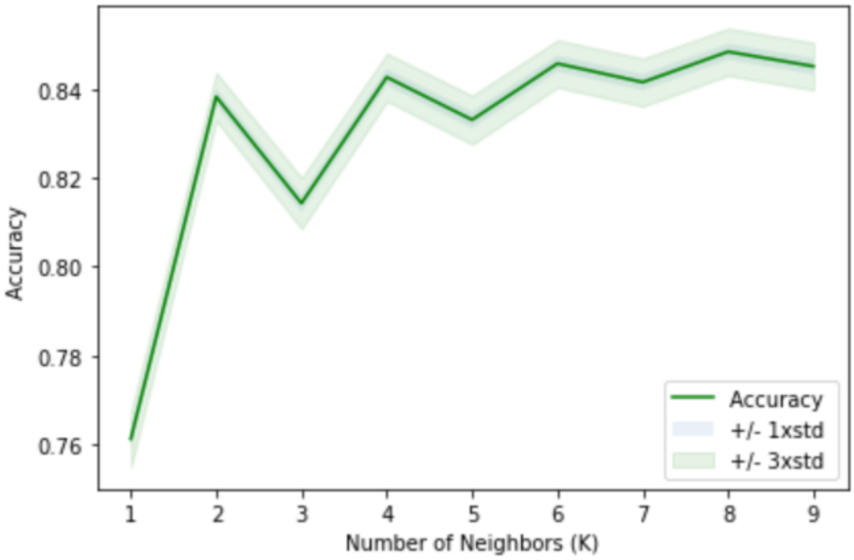
```
Ks=10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))

for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.fill_between(range(1,Ks),mean_acc - 3 * std_acc,mean_acc + 3 * std_acc, alpha=0.10,color="green")
plt.legend(('Accuracy ', '+/- 1xstd','+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```




## Accuracy:

- Logistic Regression: 0.85
- Decision Tree: 0.85
- K- Nearest Neighbour: 0.85

## Conclusion:

- USA has the most medals
- We see that most participants are between age 20-30.
- Female participation has increased over the years.
- Most gold is won by participants that are between age 20-30.
- People above age 50 are also wining gold medal
- Wining doesn’t depend much on age, weight or height.
- We see here that the accuracy shown by all the three models are equal as winning a medal depends on many factors and we have insufficient data for that.
- We need more data to accurately predict the win.