Problem Statement :

As a data scientist of a leading decision analysis firm you are required to predict the potential global user of the game based on the data provided by the customer so that they can plan their global launch.

IMPORTING THE LIBRARIES

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

IMPORTING THE DATSETS

In [2]:

```
train_data=pd.read_csv('Train.csv',index_col=None,header=0)
train_data.head()
test_data=pd.read_csv('Test.csv',index_col=None,header=0)
test_data.head()
```

Out[2]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sa
0	Nicktoons: MLB	X360	2011.0	Sports	Take-Two Interactive	0.12	0.00	
1	Shonen Jump's One Piece: Grand Battle	PS2	2005.0	Fighting	Atari	0.07	0.05	
2	Learn Math	DS	2009.0	Puzzle	DreamCatcher Interactive	0.12	0.00	
3	Nitrobike	Wii	2008.0	Racing	Ubisoft	0.11	0.01	
4	Cruise Ship Vacation Games	Wii	2009.0	Puzzle	Avanquest	0.12	0.00	

In [3]:

```
print(train_data.shape)
print(test_data.shape)
```

```
(14576, 15)
(2143, 14)
```

CHECKING NULL VALUES

In [4]:

```
train_data.isnull().sum()
```

Out[4]:

```
Name                  2
Platform              0
Year_of_Release     232
Genre                 2
Publisher            49
NA_Sales              0
EU_Sales              0
JP_Sales              0
Critic_Score       7359
Critic_Count       7359
User_Score         5816
User_Count         7780
Developer          5747
Rating             5872
Global_Sales          0
dtype: int64
```

In [5]:

```
test_data.isnull().sum()
```

Out[5]:

```
Name                  0
Platform              0
Year_of_Release      37
Genre                 0
Publisher             5
NA_Sales              0
EU_Sales              0
JP_Sales              0
Critic_Score       1223
Critic_Count       1223
User_Score          888
User_Count         1349
Developer           876
Rating              897
dtype: int64
```

In [6]:

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14576 entries, 0 to 14575
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Name             14574 non-null  object 
 1   Platform         14576 non-null  object 
 2   Year_of_Release  14344 non-null  float64
 3   Genre            14574 non-null  object 
 4   Publisher        14527 non-null  object 
 5   NA_Sales         14576 non-null  float64
 6   EU_Sales         14576 non-null  float64
 7   JP_Sales         14576 non-null  float64
 8   Critic_Score     7217 non-null   float64
 9   Critic_Count     7217 non-null   float64
 10  User_Score       8760 non-null   object 
 11  User_Count       6796 non-null   float64
 12  Developer        8829 non-null   object 
 13  Rating           8704 non-null   object 
 14  Global_Sales     14576 non-null  float64
dtypes: float64(8), object(7)
memory usage: 1.7+ MB
```

In [7]:

```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2143 entries, 0 to 2142
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Name             2143 non-null   object 
 1   Platform         2143 non-null   object 
 2   Year_of_Release  2106 non-null   float64
 3   Genre            2143 non-null   object 
 4   Publisher        2138 non-null   object 
 5   NA_Sales         2143 non-null   float64
 6   EU_Sales         2143 non-null   float64
 7   JP_Sales         2143 non-null   float64
 8   Critic_Score     920 non-null    float64
 9   Critic_Count     920 non-null    float64
 10  User_Score       1255 non-null   object 
 11  User_Count       794 non-null    float64
 12  Developer        1267 non-null   object 
 13  Rating           1246 non-null   object 
dtypes: float64(7), object(7)
memory usage: 234.5+ KB
```

CHECKING FOR UNIQUE VALUES

In [8]:

```
for i in train_data.columns:
    print({i:train_data[i].unique()})
```

```
mes',
       'Acquire', 'Broccoli', 'General Entertainment',
       'Paradox Interactive', 'Yacht Club Games', 'Imadio',
       'Swing! Entertainment', 'Sony Music Entertainment', 'Aqu
a Plus',
       'Excalibur Publishing', 'Hip Interactive', 'Tripwire Int
eractive',
       'Bigben Interactive', 'Sting', 'Data East',
       'Idea Factory International', 'Time Warner Interactive',
       'Gainax Network Systems', 'Daito', 'O3 Entertainment',
'O-Games',
       'Gameloft', 'Xicat Interactive', 'Simon & Schuster Inter
active',
       'Valcon Games', 'PopTop Software', 'TOHO', 'PM Studios',
       'Bohemia Interactive', 'Reef Entertainment', '5pb',
       'HMH Interactive', 'DreamCatcher Interactive',
       'inXile Entertainment', 'Cave', 'Microids', 'Paon', 'Ide
a Factory',
       'U.S. Gold', 'CDV Software Entertainment', 'Micro Cabi
n',
```

In [9]:

```
for i in test_data.columns:
    print({i:test_data[i].unique()})
```

```
{'Name': array(['Nicktoons: MLB', "Shonen Jump's One Piece: Gra
nd Battle",
       'Learn Math', ..., 'Theresia...', 'Sacred 2: Fallen Ange
l',
       'Dance Sensation!'], dtype=object)}
{'Platform': array(['X360', 'PS2', 'DS', 'Wii', 'GBA', 'PS', 'P
SV', 'XB', '3DS', 'PSP',
       'PS3', 'PC', 'XOne', 'GC', 'SAT', 'PS4', 'SNES', 'N64',
'WiiU',
       '2600', 'DC', 'GEN', 'NG', 'GB', 'SCD', 'NES'], dtype=ob
ject)}
{'Year_of_Release': array([2011., 2005., 2009., 2008., 2010., 1
995., 2016., 2004., 1997.,
       2007., 2015., 2000., 2013., 2003., 2001.,   nan, 1998.,
2014.,
       2002., 2006., 2012., 1992., 1993., 1999., 1981., 1996.,
1994.,
       1991., 1987.])}
{'Genre': array(['Sports', 'Fighting', 'Puzzle', 'Racing', 'Mis
```

In [10]:

```
train_data.describe()
```

Out[10]:

	Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Critic_Score	Critic_Coun
count	14344.000000	14576.000000	14576.000000	14576.000000	7217.000000	7217.00000
mean	2006.437117	0.295577	0.163957	0.085659	69.676043	27.31204
std	5.955664	0.866491	0.536354	0.329646	13.773391	19.46419
min	1980.000000	0.000000	0.000000	0.000000	13.000000	3.00000
25%	2003.000000	0.000000	0.000000	0.000000	61.000000	12.00000
50%	2007.000000	0.100000	0.030000	0.000000	71.000000	22.00000
75%	2010.000000	0.280000	0.130000	0.030000	80.000000	38.00000
max	2020.000000	41.360000	28.960000	10.220000	98.000000	113.00000

In [11]:

```
test_data.describe()
```

Out[11]:

	Year_of_Release	NA_Sales	EU_Sales	JP_Sales	Critic_Score	Critic_Count
count	2106.000000	2143.000000	2143.000000	2143.000000	920.000000	920.000000
mean	2006.829535	0.043999	0.016253	0.022804	63.410870	18.898913
std	5.316215	0.036211	0.020982	0.038525	13.981594	12.320334
min	1981.000000	0.000000	0.000000	0.000000	19.000000	4.000000
25%	2004.000000	0.000000	0.000000	0.000000	55.000000	9.000000
50%	2008.000000	0.050000	0.010000	0.000000	65.000000	17.000000
75%	2011.000000	0.070000	0.030000	0.060000	73.000000	25.000000
max	2016.000000	0.120000	0.130000	0.130000	92.000000	77.000000

In [12]:

```
train_data.columns
```

Out[12]:

```
Index(['Name', 'Platform', 'Year_of_Release', 'Genre', 'Publisher',
'NA_Sales',
       'EU_Sales', 'JP_Sales', 'Critic_Score', 'Critic_Count', 'Use
r_Score',
       'User_Count', 'Developer', 'Rating', 'Global_Sales'],
      dtype='object')
```

In [13]:

```
# define function to check null values prcentage for each coloumn
def null_values(x):
    percent_missing = x.isnull().sum() * 100 / len(x)
    missing_value_x = pd.DataFrame({'column_name': x.columns,
                                 'percent_missing': percent_missing})
    print(missing_value_x)
```

In [14]:

```
null_values(train_data)
```

```
                     column_name  percent_missing
Name                        Name         0.013721
Platform                Platform         0.000000
Year_of_Release  Year_of_Release         1.591658
Genre                      Genre         0.013721
Publisher              Publisher         0.336169
NA_Sales                NA_Sales         0.000000
EU_Sales                EU_Sales         0.000000
JP_Sales                JP_Sales         0.000000
Critic_Score        Critic_Score        50.487102
Critic_Count        Critic_Count        50.487102
User_Score            User_Score        39.901207
User_Count            User_Count        53.375412
Developer              Developer        39.427827
Rating                    Rating        40.285401
Global_Sales        Global_Sales         0.000000
```

In [15]:

```
null_values(test_data)
```

```
                     column_name  percent_missing
Name                        Name         0.000000
Platform                Platform         0.000000
Year_of_Release  Year_of_Release         1.726552
Genre                      Genre         0.000000
Publisher              Publisher         0.233318
NA_Sales                NA_Sales         0.000000
EU_Sales                EU_Sales         0.000000
JP_Sales                JP_Sales         0.000000
Critic_Score        Critic_Score        57.069529
Critic_Count        Critic_Count        57.069529
User_Score            User_Score        41.437238
User_Count            User_Count        62.949137
Developer              Developer        40.877275
Rating                    Rating        41.857210
```

TREATING THE NULL VALUES

In [16]:

```
train_data.drop_duplicates(subset='Name', keep='first', inplace=True)

print(train_data.shape)
```

```
(10328, 15)
```

In [17]:

```
test_data.drop_duplicates(subset='Name', keep='first', inplace=True)

print(test_data.shape)
```

```
(1988, 14)
```

In [18]:

```
# deleate tbd from user score and replace it with nan to replace nan with appopr
train_data['User_Score']=train_data['User_Score'].replace('tbd', np.NaN)

#convert user score to float insted of string
train_data.User_Score=train_data.User_Score.values.astype(float)
```

In [19]:

```
# deleate tbd from user score and replace it with nan to replace nan with appopr
test_data['User_Score']=test_data['User_Score'].replace('tbd', np.NaN)

#convert user score to float insted of string
test_data.User_Score=test_data.User_Score.values.astype(float)
```

FILLING THE NULL VALUES

In [20]:

```
#since null values is exceeding 30%-51% in these coloums
#if the numeric colom is close to the mean fill nan with mean else fill it with
#fill nan in categorical data by mode or median

# since the coloums has no outliers we obtain mean value to null values in the c
train_data['Critic_Score'] = train_data['Critic_Score'].fillna(train_data['Criti
train_data['Critic_Count'] = train_data['Critic_Count'].fillna(train_data['Criti
train_data['User_Score'] = train_data['User_Score'].fillna(train_data['User_Scor

# since the coloums has outliers we obtain median value to null values in the co
train_data['User_Count'] = train_data['User_Count'].fillna(train_data['User_Coun

# since the coloums has categorical data we obtain mode value to null values in
train_data['Rating'] = train_data['Rating'].fillna(train_data['Rating'].mode().i
train_data['Developer'] = train_data['Developer'].fillna(train_data['Developer']
train_data['Publisher'] = train_data['Publisher'].fillna(train_data['Publisher']
train_data['Genre'] = train_data['Genre'].fillna(train_data['Genre'].mode().iat[
```

In [21]:

```
#since null values is exceeding 30%-51% in these coloums
#if the numeric colom is close to the mean fill nan with mean else fill it with
#fill nan in categorical data by mode or median

# since the coloums has no outliers we obtain mean value to null values in the c
test_data['Critic_Score'] = test_data['Critic_Score'].fillna(test_data['Critic_S
test_data['Critic_Count'] = test_data['Critic_Count'].fillna(test_data['Critic_C
test_data['User_Score'] = test_data['User_Score'].fillna(test_data['User_Score']

# since the coloums has outliers we obtain median value to null values in the co
test_data['User_Count'] = test_data['User_Count'].fillna(test_data['User_Count']

# since the coloums has categorical data we obtain mode value to null values in
test_data['Rating'] = test_data['Rating'].fillna(test_data['Rating'].mode().iat[
test_data['Developer'] = test_data['Developer'].fillna(test_data['Developer'].mo
test_data['Publisher'] = test_data['Publisher'].fillna(test_data['Publisher'].mo
test_data['Genre'] = test_data['Genre'].fillna(test_data['Genre'].mode().iat[0])
```

In [22]:

```
train_data.head()
```

Out[22]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sa
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.96	3
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	12.76	3
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.93	3
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10

In [23]:

```
test_data.head()
```

Out[23]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sa
0	Nicktoons: MLB	X360	2011.0	Sports	Take-Two Interactive	0.12	0.00	
1	Shonen Jump's One Piece: Grand Battle	PS2	2005.0	Fighting	Atari	0.07	0.05	
2	Learn Math	DS	2009.0	Puzzle	DreamCatcher Interactive	0.12	0.00	
3	Nitrobike	Wii	2008.0	Racing	Ubisoft	0.11	0.01	
4	Cruise Ship Vacation Games	Wii	2009.0	Puzzle	Avanquest	0.12	0.00	

In [24]:

```
train_data=train_data.drop(columns=['Year_of_Release'], axis=1)
train_data
```

Out[24]:

	Name	Platform	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Crit
0	Wii Sports	Wii	Sports	Nintendo	41.36	28.96	3.77	7
1	Super Mario Bros.	NES	Platform	Nintendo	29.08	3.58	6.81	6
2	Mario Kart Wii	Wii	Racing	Nintendo	15.68	12.76	3.79	8.
3	Wii Sports Resort	Wii	Sports	Nintendo	15.61	10.93	3.28	8
4	Pokemon Red/Pokemon Blue	GB	Role-Playing	Nintendo	11.27	8.89	10.22	6
...	...	...	...	...	...	...	...	
14566	15 Days	PC	Adventure	DTP Entertainment	0.00	0.01	0.00	6.
14568	Aiyoku no Eustia	PSV	Misc	dramatic create	0.00	0.00	0.01	6
14569	Woody Woodpecker in Crazy Castle 5	GBA	Platform	Kemco	0.01	0.00	0.00	6
14572	LMA Manager 2007	X360	Sports	Codemasters	0.00	0.01	0.00	6
14573	Haitaka no Psychedelica	PSV	Adventure	Idea Factory	0.00	0.00	0.01	6

10328 rows × 14 columns

In [25]:

```
train_data.isnull().sum()
```

Out[25]:

```
Name            1
Platform        0
Genre           0
Publisher       0
NA_Sales        0
EU_Sales        0
JP_Sales        0
Critic_Score    0
Critic_Count    0
User_Score      0
User_Count      0
Developer       0
Rating          0
Global_Sales    0
dtype: int64
```

In [26]:

```
test_data=test_data.drop(columns=['Year_of_Release'], axis=1)
test_data
```

Out[26]:

	Name	Platform	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Critic
0	Nicktoons: MLB	X360	Sports	Take-Two Interactive	0.12	0.00	0.0	63.
1	Shonen Jump's One Piece: Grand Battle	PS2	Fighting	Atari	0.07	0.05	0.0	63.
2	Learn Math	DS	Puzzle	DreamCatcher Interactive	0.12	0.00	0.0	63.
3	Nitrobike	Wii	Racing	Ubisoft	0.11	0.01	0.0	49.
4	Cruise Ship Vacation Games	Wii	Puzzle	Avanquest	0.12	0.00	0.0	63.
...	...	...	...	...	...	...	...	
2136	PoPoLoCrois	PSP	Role-Playing	Ignition Entertainment	0.05	0.00	0.0	66.
2137	Dragon Rage	PS2	Shooter	3DO	0.03	0.02	0.0	50.
2138	Theresia...	DS	Adventure	Arc System Works	0.05	0.00	0.0	61.
2139	Sacred 2: Fallen Angel	PC	Role-Playing	Ascaron Entertainment GmbH	0.00	0.05	0.0	71.
2140	Dance Sensation!	Wii	Misc	Majesco Entertainment	0.06	0.00	0.0	63.

1988 rows × 13 columns

In [27]:

```
test_data.isnull().sum()
```

Out[27]:

```
Name            0
Platform        0
Genre           0
Publisher       0
NA_Sales        0
EU_Sales        0
JP_Sales        0
Critic_Score    0
Critic_Count    0
User_Score      0
User_Count      0
Developer       0
Rating          0
dtype: int64
```

In [28]:

```
test_data.shape
```

Out[28]:

```
(1988, 13)
```

In [ ]:

In [ ]:

In [ ]:

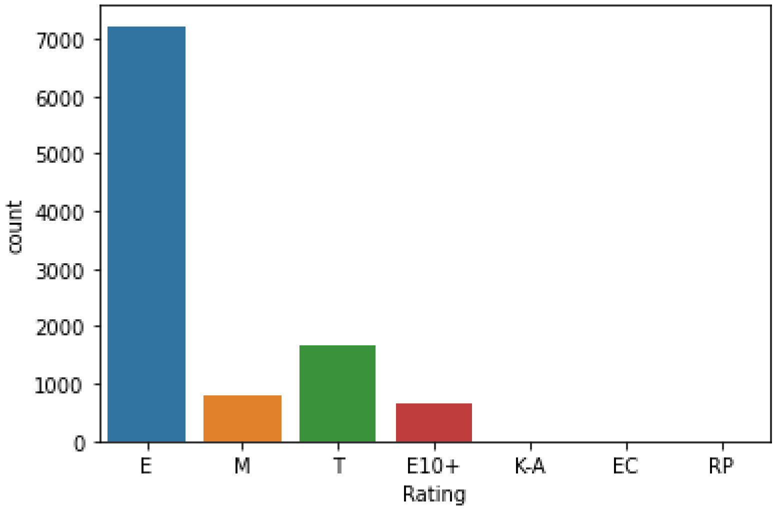
COUNT PLOT WITH RATING FOR EDA

In [29]:

```
sns.countplot(x='Rating',data=train_data)
```

Out[29]:

<AxesSubplot:xlabel='Rating', ylabel='count'>

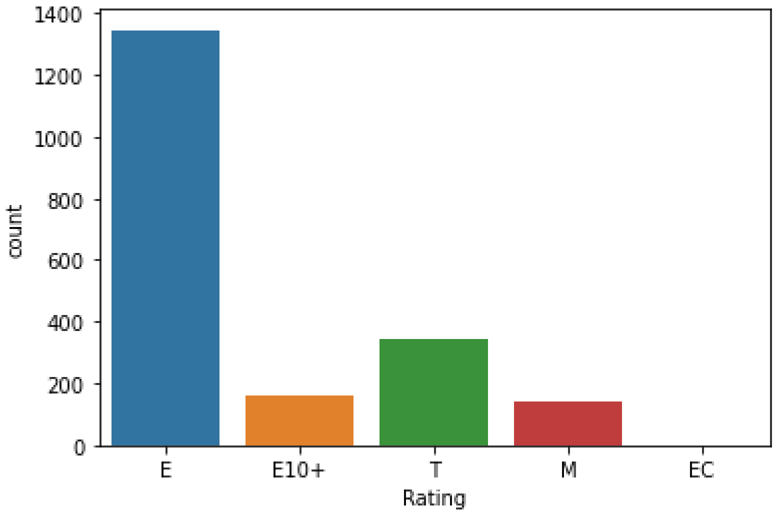


In [30]:

```
sns.countplot(x='Rating',data=test_data)
```

Out[30]:

<AxesSubplot:xlabel='Rating', ylabel='count'>



In [31]:

```
colname=[]
for x in train_data.columns:
    if train_data[x].dtype=='object':
        colname.append(x)
colname
```

Out[31]:

```
['Name', 'Platform', 'Genre', 'Publisher', 'Developer', 'Rating']
```

In [32]:

```
colname=['Platform', 'Genre', 'Publisher', 'Developer', 'Rating']
```

CONVERTING CATEGORICAL TO NUMERICAL

In [33]:

```
#for preprocessing the data
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()

for x in colname:
    train_data[x]=le.fit_transform(train_data[x])

    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    print('Feature', x)
    print('mapping', le_name_mapping)
```

```
n': 58, 'Amanita Design': 59, 'Amaze Entertainment': 60, 'Amaze
Entertainment, Walt Disney Japan': 61, 'Amble': 62, 'Ambrella':
63, 'Ambrella, The Pokemon Company': 64, 'Amusement Vision': 6
5, 'Amuze': 66, 'Anchor': 67, 'Ancient': 68, 'Andamiro U.S.A. C
orp.': 69, 'Angel Studios': 70, 'Anino Entertainment': 71, 'Apo
lloSoft': 72, 'Appaloosa Interactive': 73, 'Aqua Pacific': 74,
'Aqua Pacific, In2Games': 75, 'Aqua Plus': 76, 'Aquria': 77, 'A
rc System Works': 78, 'Arcade Moon': 79, 'ArenaNet': 80, 'Argon
aut Games': 81, 'Arika': 82, 'Arkane Studios': 83, 'Arkedo Stud
io': 84, 'Armature Studio': 85, 'Armature Studio, comcept': 86,
'Arrowhead Game Studios': 87, 'Art': 88, 'Artdink': 89, 'ArtePi
azza': 90, 'Artech Studios': 91, 'Artefacts Studio': 92, 'Artif
icial Mind and Movement': 93, 'Artificial Mind and Movement, Po
lygon Magic': 94, 'Artificial Studios, Immersion Software & Gra
phics': 95, 'Artoon': 96, 'Arts Software': 97, 'Arzest': 98, 'A
scaron Entertainment GmbH': 99, 'Ascaron Entertainment GmbH, As
caron Entertainment': 100, 'Asmik Ace Entertainment, Inc': 101,
'Asobo Studio': 102, 'Aspect': 103, 'Aspect, Takara Tomy': 104,
'Aspyr': 105, 'Astroll': 106, 'Asylum Entertainment': 107, 'Ata
ri': 108, 'Atari, Atari SA': 109, 'Atari, Transmission Games, A
```

In [34]:

```
#for preprocessing the data
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()

for x in colname:
    test_data[x]=le.fit_transform(test_data[x])

    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    print('Feature', x)
    print('mapping', le_name_mapping)
```

```
mapping {'10TACLE Studios': 0, '3DO': 1, '505 Games': 2, '5pb':
3, 'ASCII Entertainment': 4, 'Abylight': 5, 'Acclaim Entertainm
ent': 6, 'Ackkstudios': 7, 'Acquire': 8, 'Activision': 9, 'Acti
vision Value': 10, 'Agatsuma Entertainment': 11, 'Agetec': 12,
'Aksys Games': 13, 'Alchemist': 14, 'Alternative Software': 15,
'Altron': 16, 'Angel Studios': 17, 'Aqua Plus': 18, 'Arc System
Works': 19, 'Aria': 20, 'ArtDink': 21, 'Ascaron Entertainment G
mbH': 22, 'Asgard': 23, 'Asmik Corp': 24, 'Aspyr': 25, 'Asylum
Entertainment': 26, 'Atari': 27, 'Athena': 28, 'Atlus': 29, 'Av
alon Interactive': 30, 'Avanquest': 31, 'BAM! Entertainment': 3
2, 'BMG Interactive Entertainment': 33, 'Banpresto': 34, 'Benes
se': 35, 'Bethesda Softworks': 36, 'Big Ben Interactive': 37,
'Bigben Interactive': 38, 'Black Bean Games': 39, 'Brash Entert
ainment': 40, 'Broccoli': 41, 'Capcom': 42, 'Cave': 43, 'City I
nteractive': 44, 'Codemasters': 45, 'Compile Heart': 46, 'Consp
iracy Entertainment': 47, 'Crave Entertainment': 48, 'Crimson C
ow': 49, 'Crystal Dynamics': 50, 'Crytek': 51, 'D3Publisher': 5
2, 'DHM Interactive': 53, 'DSI Games': 54, 'DTP Entertainment':
55, 'Daedalic Entertainment': 56, 'Daito': 57, 'Data Design Int
eractive': 58, 'Deep Silver': 59, 'Destination Software, Inc':
```

In [35]:

```
test_data.head()
```

Out[35]:

	Name	Platform	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Critic_Score	Criti
0	Nicktoons: MLB	23	10	231	0.12	0.00	0.0	63.794293	19
1	Shonen Jump's One Piece: Grand Battle	13	2	27	0.07	0.05	0.0	63.794293	19
2	Learn Math	3	5	63	0.12	0.00	0.0	63.794293	19
3	Nitrobike	21	6	246	0.11	0.01	0.0	49.000000	22
4	Cruise Ship Vacation Games	21	5	31	0.12	0.00	0.0	63.794293	19

In [36]:

```
train_data.head()
```

Out[36]:

	Name	Platform	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Critic_Score	C
0	Wii Sports	26	10	338	41.36	28.96	3.77	76.000000	
1	Super Mario Bros.	11	4	338	29.08	3.58	6.81	69.412253	
2	Mario Kart Wii	26	6	338	15.68	12.76	3.79	82.000000	
3	Wii Sports Resort	26	10	338	15.61	10.93	3.28	80.000000	
4	Pokemon Red/Pokemon Blue	5	7	338	11.27	8.89	10.22	69.412253	

In [37]:

```
trained_data=train_data
```

In [38]:

```
x=trained_data.iloc[:,1:-1]
y=trained_data.iloc[:,-1]
```

In [39]:

```
from sklearn.preprocessing import StandardScaler
scaler= StandardScaler()
scaler.fit(x)
x=scaler.transform(x)
```

In [40]:

```
print(x)
```

```
[[ 1.40264694  1.354057    0.33971657 ...  0.66936424 -0.30052462
  -0.58103818]
 [-0.49014388 -0.26810988  0.33971657 ... -0.1543747   0.7440839
  -0.58103818]
 [ 1.40264694  0.27261241  0.33971657 ...  1.75737437 -0.30052462
  -0.58103818]
 ...
 [-1.12107415 -0.26810988 -0.19265326 ... -0.1543747   0.7440839
  -0.58103818]
 [ 1.65501905  1.354057   -1.22607704 ... -0.1543747   0.7440839
  -0.58103818]
 [ 0.64553062 -1.07919333 -0.41812754 ... -0.1543747   0.7440839
  -0.58103818]]
```

TRAIN TEST SPLIT

In [41]:

```
from sklearn.model_selection import train_test_split

#split the data into  the test and train
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.3, random_stat

## test_size is 20%  for less than 1000 obsevation and greater than 1000 30%
```

In [42]:

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(7229, 12)
(7229,)
(3099, 12)
(3099,)
```

DECISION TREE

In [43]:

```
from sklearn.tree import DecisionTreeRegressor

model_DecisionTree=DecisionTreeRegressor(random_state=10)

#fit the model on data and predict the values

model_DecisionTree.fit(x_train,y_train)

y_pred=model_DecisionTree.predict(x_test)

#print(y_pred)
print(list(zip(y_test,y_pred)))
```

```
[(0.05, 0.05000000000000002), (1.51, 1.52), (0.35, 0.46), (0.0
6, 0.05), (0.04, 0.04000000000000003), (1.83, 1.76), (0.14, 0.1
4), (0.02, 0.03), (2.15, 2.14), (0.04, 0.04), (1.04, 1.03), (0.
19, 0.19), (0.76, 0.84), (1.86, 1.82), (3.83, 2.69), (0.03, 0.0
4), (0.11, 0.11), (0.05, 0.04999999999999996), (4.48, 4.68),
(0.03, 0.03), (0.02, 0.02000000000000001), (4.8, 4.36), (1.67,
1.93), (0.06, 0.1), (1.96, 2.14), (0.03, 0.030000000000000016),
(0.02, 0.020000000000000007), (0.04, 0.05000000000000001), (1.6
4, 1.44), (0.72, 0.69), (0.49, 0.65), (0.83, 0.73), (0.28, 0.2
8), (1.75, 1.65), (0.11, 0.10999999999999996), (1.88, 1.7), (0.
21, 0.20000000000000004), (0.1, 0.11), (0.33, 0.38), (0.18, 0.1
3), (0.29, 0.26), (0.99, 0.87), (0.8, 0.83), (0.23, 0.24), (0.1
9, 0.19), (0.1, 0.11), (0.78, 0.82), (0.18, 0.16), (0.03, 0.0
4), (0.02, 0.02), (0.02, 0.020000000000000007), (0.15, 0.15),
(0.04, 0.030000000000000016), (0.31, 0.28), (0.25, 0.28), (1.7
5, 1.74), (0.25, 0.25), (0.17, 0.17), (0.21, 0.21), (0.1, 0.1),
(4.93, 5.1), (0.06, 0.06000000000000004), (0.17, 0.17), (0.19,
0.19), (0.17, 0.17), (5.0, 5.64), (0.02, 0.02000000000000001),
(0.03, 0.03000000000000002), (0.55, 0.48), (0.55, 0.58), (0.02,
```

In [44]:

```
from sklearn.metrics import r2_score,mean_squared_error
import numpy as np

r2=r2_score(y_test,y_pred)
print("R-squared:",r2)

rmse=np.sqrt(mean_squared_error(y_test,y_pred))
print("RMSE:",rmse)

adjusted_r_squared = 1 - (1-r2)*(len(y)-1)/(len(y)-x.shape[1]-1)
print("Adj R-square:",adjusted_r_squared)
```

```
R-squared: 0.7959071924570826
RMSE: 0.9475904889524515
Adj R-square: 0.7956697602040032
```

RANDOM FOREST REGRESSOR

In [45]:

```
from sklearn.ensemble import RandomForestRegressor

model_RandomForest=RandomForestRegressor(n_estimators=100, random_state=10)

##fit the model in the data and predict the values

model_RandomForest.fit(x_train,y_train)

y_pred=model_RandomForest.predict(x_test)
```

In [46]:

```
from sklearn.metrics import r2_score,mean_squared_error
import numpy as np

r2=r2_score(y_test,y_pred)
print("R-squared:",r2)

rmse=np.sqrt(mean_squared_error(y_test,y_pred))
print("RMSE:",rmse)

adjusted_r_squared = 1 - (1-r2)*(len(y)-1)/(len(y)-x.shape[1]-1)
print("Adj R-square:",adjusted_r_squared)
```

```
R-squared: 0.8162789868998828
RMSE: 0.8990549962271358
Adj R-square: 0.8160652542622482
```

EXTRA TREES

In [47]:

```
from sklearn.ensemble import ExtraTreesRegressor

model_ExtraTrees=ExtraTreesRegressor(n_estimators=10, random_state=10)

##fit the model in the data and predict the values

model_ExtraTrees.fit(x_train,y_train)

y_pred=model_ExtraTrees.predict(x_test)
```

In [48]:

```
from sklearn.metrics import r2_score,mean_squared_error
import numpy as np

r2=r2_score(y_test,y_pred)
print("R-squared:",r2)

rmse=np.sqrt(mean_squared_error(y_test,y_pred))
print("RMSE:",rmse)

adjusted_r_squared = 1 - (1-r2)*(len(y)-1)/(len(y)-x.shape[1]-1)
print("Adj R-square:",adjusted_r_squared)
```

```
R-squared: 0.7992053237712902
RMSE: 0.9399027927191265
Adj R-square: 0.7989717284135835
```

ADA BOOST

In [49]:

```
from sklearn.ensemble import AdaBoostRegressor

model_AdaBoost=AdaBoostRegressor(base_estimator=DecisionTreeRegressor(random_sta
                                  random_state=10)

##fit the model in the data and predict the values

model_AdaBoost.fit(x_train,y_train)

y_pred=model_AdaBoost.predict(x_test)
```

In [50]:

```
from sklearn.metrics import r2_score,mean_squared_error
import numpy as np

r2=r2_score(y_test,y_pred)
print("R-squared:",r2)

rmse=np.sqrt(mean_squared_error(y_test,y_pred))
print("RMSE:",rmse)

adjusted_r_squared = 1 - (1-r2)*(len(y)-1)/(len(y)-x.shape[1]-1)
print("Adj R-square:",adjusted_r_squared)
```

```
R-squared: 0.8232964853454983
RMSE: 0.8817174519386826
Adj R-square: 0.8230909165451246
```

GRADIENT BOOSTING

In [51]:

```
from sklearn.ensemble import GradientBoostingRegressor

model_GradientBoosting=GradientBoostingRegressor(n_estimators=250,random_state=1

##fit the model in the data and predict the values

model_GradientBoosting.fit(x_train,y_train)

y_pred=model_GradientBoosting.predict(x_test)
```

In [52]:

```
from sklearn.metrics import r2_score,mean_squared_error
import numpy as np

r2=r2_score(y_test,y_pred)
print("R-squared:",r2)

rmse=np.sqrt(mean_squared_error(y_test,y_pred))
print("RMSE:",rmse)

adjusted_r_squared = 1 - (1-r2)*(len(y)-1)/(len(y)-x.shape[1]-1)
print("Adj R-square:",adjusted_r_squared)
```

```
R-squared: 0.8414916491337483
RMSE: 0.8350892956043895
Adj R-square: 0.8413072477561045
```

XG BOOST

In [53]:

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /opt/anaconda3/lib/python
3.8/site-packages (1.7.4)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.
8/site-packages (from xgboost) (1.22.4)
Requirement already satisfied: scipy in /opt/anaconda3/lib/python3.
8/site-packages (from xgboost) (1.6.2)
Note: you may need to restart the kernel to use updated packages.
```

In [54]:

```
from xgboost import XGBRegressor

model_xgb=XGBRegressor(n_estimators=200,random_state=10)

##fit the model in the data and predict the values

model_xgb.fit(x_train,y_train)

y_pred=model_xgb.predict(x_test)
```

In [55]:

```
from sklearn.metrics import r2_score,mean_squared_error
import numpy as np

r2=r2_score(y_test,y_pred)
print("R-squared:",r2)

rmse=np.sqrt(mean_squared_error(y_test,y_pred))
print("RMSE:",rmse)

adjusted_r_squared = 1 - (1-r2)*(len(y)-1)/(len(y)-x.shape[1]-1)
print("Adj R-square:",adjusted_r_squared)
```

```
R-squared: 0.8239443669621294
RMSE: 0.8800995636057009
Adj R-square: 0.8237395518776451
```

KNN

In [56]:

```
##predicting using the KNeighbors_classifier
from sklearn.neighbors import KNeighborsRegressor
model_KNN=KNeighborsRegressor(n_neighbors=int(np.sqrt(len(x_train))),
                              metric='euclidean')

#euclidean,manhattan,mikowski
#fit the model in the data and predict the values

model_KNN.fit(x_train,y_train)
y_pred=model_KNN.predict(x_test)
print(list(zip(y_test,y_pred)))
```

```
[(0.05, 0.06564705882352939), (1.51, 0.33141176470588235), (0.3
5, 0.3001176470588235), (0.06, 0.10858823529411762), (0.04, 0.1
1670588235294117), (1.83, 0.9389411764705883), (0.14, 0.1712941
1764705885), (0.02, 0.2625882352941176), (2.15, 1.8952941176470
588), (0.04, 0.13305882352941176), (1.04, 0.2167058823529412),
(0.19, 0.11694117647058823), (0.76, 0.19211764705882356), (1.8
6, 0.9079999999999999), (3.83, 1.2437647058823527), (0.03, 0.33
45882352941177), (0.11, 0.1281176470588235), (0.05, 0.13), (4.4
8, 2.3058823529411763), (0.03, 0.1855294117647059), (0.02, 0.06
470588235294117), (4.8, 3.33764705882353), (1.67, 0.34223529411
76471), (0.06, 0.2890588235294118), (1.96, 1.4108235294117648),
(0.03, 0.2542352941176471), (0.02, 0.1698823529411764 8), (0.04,
0.20035294117647057), (1.64, 1.0268235294117651), (0.72, 0.4591
7647058823524), (0.49, 0.3910588235294117), (0.83, 0.5612941176
470589), (0.28, 0.3110588235294117), (1.75, 0.647411764705882
2), (0.11, 0.1718823529411764 6), (1.88, 0.3087058823529412),
(0.21, 0.22141176470588236), (0.1, 0.34058823529411775), (0.33,
0.23905882352941182), (0.18, 0.2644705882352941), (0.29, 0.2983
529411764706), (0.99, 0.4427058823529412), (0.8, 0.281647058823
```

In [57]:

```
from sklearn.metrics import r2_score,mean_squared_error
import numpy as np

r2=r2_score(y_test,y_pred)
print("R-squared:",r2)

rmse=np.sqrt(mean_squared_error(y_test,y_pred))
print("RMSE:",rmse)

adjusted_r_squared = 1 - (1-r2)*(len(y)-1)/(len(y)-x.shape[1]-1)
print("Adj R-square:",adjusted_r_squared)
```

```
R-squared: 0.49162405113661145
RMSE: 1.4955445035415993
Adj R-square: 0.4910326297709924
```

In [ ]:

In [ ]:

In [ ]:

SVM

In [58]:

```
from sklearn.svm import SVR
#create a model
svr_model=SVR(kernel='rbf',C=100,gamma=0.001)
#fitting training data into the model
svr_model.fit(x_train,y_train)
y_pred=svr_model.predict(x_test)
# print(y_pred)
# print(list(zip(y_test,y_pred)))
```

In [59]:

```
from sklearn.metrics import r2_score,mean_squared_error
import numpy as np

r2=r2_score(y_test,y_pred)
print("R-squared:",r2)

rmse=np.sqrt(mean_squared_error(y_test,y_pred))
print("RMSE:",rmse)

adjusted_r_squared = 1 - (1-r2)*(len(y)-1)/(len(y)-x.shape[1]-1)
print("Adj R-square:",adjusted_r_squared)
```

```
R-squared: 0.7526831045351958
RMSE: 1.0431187121888073
Adj R-square: 0.7523953873519116
```

LINEAR REGRESSION

In [60]:

```
x_lr=trained_data.iloc[:,1:-1]
y_lr=trained_data.iloc[:,-1]
```

In [61]:

```
## LOG TRANSFORMATION
import numpy as np
Y_log=np.log(y_lr) ## this transformation is used when the data is skewed
y_lr=Y_log
```

In [62]:

```
from sklearn.preprocessing import StandardScaler
scaler= StandardScaler()
scaler.fit(x_lr)
x_lr=scaler.transform(x)
```

In [63]:

```
print(x_lr)
```

```
[[-1.701196   -0.98347007 -1.77510868 ... -0.23402332 -2.33774151
  -0.83311946]
 [-1.9400398  -1.42204097 -1.77510868 ... -0.23633918 -2.33534851
  -0.83311946]
 [-1.701196   -1.27585067 -1.77510868 ... -0.2309645  -2.33774151
  -0.83311946]
 ...
 [-2.0196544  -1.42204097 -1.77844301 ... -0.23633918 -2.33534851
  -0.83311946]
 [-1.66935015 -0.98347007 -1.78491552 ... -0.23633918 -2.33534851
  -0.83311946]
 [-1.79673352 -1.64132642 -1.77985519 ... -0.23633918 -2.33534851
  -0.83311946]]
```

In [64]:

```
from sklearn.model_selection import train_test_split

#split the data into  the test and train
x_train_lr, x_test_lr, y_train_lr, y_test_lr=train_test_split(x_lr,y_lr,test_siz

## test_size is 20%  for less than 1000 obsevation and greater than 1000 30%
```

In [65]:

```
print(x_train_lr.shape)
print(y_train_lr.shape)
print(x_test_lr.shape)
print(y_test_lr.shape)
```

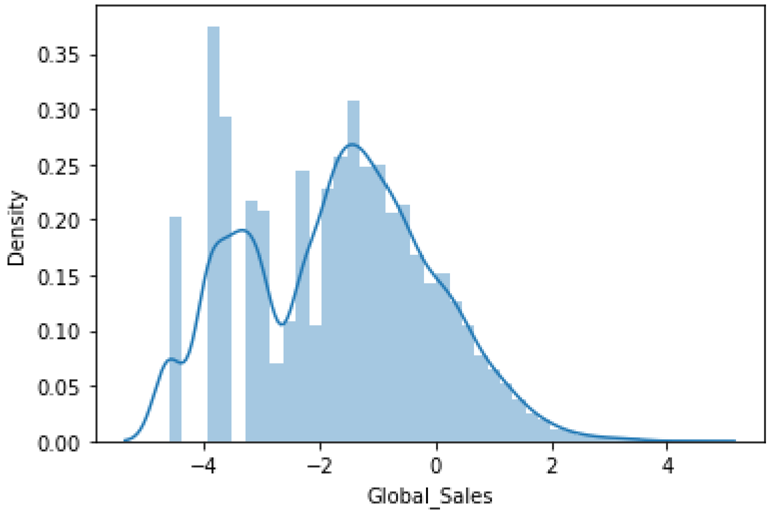
```
(7229, 12)
(7229,)
(3099, 12)
(3099,)
```

In [66]:

```
sns.distplot(y_lr,hist=True)
```

Out[66]:

```
<AxesSubplot:xlabel='Global_Sales', ylabel='Density'>
```



In [67]:

```
from sklearn.linear_model import LinearRegression
#create a model object
lm=LinearRegression()
#train the model object
lm.fit(x_train_lr,y_train_lr)

# print intercept and coefficeint
print(lm.intercept_)
print(lm.coef_)
```

```
-142.27366208577962
[ 1.21348807e-01  3.77827835e-01  8.00597604e+00  4.00469158e-01
  2.10508516e-01  8.18385836e-02  1.32723846e+00  2.54012388e+00
 -9.44637631e-03 -3.63604934e+00 -7.29676933e+01  2.36105487e-01]
```

In [68]:

```
y_pred_lr=lm.predict(x_test_lr)
print(y_pred_lr)
```

```
[-2.29078583 -1.49012158 -1.29537402 ... -2.19464291 -2.2727513
 -2.06391913]
```

In [69]:

```
from sklearn.metrics import r2_score,mean_squared_error
import numpy as np

r2=r2_score(y_test_lr,y_pred_lr)
print("R-squared:",r2)

rmse=np.sqrt(mean_squared_error(y_test_lr,y_pred_lr))
print("RMSE:",rmse)

adjusted_r_squared = 1 - (1-r2)*(len(y_lr)-1)/(len(y)-x_lr.shape[1]-1)
print("Adj R-square:",adjusted_r_squared)
```

```
R-squared: 0.27635876624529265
RMSE: 1.3535624993079574
Adj R-square: 0.2755169150766008
```

Tuning the Model Using Ridge

In [70]:

```
from sklearn.linear_model import Ridge

lm = Ridge()

lm.fit(x_train_lr,y_train_lr)

print(lm.intercept_)
print(lm.coef_)
```

```
11.326368206101835
[ 0.14129622  0.43177382  1.15656462  0.407137    0.21872365  0.076
51706
  1.3140562   2.16568907 -0.00835881  0.08243046 -2.10621363  0.383
60332]
```

In [71]:

```
from sklearn.metrics import r2_score,mean_squared_error
import numpy as np

r2=r2_score(y_test_lr,y_pred_lr)
print("R-squared:",r2)

rmse=np.sqrt(mean_squared_error(y_test_lr,y_pred_lr))
print("RMSE:",rmse)

adjusted_r_squared = 1 - (1-r2)*(len(y_lr)-1)/(len(y)-x_lr.shape[1]-1)
print("Adj R-square:",adjusted_r_squared)
```

```
R-squared: 0.27635876624529265
RMSE: 1.3535624993079574
Adj R-square: 0.2755169150766008
```

LOGISTIC REGRESSSION

In [72]:

```
x_log=trained_data.values[:,1:-1]
y_log=trained_data.values[:,-1]
```

In [73]:

```
from sklearn import preprocessing
from sklearn import utils

#convert y values to categorical values
lab = preprocessing.LabelEncoder()
y_transformed = lab.fit_transform(y)

#view transformed values
print(y_transformed)
```

```
[594 593 592 ...   0   0   0]
```

In [74]:

```
from sklearn.preprocessing import StandardScaler
scaler= StandardScaler()
scaler.fit(x_log)
x_log=scaler.transform(x_log)
```

In [75]:

```
from sklearn.model_selection import train_test_split

#split the data into  the test and train
x_train_log, x_test_log, y_train_log, y_test_log=train_test_split(x_log,y_transf

## test_size is 20%  for less than 1000 obsevation and greater than 1000 30%
```

In [76]:

```
from sklearn.linear_model import LogisticRegression
#create a model
logisticRegr = LogisticRegression()
#fitting training data into the model
logisticRegr.fit(x_train_log,y_train_log)
y_pred_log=logisticRegr.predict(x_test_log)
#print(y_pred_log)
#print(list(zip(y_test_log,y_pred_log)))
```

In [77]:

```
from sklearn.metrics import r2_score,mean_squared_error
import numpy as np

r2=r2_score(y_test_log,y_pred_log)
print("R-squared:",r2)

rmse=np.sqrt(mean_squared_error(y_test_log,y_pred_log))
print("RMSE:",rmse)

adjusted_r_squared = 1 - (1-r2)*(len(y_log)-1)/(len(y_log)-x_log.shape[1]-1)
print("Adj R-square:",adjusted_r_squared)
```

```
R-squared: 0.8239604727975907
RMSE: 40.22354042422719
Adj R-square: 0.8237556764499
```

TESTING DATA

In [ ]:

```

```

In [78]:

```
test=test_data.values[:,1:]
test=scaler.transform(test)
```

In [79]:

```
from sklearn.ensemble import GradientBoostingRegressor

model_GradientBoosting=GradientBoostingRegressor(n_estimators=250,random_state=1



##fit the model in the data and predict the values

model_GradientBoosting.fit(x_train,y_train)
```

Out[79]:

```
GradientBoostingRegressor(n_estimators=250, random_state=10)
```

In [80]:

```
test_pred=model_GradientBoosting.predict(test)
test_pred.shape
```

Out[80]:

```
(1988,)
```

In [81]:

```
test_data.columns
```

Out[81]:

```
Index(['Name', 'Platform', 'Genre', 'Publisher', 'NA_Sales', 'EU_Sa
les',
       'JP_Sales', 'Critic_Score', 'Critic_Count', 'User_Score', 'U
ser_Count',
       'Developer', 'Rating'],
      dtype='object')
```

In [82]:

```
test_data=test_data.drop(columns=['Platform', 'Genre', 'Publisher','Genre','Deve
       'JP_Sales', 'Critic_Score', 'Critic_Count', 'User_Score', 'User_Count',
       'Rating'], axis=1)
```

In [83]:

```
test_data['Global sales']=test_pred
test_data.head()
```

Out[83]:

	Name	Global sales
0	Nicktoons: MLB	0.145948
1	Shonen Jump's One Piece: Grand Battle	0.122503
2	Learn Math	0.145694
3	Nitrobike	0.157818
4	Cruise Ship Vacation Games	0.154838

In [84]:

```
test_data.shape
```

Out[84]:

(1988, 2)

In [ ]:

```
```

In [85]:

```
test_data.to_csv('Global_Sales_For_Test_Data_dt.csv',header=True,index=None)
```

WE CAN SEE ABOVE THAT THE DATA IS TRAINED ON ALL THE REGRESSOR ALGORITHMS BUT WE ARE PREDICTING OR TESTING TEST DATA ON GRADIENT BOOSTING REGRESSOR AS ONE DECISION TREE IS WEAK LEARNER BUT MULTIPLE DECISION TREE MAKES STRONG LEARNER HENCE EVEN AFTER LOWEST RMSE AND HIGHEST R SQUARE ON DECISION TREE WE ARE USING GRADIENT BOOSTING AS THE MODEL IS TRAINED ENOUGH ON IT AND THERE IS NO MAJOR DIFFERENCE BETWEEN RMSE AND R SQUARE WHEN COMPARED WITH DECISION TREE.

In [ ]:

```
```

In [ ]:

```
```