Practical 1

Problem Statement - To apply Preprocessing techniques on raw data set
To perform E.D.A.

Libraries used - numpy
Pandas
Matplot
seaborn
SKlearn

data set - titanic

Plots used - histograms & Boxplots

train to test split ratio :> 80:20

Practical No.1

Data Science and Visualization (Honors Course)

Name:Manjunath GB

PRN: 72018269H

Class: TE ENTC 'B'

In this practical we will access an open source dataset 'titanic.csv' and apply pre-processing techniques on the raw dataset.

In [1]:

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

We will now check the current version of all the packages which we imported.

In [2]:

```
pd.__version__
```

Out[2]:

```
'1.2.4'
```

In [3]:

```
np.__version__
```

Out[3]:

```
'1.20.1'
```

In [4]:

```
sns.__version__
```

Out[4]:

```
'0.11.1'
```

We will now get the datasets which are already inbuilt in the packages.

In [5]:

```
sns.get_dataset_names()
```

Out[5]:

```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'exercise',
 'flights',
 'fmri']
```

```
 'gammas',
 'geyser',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'taxis',
 'tips',
 'titanic']
```

There are various ways to import a dataset which are as follows:

In [6]:

```
dataset = sns.load_dataset('titanic')
```

In [7]:

```
dataset
```

Out[7]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no

891 rows × 15 columns

In [8]:

```
df = pd.read_csv('https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff/titanic.csv')
```

In [9]:

```
df
```

Out[9]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500
...	...	...	...	...	...	...	...	...

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
882	0	2	Rev. Juozas Montvila	male	27.0	0	0	13.0000
883	1	1	Miss. Margaret Edith Graham	female	19.0	0	0	30.0000
884	0	3	Miss. Catherine Helen Johnston	female	7.0	1	2	23.4500
885	1	1	Mr. Karl Howell Behr	male	26.0	0	0	30.0000
886	0	3	Mr. Patrick Dooley	male	32.0	0	0	7.7500

887 rows × 8 columns

We will now perform certain pre processing operations on our dataset.

In [11]:

```
df.columns  #The title of all the columns in the dataset.
```

Out[11]:

```
Index(['Survived', 'Pclass', 'Name', 'Sex', 'Age', 'Siblings/Spouses Aboard',
       'Parents/Children Aboard', 'Fare'],
      dtype='object')
```

In [12]:

```
df.shape
```

Out[12]:

```
(887, 8)
```

In [13]:

```
dataset.shape
```

Out[13]:

```
(891, 15)
```

In [14]:

```
dataset.columns
```

Out[14]:

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
       'alive', 'alone'],
      dtype='object')
```

In [16]:

```
df.head() #the .head() function returns the first five rows of dataset by default.
```

Out[16]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500

In [17]:

```
dataset.head()
```

dataset.head()

Out[17]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alo
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	Fa
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	Fa
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	Tı
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	Fa
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	Tı

In [19]:

```
df.tail() #the .tail() function returns the last five rows of dataset by default.
```

Out[19]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
882	0	2	Rev. Juozas Montvila	male	27.0	0	0	13.00
883	1	1	Miss. Margaret Edith Graham	female	19.0	0	0	30.00
884	0	3	Miss. Catherine Helen Johnston	female	7.0	1	2	23.45
885	1	1	Mr. Karl Howell Behr	male	26.0	0	0	30.00
886	0	3	Mr. Patrick Dooley	male	32.0	0	0	7.75

In [20]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 887 entries, 0 to 886
Data columns (total 8 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Survived                 887 non-null    int64  
 1   Pclass                   887 non-null    int64  
 2   Name                     887 non-null    object 
 3   Sex                      887 non-null    object 
 4   Age                      887 non-null    float64
 5   Siblings/Spouses Aboard  887 non-null    int64  
 6   Parents/Children Aboard  887 non-null    int64  
 7   Fare                     887 non-null    float64
dtypes: float64(2), int64(4), object(2)
memory usage: 55.6+ KB
```

In [21]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   survived     891 non-null    int64   
 1   pclass       891 non-null    int64   
 2   sex          891 non-null    object  
 3   age          714 non-null    float64 
 4   sibsp        891 non-null    int64   
 5   parch        891 non-null    int64   
 6   fare         891 non-null    float64 
 7   embarked     889 non-null    object  
 8   class        891 non-null    category
```

```
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

In [22]:

```
dataset.describe()
```

Out[22]:

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [23]:

```
df.describe()
```

Out[23]:

	Survived	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
count	887.000000	887.000000	887.000000	887.000000	887.000000	887.00000
mean	0.385569	2.305524	29.471443	0.525366	0.383315	32.30542
std	0.487004	0.836662	14.121908	1.104669	0.807466	49.78204
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.00000
25%	0.000000	2.000000	20.250000	0.000000	0.000000	7.92500
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.45420
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.13750
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.32920

In [24]:

```
df.count()
```

Out[24]:

```
Survived                    887
Pclass                      887
Name                        887
Sex                         887
Age                         887
Siblings/Spouses Aboard     887
Parents/Children Aboard     887
Fare                        887
dtype: int64
```

In [25]:

```
dataset.count()
```

Out[25]:

```
survived       891
pclass         891
sex            891
age            714
sibsp          891
parch          891
fare           891
embarked       889
class          891
who            891
adult_male     891
deck           203
embark_town    889
alive          891
alone          891
dtype: int64
```

In [26]:

```
dataset.isnull().sum()
```

Out[26]:

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

In [27]:

```
dataset = dataset.drop('deck', axis = 1)
```

In [28]:

```
dataset.isnull().sum()
```

Out[28]:

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
embark_town      2
alive            0
alone            0
dtype: int64
```

In [29]:

```
dataset['age'] = dataset['age'].fillna(dataset['age'].median())
```

In [30]:

```
dataset.isnull().sum()
```

Out[30]:

```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       2
class          0
who            0
adult_male     0
embark_town    2
alive          0
alone          0
dtype: int64
```

In [31]:

```
dataset['embarked'].mode()[0]
```

Out[31]:

```
'S'
```

In [32]:

```
dataset['embark_town'].mode()[0]
```

Out[32]:

```
'Southampton'
```

In [33]:

```
dataset['embarked'] = dataset['embarked'].fillna(
    dataset['embarked'].mode()[0])
```

In [34]:

```
dataset['embark_town'] = dataset['embark_town'].fillna(
    dataset['embark_town'].mode()[0])
```

In [35]:

```
dataset.isnull().sum()
```

Out[35]:

```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
embark_town    0
alive          0
alone          0
dtype: int64
```

In [36]:

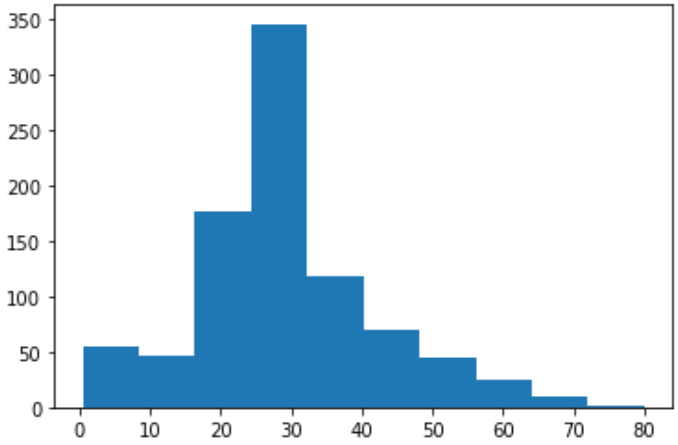
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          891 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     891 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  embark_town  891 non-null    object
 12  alive        891 non-null    object
 13  alone        891 non-null    bool
dtypes: bool(2), category(1), float64(2), int64(4), object(5)
memory usage: 79.4+ KB
```

Visualization of dataset

In [37]:

```
plt.hist(dataset['age']);
```

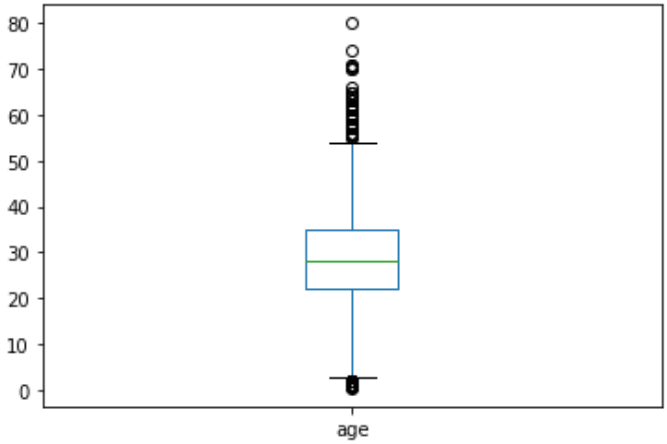


In [38]:

```
dataset['age'].plot(kind='box')
```

Out[38]:

```
<AxesSubplot:>
```

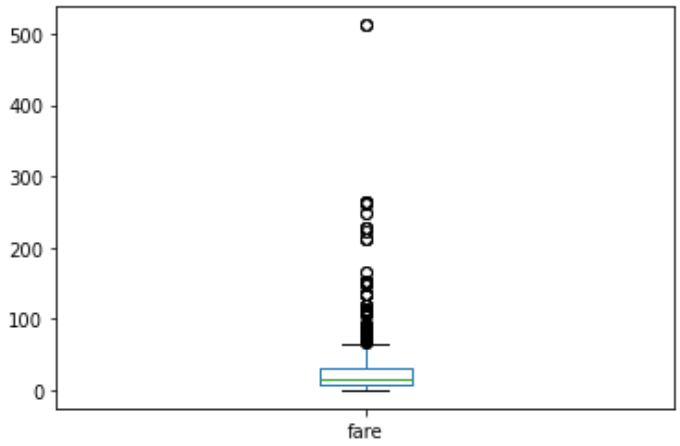


In [39]:

```
dataset['fare'].plot(kind='box')
```

Out[39]:

<AxesSubplot:>



In [40]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   survived     891 non-null    int64   
 1   pclass       891 non-null    int64   
 2   sex          891 non-null    object  
 3   age          891 non-null    float64 
 4   sibsp        891 non-null    int64   
 5   parch        891 non-null    int64   
 6   fare         891 non-null    float64 
 7   embarked     891 non-null    object  
 8   class        891 non-null    category
 9   who          891 non-null    object  
 10  adult_male   891 non-null    bool    
 11  embark_town  891 non-null    object  
 12  alive        891 non-null    object  
 13  alone        891 non-null    bool    
dtypes: bool(2), category(1), float64(2), int64(4), object(5)
memory usage: 79.4+ KB
```

In [41]:

```
pd.get_dummies(dataset).head()
```

Out[41]:

	survived	pclass	age	sibsp	parch	fare	adult_male	alone	sex_female	sex_male	...	class_Second	class_Third	who
0	0	3	22.0	1	0	7.2500	True	False	0	1	...	0	1	
1	1	1	38.0	1	0	71.2833	False	False	1	0	...	0	0	
2	1	3	26.0	0	0	7.9250	False	True	1	0	...	0	1	
3	1	1	35.0	1	0	53.1000	False	False	1	0	...	0	0	
4	0	3	35.0	0	0	8.0500	True	True	0	1	...	0	1	

5 rows × 24 columns

Training our Dataset

Importing the required library.

In [42]:

```
from sklearn.model_selection import train_test_split
```

In [43]:

```
train, test = train_test_split(dataset,test_size=0.20)
```

In [44]:

```
len(dataset)
```

Out[44]:

```
891
```

In [45]:

```
len(train)
```

Out[45]:

```
712
```

In [46]:

```
len(test)
```

Out[46]:

```
179
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Practical -2

Problem statement - Prediction of Survival based on :-
a) age
b) pclass
c) sex and pclass

Libraries used - a) Numpy
b) seaborn
c) Pandas

dataset - titanic
Packages used - collection, label encoder
Plots used - Bar chart

Metrics used - accuracy score

Model - Naive Bayes

$$P(H/E) = \frac{P(E/H) * P(H)}{P(E)}$$

Train & test split ratio - 75:25
Train accuracy - 78.85%
Test " - 78.02%

Practical No.2

Data Science and Visualization (Honors Course)

Name:Manjunath GB

PRN: 72018269H

Class: TE ENTC 'B'

In this practical we will predict the probability os survival on the basis of age,gender and passenger class in titanic dataset.

Firstly,we will import certain libraries.

In [3]:

```
import numpy as np #This library provide support to the arrays
import seaborn as sns #Library for data visualization
import pandas as pd #for data manipulation
```

In [4]:

```
ds = sns.load_dataset('titanic') #The titanic dataset is already built in seaborn library
```

In [5]:

```
ds.head(10) #Displaying the first 10 rows
```

Out[5]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no
5	0	3	male	NaN	0	0	8.4583	Q	Third	man	True	NaN	Queenstown	no
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	E	Southampton	no
7	0	3	male	2.0	3	1	21.0750	S	Third	child	False	NaN	Southampton	no
8	1	3	female	27.0	0	2	11.1333	S	Third	woman	False	NaN	Southampton	yes
9	1	2	female	14.0	1	0	30.0708	C	Second	child	False	NaN	Cherbourg	yes

In [6]:

```
len(ds) #To find the number of entries
```

Out[6]:

```
891
```

Data Cleaning

In [8]:

```
ds['age'] = ds['age'].fillna(ds['age'].median())
#The Null or Not Available values in the Age Column will be replaced by the median values
```

In [10]:

```
x = ds['age'].values  #x is Input
y = ds['survived']    #y is the output
# We will predicting the number of survived persons(y) on the basis of their age(x).
```

In [11]:

```
x.shape #checking the shape
```

Out[11]:

```
(891,)
```

In [13]:

```
x=x.reshape(-1,1) #Reshaping the column.
# -1 indicates to keep 891 as it is and 1 indicates to add 1 to the shaoe of column.
```

In [14]:

```
x.shape
```

Out[14]:

```
(891, 1)
```

Data Cleaning part has been completed.

Splitting the data for Testing and Training

In [15]:

```
from sklearn.model_selection import train_test_split
```

In [16]:

```
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=0,test_size=0.25)
#test_size indicates how many samples should be present for test dataset.
#Therefore,25% of the entries will got to test dataset and the rest 75% will go to train
dataset.
```

In [17]:

```
len(x_train)
```

Out[17]:

```
668
```

In [18]:

```
len(y_train)
```

Out[18]:

```
668
```

In [19]:

```
len(x_test)
```

Out[19]:

```
223
```

In [21]:

In [21]:

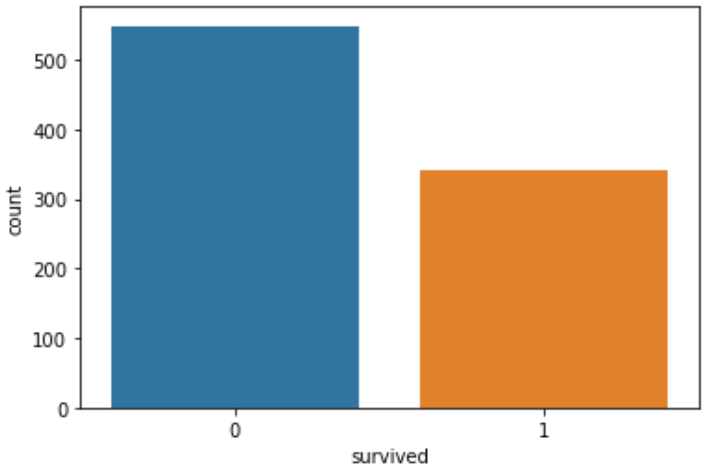
```
from collections import Counter
```

In [22]:

```
Counter(y)
sns.countplot(x=y)
```

Out[22]:

```
<AxesSubplot:xlabel='survived', ylabel='count'>
```



The counter will count the number of individuals who have survived and those who have not survived.

Making the prediction using GaussianNB

In [24]:

```
from sklearn.naive_bayes import GaussianNB
```

In [25]:

```
model = GaussianNB()
```

In [26]:

```
model.fit(x_train,y_train) #Fitting the model
```

Out[26]:

```
GaussianNB()
```

In [28]:

```
y_pred = model.predict_proba(x_test)  #Storing the predicted values in y_pred
```

In [29]:

```
y_pred
```

Out[29]:

```
array([[0.62876821, 0.37123179],
       [0.62876821, 0.37123179],
       [0.51657653, 0.48342347],
       [0.62876821, 0.37123179],
       [0.63148867, 0.36851133],
       [0.62876821, 0.37123179],
       [0.64689984, 0.35310016],
       [0.63625703, 0.36374297],
       [0.6192395 , 0.3807605 ],
       [0.62876821, 0.37123179],
       [0.62264556, 0.37735444],
       [0.64689984, 0.35310016]
```

[0.61009964, 0.38990036],
[0.62876821, 0.37123179],
[0.51657653, 0.48342347],
[0.61560095, 0.38439905],
[0.56600603, 0.43399397],
[0.61172787, 0.38827213],
[0.59385049, 0.40614951],
[0.64314813, 0.35685187],
[0.45943048, 0.54056952],
[0.58877545, 0.41122455],
[0.60761825, 0.39238175],
[0.62876821, 0.37123179],
[0.62876821, 0.37123179],
[0.60761825, 0.39238175],
[0.64689984, 0.35310016],
[0.63830829, 0.36169171],
[0.60761825, 0.39238175],
[0.6192395 , 0.3807605],
[0.47403058, 0.52596942],
[0.64013966, 0.35986034],
[0.63835894, 0.36164106],
[0.62876821, 0.37123179],
[0.62876821, 0.37123179],
[0.63148867, 0.36851133],
[0.63830829, 0.36169171],
[0.64658851, 0.35341149],
[0.62876821, 0.37123179],
[0.6192395 , 0.3807605],
[0.62884219, 0.37115781],
[0.60772869, 0.39227131],
[0.6192395 , 0.3807605],
[0.62876821, 0.37123179],
[0.56600603, 0.43399397],
[0.64314813, 0.35685187],
[0.62876821, 0.37123179],
[0.62876821, 0.37123179],
[0.58877545, 0.41122455],
[0.64657737, 0.35342263],
[0.6340466 , 0.3659534],
[0.62876821, 0.37123179],
[0.61172787, 0.38827213],
[0.49173773, 0.50826227],
[0.59385049, 0.40614951],
[0.62876821, 0.37123179],
[0.61560095, 0.38439905],
[0.59880446, 0.40119554],
[0.5463121 , 0.4536879],
[0.53924847, 0.46075153],
[0.62876821, 0.37123179],
[0.60761825, 0.39238175],
[0.63148867, 0.36851133],
[0.62884219, 0.37115781],
[0.62876821, 0.37123179],
[0.62582114, 0.37417886],
[0.63830829, 0.36169171],
[0.64018459, 0.35981541],
[0.62876821, 0.37123179],
[0.47403058, 0.52596942],
[0.59385049, 0.40614951],
[0.61560095, 0.38439905],
[0.64179176, 0.35820824],
[0.62876821, 0.37123179],
[0.62876821, 0.37123179],
[0.61744941, 0.38255059],
[0.63398437, 0.36601563],
[0.59385049, 0.40614951],
[0.64689984, 0.35310016],
[0.64175253, 0.35824747],
[0.62876821, 0.37123179],
[0.62876821, 0.37123179],
[0.53193679, 0.46806321],
[0.62264556, 0.37735444],
[0.57788703, 0.42211297],

[0.64435549, 0.35564451],
[0.6032701 , 0.3967299],
[0.64690541, 0.35309459],
[0.59397985, 0.40602015],
[0.64175253, 0.35824747],
[0.58877545, 0.41122455],
[0.62876821, 0.37123179],
[0.59385049, 0.40614951],
[0.48299524, 0.51700476],
[0.64605821, 0.35394179],
[0.53193679, 0.46806321],
[0.62876821, 0.37123179],
[0.61560095, 0.38439905],
[0.57788703, 0.42211297],
[0.60772869, 0.39227131],
[0.62582114, 0.37417886],
[0.56600603, 0.43399397],
[0.62884219, 0.37115781],
[0.59385049, 0.40614951],
[0.58877545, 0.41122455],
[0.63148867, 0.36851133],
[0.64314813, 0.35685187],
[0.62876821, 0.37123179],
[0.64700915, 0.35299085],
[0.6192395 , 0.3807605],
[0.62876821, 0.37123179],
[0.64179176, 0.35820824],
[0.60761825, 0.39238175],
[0.63830829, 0.36169171],
[0.62876821, 0.37123179],
[0.64432756, 0.35567244],
[0.62876821, 0.37123179],
[0.61560095, 0.38439905],
[0.58877545, 0.41122455],
[0.64689984, 0.35310016],
[0.64531408, 0.35468592],
[0.62876821, 0.37123179],
[0.62876821, 0.37123179],
[0.59868146, 0.40131854],
[0.64604149, 0.35395851],
[0.64013966, 0.35986034],
[0.61569902, 0.38430098],
[0.62876821, 0.37123179],
[0.6192395 , 0.3807605],
[0.62876821, 0.37123179],
[0.63631345, 0.36368655],
[0.6032701 , 0.3967299],
[0.6032701 , 0.3967299],
[0.6192395 , 0.3807605],
[0.59868146, 0.40131854],
[0.58891121, 0.41108879],
[0.6032701 , 0.3967299],
[0.62876821, 0.37123179],
[0.64689984, 0.35310016],
[0.64605821, 0.35394179],
[0.61172787, 0.38827213],
[0.62876821, 0.37123179],
[0.64013966, 0.35986034],
[0.62876821, 0.37123179],
[0.58877545, 0.41122455],
[0.62876821, 0.37123179],
[0.61560095, 0.38439905],
[0.63398437, 0.36601563],
[0.64690541, 0.35309459],
[0.63835894, 0.36164106],
[0.62876821, 0.37123179],
[0.64531408, 0.35468592],
[0.63398437, 0.36601563],
[0.64529175, 0.35470825],
[0.63148867, 0.36851133],
[0.58345477, 0.41654523],
[0.62264556, 0.37735444],

```
[0.62264556, 0.37735444],
[0.64531408, 0.35468592],
[0.59880446, 0.40119554],
[0.62876821, 0.37123179],
[0.46252279, 0.53747721],
[0.62876821, 0.37123179],
[0.59868146, 0.40131854],
[0.62884219, 0.37115781],
[0.64013966, 0.35986034],
[0.57207109, 0.42792891],
[0.6192395 , 0.3807605 ],
[0.63631345, 0.36368655],
[0.60761825, 0.39238175],
[0.61560095, 0.38439905],
[0.58345477, 0.41654523],
[0.63631345, 0.36368655],
[0.64604149, 0.35395851],
[0.62876821, 0.37123179],
[0.62876821, 0.37123179],
[0.55969127, 0.44030873],
[0.62876821, 0.37123179],
[0.63625703, 0.36374297],
[0.61172787, 0.38827213],
[0.59385049, 0.40614951],
[0.62876821, 0.37123179],
[0.63625703, 0.36374297],
[0.63625703, 0.36374297],
[0.59868146, 0.40131854],
[0.6032701 , 0.3967299 ],
[0.64486164, 0.35513836],
[0.60761825, 0.39238175],
[0.62876821, 0.37123179],
[0.62876821, 0.37123179],
[0.62264556, 0.37735444],
[0.6192395 , 0.3807605 ],
[0.6032701 , 0.3967299 ],
[0.63625703, 0.36374297],
[0.57207109, 0.42792891],
[0.62876821, 0.37123179],
[0.62876821, 0.37123179],
[0.58359697, 0.41640303],
[0.62876821, 0.37123179],
[0.46485056, 0.53514944],
[0.64175253, 0.35824747],
[0.64018459, 0.35981541],
[0.58877545, 0.41122455],
[0.62876821, 0.37123179],
[0.55330133, 0.44669867],
[0.56600603, 0.43399397],
[0.59385049, 0.40614951],
[0.63398437, 0.36601563],
[0.63625703, 0.36374297],
[0.63830829, 0.36169171],
[0.57788703, 0.42211297],
[0.63835894, 0.36164106],
[0.61560095, 0.38439905],
[0.62273148, 0.37726852],
[0.51657653, 0.48342347],
[0.53193679, 0.46806321],
[0.64013966, 0.35986034],
[0.59385049, 0.40614951],
[0.63925137, 0.36074863],
[0.46485056, 0.53514944],
[0.64531408, 0.35468592],
[0.62876821, 0.37123179],
[0.59385049, 0.40614951],
[0.6032701 , 0.3967299 ],
[0.49173773, 0.50826227]])
```

The probability values are displayed.We need to convert them into binary values of 0 and 1.

In [31]:

In [31]:

```
y_pred=model.predict(x_test)
```

In [32]:

```
y_pred
```

Out[32]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1], dtype=int64)
```

In [33]:

```
y_test
```

Out[33]:

```
495    0
648    0
278    0
31     1
255    1
      ..
167    0
306    1
379    0
742    1
10     1
Name: survived, Length: 223, dtype: int64
```

Determing the accuracy

Accuracy is given by (True Positive+True Negative)/Total no. of entries

In [34]:

```
from sklearn.metrics import accuracy_score
```

In [35]:

```
accuracy_score(y_test, y_pred)
```

Out[35]:

```
0.6547085201793722
```

The accuracy of the model is 65.47%.

Hence we have predicted the survival on the basis of age.

Predicting the survival on the basis of pclass

In [36]:

```
x=ds['pclass'].values
y=ds['survived']
```

In [37]:

```
x=x.reshape(-1,1)
```

In [38]:

```
x.shape
```

Out[38]:

```
(891, 1)
```

In [40]:

```
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=0,test_size=0.25)
#Updating all the values once again
```

In [41]:

```
model = GaussianNB()
```

In [42]:

```
model.fit(x_train,y_train)
```

Out[42]:

```
GaussianNB()
```

In [43]:

```
model.predict_proba(x_test)
```

Out[43]:

```
array([[0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.29957107, 0.70042893],
       [0.29957107, 0.70042893],
       [0.29957107, 0.70042893],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.58864036, 0.41135964],
       [0.7559945 , 0.2440055 ],
       [0.58864036, 0.41135964],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.58864036, 0.41135964],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.58864036, 0.41135964],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.58864036, 0.41135964],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.58864036, 0.41135964],
       [0.7559945 , 0.2440055 ],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.58864036, 0.41135964],
```

```
[0.58864036, 0.41135964],
[0.58864036, 0.41135964],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.58864036, 0.41135964],
[0.29957107, 0.70042893],
[0.29957107, 0.70042893],
[0.29957107, 0.70042893],
[0.58864036, 0.41135964],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.58864036, 0.41135964],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.58864036, 0.41135964],
[0.58864036, 0.41135964],
[0.29957107, 0.70042893],
[0.58864036, 0.41135964],
[0.29957107, 0.70042893],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.29957107, 0.70042893],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.29957107, 0.70042893],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.29957107, 0.70042893],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.7559945 , 0.2440055 ],
```

```
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.58864036, 0.41135964],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.58864036, 0.41135964],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.58864036, 0.41135964],
[0.58864036, 0.41135964],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.58864036, 0.41135964],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.58864036, 0.41135964],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.29957107, 0.70042893],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.7559945 , 0.2440055 ],
[0.58864036, 0.41135964],
[0.58864036, 0.41135964],
```

```
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.58864036, 0.41135964],
       [0.7559945 , 0.2440055 ],
       [0.58864036, 0.41135964],
       [0.7559945 , 0.2440055 ],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.58864036, 0.41135964],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.58864036, 0.41135964],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.58864036, 0.41135964],
       [0.7559945 , 0.2440055 ],
       [0.58864036, 0.41135964],
       [0.7559945 , 0.2440055 ],
       [0.7559945 , 0.2440055 ],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ],
       [0.29957107, 0.70042893],
       [0.7559945 , 0.2440055 ]])
```

In [44]:

```
y_pred=model.predict(x_test)
```

Determining the Accuracy

In [45]:

```
accuracy_score(y_test, y_pred)
```

Out[45]:

```
0.7085201793721974
```

The accuracy score is 70.85% compared to previous attribute.Hence,we can say that pclass attribute gives more accuarcy.

We will now use two columns and determine the accuracy

In [46]:

```
x = ds[['sex','pclass']]
y = ds['survived']
```

In [47]:

```
x['sex'].head()
```

Out[47]:

```
0      male
1    female
2    female
3    female
4      male
Name: sex, dtype: object
```

We will encode the entries in a certain format.

In [48]:

```
from sklearn.preprocessing import LabelEncoder
```

In [49]:

```
enc=LabelEncoder()
```

In [50]:

```
x['sex']=enc.fit_transform(x['sex']) #Converting the strings into number
```

```
<ipython-input-50-3ffee5c86835>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  x['sex']=enc.fit_transform(x['sex'])
```

In [52]:

```
x['sex'].head() #male-1 female-0
```

Out[52]:

```
0    1
1    0
2    0
3    0
4    1
Name: sex, dtype: int32
```

In [53]:

```
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=0,test_size=0.25)
```

In [54]:

```
model=GaussianNB()
```

In [55]:

```
model.fit(x_train,y_train)
```

Out[55]:

```
GaussianNB()
```

In [56]:

```
model.predict_proba(x_test)
```

Out[56]:

```
array([[0.91599965, 0.08400035],
       [0.91599965, 0.08400035],
       [0.91599965, 0.08400035],
       [0.03226176, 0.96773824],
       [0.19452137, 0.80547863],
```

```
[0.60085059, 0.39914941],
[0.03226176, 0.96773824],
[0.03226176, 0.96773824],
[0.60085059, 0.39914941],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.1003456 , 0.8996544 ],
[0.91599965, 0.08400035],
[0.1003456 , 0.8996544 ],
[0.03226176, 0.96773824],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.83433941, 0.16566059],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.83433941, 0.16566059],
[0.03226176, 0.96773824],
[0.91599965, 0.08400035],
[0.60085059, 0.39914941],
[0.19452137, 0.80547863],
[0.03226176, 0.96773824],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.1003456 , 0.8996544 ],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.1003456 , 0.8996544 ],
[0.91599965, 0.08400035],
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.83433941, 0.16566059],
[0.83433941, 0.16566059],
[0.83433941, 0.16566059],
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.03226176, 0.96773824],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.03226176, 0.96773824],
[0.83433941, 0.16566059],
[0.60085059, 0.39914941],
[0.60085059, 0.39914941],
[0.60085059, 0.39914941],
[0.1003456 , 0.8996544 ],
[0.91599965, 0.08400035],
[0.60085059, 0.39914941],
[0.83433941, 0.16566059],
[0.60085059, 0.39914941],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.03226176, 0.96773824],
[0.60085059, 0.39914941],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.03226176, 0.96773824],
[0.83433941, 0.16566059],
[0.1003456 , 0.8996544 ],
[0.03226176, 0.96773824],
[0.1003456 , 0.8996544 ],
[0.60085059, 0.39914941],
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.1003456 , 0.8996544 ],
[0.60085059, 0.39914941],
```

```
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.83433941, 0.16566059],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.1003456 , 0.8996544 ],
[0.91599965, 0.08400035],
[0.83433941, 0.16566059],
[0.03226176, 0.96773824],
[0.03226176, 0.96773824],
[0.19452137, 0.80547863],
[0.1003456 , 0.8996544 ],
[0.60085059, 0.39914941],
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.83433941, 0.16566059],
[0.03226176, 0.96773824],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.1003456 , 0.8996544 ],
[0.91599965, 0.08400035],
[0.60085059, 0.39914941],
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.83433941, 0.16566059],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.19452137, 0.80547863],
[0.1003456 , 0.8996544 ],
[0.19452137, 0.80547863],
[0.83433941, 0.16566059],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.03226176, 0.96773824],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.60085059, 0.39914941],
[0.1003456 , 0.8996544 ],
[0.19452137, 0.80547863],
[0.03226176, 0.96773824],
[0.91599965, 0.08400035],
[0.1003456 , 0.8996544 ],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.83433941, 0.16566059],
[0.83433941, 0.16566059],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.83433941, 0.16566059],
[0.83433941, 0.16566059],
[0.83433941, 0.16566059],
[0.03226176, 0.96773824],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.1003456 , 0.8996544 ],
[0.19452137, 0.80547863],
[0.83433941, 0.16566059],
[0.1003456 , 0.8996544 ],
[0.03226176, 0.96773824],
```

```
[0.91599965, 0.08400035],
[0.60085059, 0.39914941],
[0.19452137, 0.80547863],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.1003456 , 0.8996544 ],
[0.83433941, 0.16566059],
[0.19452137, 0.80547863],
[0.83433941, 0.16566059],
[0.19452137, 0.80547863],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.03226176, 0.96773824],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.03226176, 0.96773824],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.03226176, 0.96773824],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.83433941, 0.16566059],
[0.83433941, 0.16566059],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.83433941, 0.16566059],
[0.91599965, 0.08400035],
[0.1003456 , 0.8996544 ],
[0.19452137, 0.80547863],
[0.60085059, 0.39914941],
[0.19452137, 0.80547863],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.83433941, 0.16566059],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.83433941, 0.16566059],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.91599965, 0.08400035],
[0.91599965, 0.08400035],
[0.60085059, 0.39914941],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.83433941, 0.16566059],
[0.91599965, 0.08400035],
[0.83433941, 0.16566059],
[0.91599965, 0.08400035],
[0.19452137, 0.80547863],
[0.03226176, 0.96773824],
[0.91599965, 0.08400035],
```

```
       [0.03226176, 0.96773824],
       [0.19452137, 0.80547863]])
```

By using two variables/attributes we can predict the result more accurately;therefore always use subset of two variables which provide most accurate result.

In [57]:

```
y_pred=model.predict(x_test)
```

Determining the accuracy

In [58]:

```
accuracy_score(y_test, y_pred)
```

Out[58]:

```
0.7802690582959642
```

Therefore by using the relevant columns we can predict the result with more accuracy.

In [ ]:

```

```

Practical-3

Regression

Problem statement - To predict the age based on other variables

Libraries used - a) pandas
b) collections
c) sklearn

dataset - abalone
Plots - none
Metrics - accuracy score, mean absolute error.
Model - Regression & naive Bayes
NB - /coded
hence ↓ Regression

$$P(H/E) = \frac{P(E/H) * P(H)}{P(E)}$$

train to test split ratio - 75:25

~~Train accuracy~~

Practical No.3

Data Science and Visualization (Honors Course)

Name:Manjunath GB

PRN: 72018269H

Class: TE ENTC 'B'

To determine the age of abalone on the basis of its physical measurements

In [19]:

```
import pandas as pd
```

In [20]:

```
col = ['sex', 'length', 'diameter', 'height', 'weight', 'sweight', 'vweight', 'shweight',
'rings']
df=pd.read_csv('abalone.csv')
```

In [21]:

```
df.head()
```

Out[21]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

In [22]:

```
df.describe()
```

Out[22]:

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000
mean	0.523992	0.407881	0.139516	0.828742	0.359367	0.180594	0.238831	9.933684
std	0.120093	0.099240	0.041827	0.490389	0.221963	0.109614	0.139203	3.224169
min	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500	0.001500	1.000000
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500	0.130000	8.000000
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000	0.234000	9.000000
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000	0.329000	11.000000
max	0.815000	0.650000	1.130000	2.825500	1.488000	0.760000	1.005000	29.000000

We can say the dataset here is already cleaned because there are no null values.

In [24]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Sex             4177 non-null   object 
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 3   Height          4177 non-null   float64
 4   Whole weight    4177 non-null   float64
 5   Shucked weight  4177 non-null   float64
 6   Viscera weight  4177 non-null   float64
 7   Shell weight    4177 non-null   float64
 8   Rings           4177 non-null   int64  
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
```

In [12]:

```
X = df.drop('Rings' , axis=1) #Input
y = df['Rings'] #Output
```

In [13]:

```
X.head()
```

Out[13]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight
0	**M**	**0.455**	**0.365**	**0.095**	**0.5140**	**0.2245**	**0.1010**	**0.150**
1	**M**	**0.350**	**0.265**	**0.090**	**0.2255**	**0.0995**	**0.0485**	**0.070**
2	**F**	**0.530**	**0.420**	**0.135**	**0.6770**	**0.2565**	**0.1415**	**0.210**
3	**M**	**0.440**	**0.365**	**0.125**	**0.5160**	**0.2155**	**0.1140**	**0.155**
4	**I**	**0.330**	**0.255**	**0.080**	**0.2050**	**0.0895**	**0.0395**	**0.055**

In [14]:

```
from collections import Counter
Counter(y)
```

Out[14]:

```
Counter({15: 103,
         7: 391,
         9: 689,
         10: 634,
         8: 568,
         20: 26,
         16: 67,
         19: 32,
         14: 126,
         11: 487,
         12: 267,
         18: 42,
         13: 203,
         5: 115,
         4: 57,
         6: 259,
         21: 14,
         17: 58,
         22: 6,
         1: 1,
         3: 15,
         26: 1,
         23: 9,
         29: 1,
```

```
          2: 1,
          27: 2,
          25: 1,
          24: 2})
```

In [17]:

```
set(X['Sex']) #Displaying unique entries
```

Out[17]:

```
{'F', 'I', 'M'}
```

In [26]:

```
from sklearn.preprocessing import LabelEncoder
enc=LabelEncoder()
X['Sex']=enc.fit_transform(X['Sex'])
```

In [27]:

```
set(X['Sex'])
```

Out[27]:

```
{0, 1, 2}
```

In [28]:

```
df.head()
```

Out[28]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	**M**	**0.455**	**0.365**	**0.095**	**0.5140**	**0.2245**	**0.1010**	**0.150**	**15**
1	**M**	**0.350**	**0.265**	**0.090**	**0.2255**	**0.0995**	**0.0485**	**0.070**	**7**
2	**F**	**0.530**	**0.420**	**0.135**	**0.6770**	**0.2565**	**0.1415**	**0.210**	**9**
3	**M**	**0.440**	**0.365**	**0.125**	**0.5160**	**0.2155**	**0.1140**	**0.155**	**10**
4	**I**	**0.330**	**0.255**	**0.080**	**0.2050**	**0.0895**	**0.0395**	**0.055**	**7**

In [29]:

```
from sklearn.model_selection import train_test_split
```

In [34]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=0,test_size=0.25)

#Splitting the dataset
```

In [33]:

```
len(X_train)
```

Out[33]:

```
3132
```

In [35]:

```
len(X_test)
```

Out[35]:

```
1045
```

In [36]:

```
X_train.head()
```

```
X_train.head()
```

Out[36]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight
940	1	0.460	0.345	0.105	0.4490	0.1960	0.0945	0.1265
2688	2	0.630	0.465	0.150	1.0270	0.5370	0.1880	0.1760
1948	2	0.635	0.515	0.165	1.2290	0.5055	0.2975	0.3535
713	2	0.355	0.265	0.085	0.2010	0.0690	0.0530	0.0695
3743	0	0.705	0.555	0.195	1.7525	0.7105	0.4215	0.5160

Prediction

In [37]:

```
from sklearn.naive_bayes import GaussianNB
```

In [38]:

```
clf = GaussianNB()
```

In [39]:

```
#train
clf.fit(X_train,y_train)
```

Out[39]:

```
GaussianNB()
```

In [40]:

```
y_pred=clf.predict(X_test)
```

In [42]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

In [43]:

```
accuracy_score(y_test,y_pred)*100
```

Out[43]:

```
26.02870813397129
```

The accuracy score is low due to presence of multiple classes.

Regression

precision=TP/TP+FP

recall=TP/TP+FN

f1-score=2PR/P+R

Support is the number of actual occurences of class in a specified dataset.

In [44]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support
```

```
           3       0.50      1.00      0.67         7
           4       0.30      0.62      0.40        13
           5       0.27      0.42      0.33        40
           6       0.32      0.43      0.36        63
           7       0.26      0.36      0.30       114
           8       0.27      0.29      0.28       139
           9       0.25      0.30      0.27       152
          10       0.21      0.24      0.23       139
          11       0.26      0.42      0.32       121
          12       0.50      0.01      0.02        93
          13       0.00      0.00      0.00        51
          14       0.00      0.00      0.00        32
          15       0.00      0.00      0.00        22
          16       0.00      0.00      0.00        16
          17       0.00      0.00      0.00        12
          18       0.00      0.00      0.00         6
          19       0.00      0.00      0.00        10
          20       0.00      0.00      0.00         8
          21       0.00      0.00      0.00         2
          22       0.00      0.00      0.00         1
          23       0.00      0.00      0.00         2
          24       0.00      0.00      0.00         1
          27       0.00      0.00      0.00         0
          29       0.00      0.00      0.00         1

    accuracy                           0.26      1045
   macro avg       0.13      0.17      0.13      1045
weighted avg       0.24      0.26      0.22      1045
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\HP\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\HP\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\HP\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\HP\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\HP\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [45]:

```
from sklearn.linear_model import LinearRegression
```

In [46]:

```
reg=LinearRegression()
```

In [47]:

```
reg.fit(X_train,y_train)
```

Out[47]:

```
LinearRegression()
```

In [48]:

```
y_pred = reg.predict(X_test)
```

In [49]:

```
y_pred
```

Out[49]:

```
array([13.10451425,  9.66747548, 10.35605247, ...,  9.95962005,
       12.59111443, 12.18516586])
```

In [50]:

```
from sklearn.metrics import mean_absolute_error
```

In [51]:

```
mean_absolute_error(y_test,y_pred) #summation of (|y_pred-y_train|/no.of entries)
```

Out[51]:

```
1.5955158378194019
```

In [52]:

```
from sklearn.metrics import r2_score
```

In [53]:

```
r2_score(y_test,y_pred)    #r2_score = 1-(summation of (y_pred-y_train)^2 / summation of (mean of y_train - y_train)^2)
```

Out[53]:

```
0.5354158501894077
```

In this case we can say that Regression outperforms GaussianNB in terms of accuracy. ***(due to the dataset)***

In [ ]:

```

```

Practical 4

Problem Statement - To visualize diff attributes of dataset using bar & line graphs

dataset - netflix titles

Libraries used - Pandas, Collections, matplot

Plots used - Bar chart & Line chart.

Practical No.4

Data Science and Visualization (Honors Course)

Name:Manjunath GB

PRN: 72018269H

Class: TE ENTC 'B'

In this practical we will perform Data Visualization.

In [1]:

```
import pandas as pd
```

In [6]:

```
df = pd.read_csv('netflix_titles.csv')
df.head(8807)
```

Out[6]:

	show_id	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in
0	s1	Movie	Dick Johnson Is Dead	Kirsten Johnson	NaN	United States	September 25, 2021	2020	PG-13	90 min	Documentaries
1	s2	TV Show	Blood & Water	NaN	Ama Qamata, Khosi Ngema, Gail Mabalane, Thaban...	South Africa	September 24, 2021	2021	TV-MA	2 Seasons	International TV Shows, TV Dramas, TV Mysteries
2	s3	TV Show	Ganglands	Julien Leclercq	Sami Bouajila, Tracy Gotoas, Samuel Jouy, Nabi...	NaN	September 24, 2021	2021	TV-MA	1 Season	Crime TV Shows, International TV Shows, TV Act...
3	s4	TV Show	Jailbirds New Orleans	NaN	NaN	NaN	September 24, 2021	2021	TV-MA	1 Season	Docuseries, Reality TV
4	s5	TV Show	Kota Factory	NaN	Mayur More, Jitendra Kumar, Ranjan Raj, Alam K...	India	September 24, 2021	2021	TV-MA	2 Seasons	International TV Shows, Romantic TV Shows, TV ...
...	...	...	...	...	...	...	...	...	...	...	...
8802	s8803	Movie	Zodiac	David Fincher	Mark Ruffalo, Jake Gyllenhaal, Robert	United States	November 20, 2019	2007	R	158 min	Cult Movies, Dramas, Thrillers

	show_id	type	title	director	Downey cast J...	country	date_added	release_year	rating	duration	listed_in
8803	s8804	TV Show	Zombie Dumb	NaN	NaN	NaN	July 1, 2019	2018	TV-Y7	2 Seasons	Kids' TV, Korean TV Shows, TV Comedies
8804	s8805	Movie	Zombieland	Ruben Fleischer	Jesse Eisenberg, Woody Harrelson, Emma Stone, ...	United States	November 1, 2019	2009	R	88 min	Comedies, Horror Movies
8805	s8806	Movie	Zoom	Peter Hewitt	Tim Allen, Courteney Cox, Chevy Chase, Kate Ma...	United States	January 11, 2020	2006	PG	88 min	Children & Family Movies, Comedies
8806	s8807	Movie	Zubaan	Mozez Singh	Vicky Kaushal, Sarah-Jane Dias, Raaghav Chanan...	India	March 2, 2019	2015	TV-14	111 min	Dramas, International Movies, Music & Musicals

8807 rows × 12 columns

In [5]:

```
df.shape
```

Out[5]:

```
(8807, 12)
```

In [8]:

```
categories=df['listed_in']
```

In [9]:

```
total_child=sum(df['listed_in'].str.contains('Child'))
```

In [10]:

```
total_child
```

Out[10]:

```
641
```

In [11]:

```
Standup_Comedies=sum(df['listed_in'].str.contains('Stand'))
```

In [13]:

```
Standup_Comedies
```

Out[13]:

```
399
```

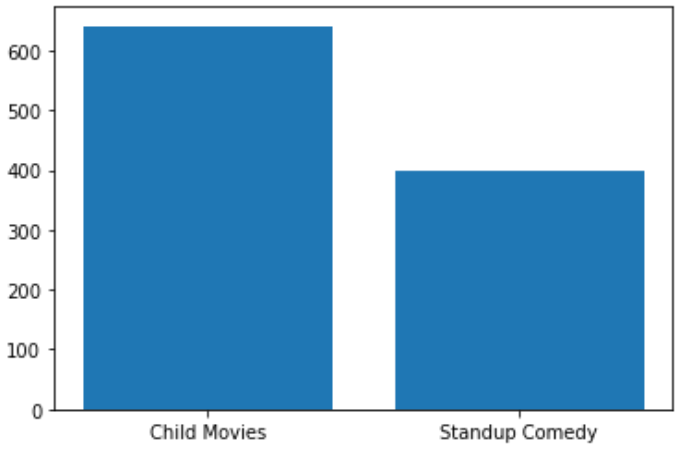
We determined the number of child movies/shows and standup comedies.We will visualize this number using plot.

In [14]:

```
import matplotlib.pyplot as plt
```

In [19]:

```
plt.bar(['Child Movies','Standup Comedy'],
        [total_child, Standup_Comedies])
plt.show()
```



In [20]:

```
set(df['type'])
```

Out[20]:

```
{'Movie', 'TV Show'}
```

In [21]:

```
tv_shows = df[df['type'] == 'TV Show'] #Boolean Filtering
```

In [30]:

```
seasons13 = tv_shows [tv_shows [ 'duration'] == '13 Seasons']

seasons15 = tv_shows [tv_shows['duration'] == '15 Seasons']

seasons16= tv_shows [tv_shows['duration'] == '16 Seasons']

seasons12 = tv_shows [tv_shows['duration'] == '12 Seasons']

seasons11= tv_shows [tv_shows['duration'] == '11 Seasons']
```

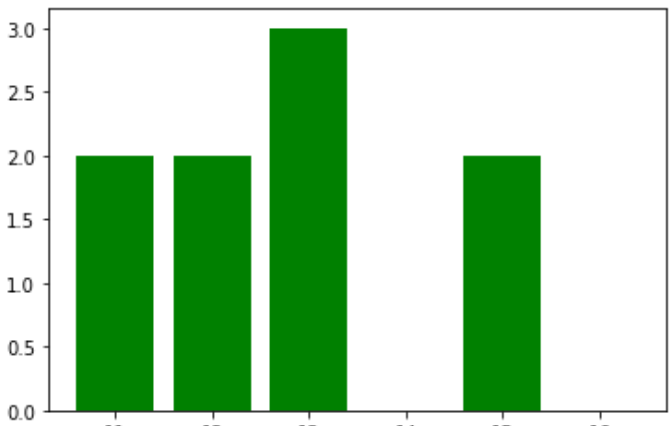
In [31]:

```
plt.bar ([11, 12, 13, 15, 16],
[len(seasons11), len(seasons12), len(seasons13), len (seasons15), len(seasons16)],

color='green')
```

Out[31]:

```
<BarContainer object of 5 artists>
```



11 12 13 14 15 16

In [32]:

```
from collections import Counter
ratings = Counter(df['rating'])
```

In [33]:

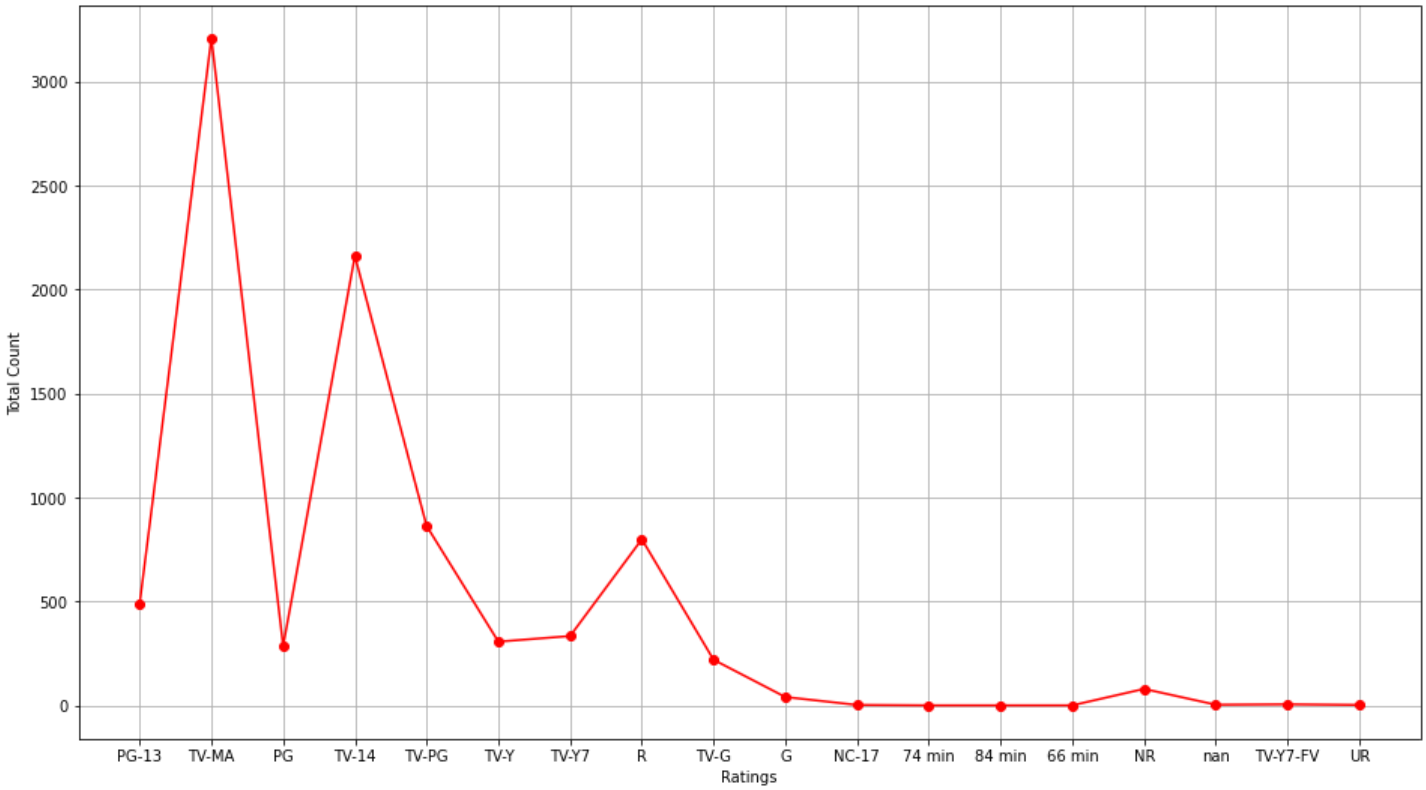
```
ratings
```

Out[33]:

```
Counter({'PG-13': 490,
         'TV-MA': 3207,
         'PG': 287,
         'TV-14': 2160,
         'TV-PG': 863,
         'TV-Y': 307,
         'TV-Y7': 334,
         'R': 799,
         'TV-G': 220,
         'G': 41,
         'NC-17': 3,
         '74 min': 1,
         '84 min': 1,
         '66 min': 1,
         'NR': 80,
         nan: 4,
         'TV-Y7-FV': 6,
         'UR': 3})
```

In [36]:

```
plt. figure(figsize=(16,9))
plt.plot(ratings.keys(), ratings.values(), color = 'red', marker='o')
plt.xlabel('Ratings'); plt.ylabel('Total Count')
plt.grid()
```



If we wish to plot all these plots in the same plot we can use subplot.

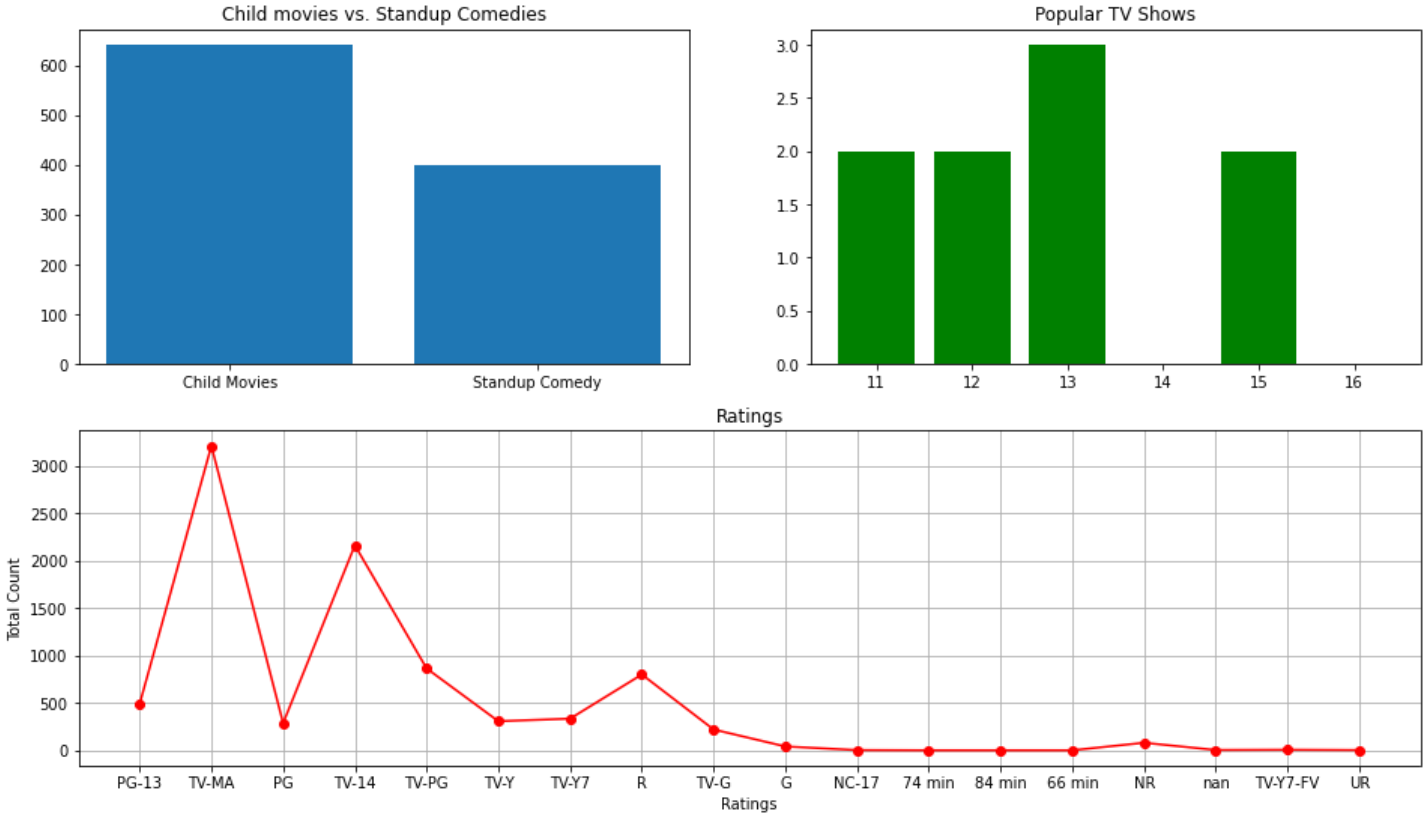
In [40]:

```
plt. figure (figsize=(16,9))
```

```
#plot1
plt.subplot (2,2,1)
plt.title ("Child movies vs. Standup Comedies")
plt.bar(['Child Movies', 'Standup Comedy'], [total_child, Standup_Comedies])

#plot2
plt. subplot (2,2,2)
plt.title('Popular TV Shows')
plt.bar([11, 12, 13, 15, 16],
[len (seasons11), len (seasons12), len(seasons13),
len (seasons15), len (seasons16)],
color='green')

#plot3
plt.subplot (2,1,2)
plt.title('Ratings')
plt.plot(ratings.keys (), ratings.values(), color='red', marker='o')
plt.xlabel('Ratings'); plt.ylabel('Total Count')
plt.grid()
```



In [ ]: