Group project-Video games sales rating

Niranjan Poudle(Niranjan111@hotmail.com),Tom Wilde(Tomwilde35@gmail.com), Wyatt Feuz (wyatt.feuz@aggiemail.usu.edu)

Importing some of the initial packages

In [1]:

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

Out[1]:

```
<function seaborn.rcmod.set(context='notebook', style
='darkgrid', palette='deep', font='sans-serif', font_s
cale=1, color_codes=True, rc=None)>
```

Reading the dataset

In [2]:

```
rand_state=1000
```

In [3]:

```
data = pd.read_csv(r"C:\Users\Niranjan.p\Desktop\Home work and assi
gnments\machine learning\Group project\Video_Games_Sales_as_at_22_D
ec_2016.csv")
```

Now lets take a brief look into the data

In [4]:

```
data.head(5)
```

Out[4]:

	Name	Platform	Year_of_Release	Genre	Publisher	NA
0	Wii Sports	Wii	2006.0	Sports	Nintendo	
1	Super Mario Bros.	NES	1985.0	Platform	Nintendo	
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	

In [5]:

```
data.tail(5)
```

Out[5]:

	Name	Platform	Year_of_Release	Genre	Publis
16714	Samurai Warriors: Sanada Maru	PS3	2016.0	Action	Tecmo K
16715	LMA Manager 2007	X360	2006.0	Sports	Codemas
16716	Haitaka no Psychedelica	PSV	2016.0	Adventure	Idea Fac
16717	Spirits & Spells	GBA	2003.0	Platform	Wana
16718	Winning Post 8 2016	PSV	2016.0	Simulation	Tecmo K

Lets do some further inspection of the data

In [6]:

```
data.shape
```

Out[6]:

```
(16719, 16)
```

In [7]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16719 entries, 0 to 16718
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Name             16717 non-null  object 
 1   Platform         16719 non-null  object 
 2   Year_of_Release  16450 non-null  float64
 3   Genre            16717 non-null  object 
 4   Publisher        16665 non-null  object 
 5   NA_Sales         16719 non-null  float64
 6   EU_Sales         16719 non-null  float64
 7   JP_Sales         16719 non-null  float64
 8   Other_Sales      16719 non-null  float64
 9   Global_Sales     16719 non-null  float64
 10  Critic_Score     8137 non-null   float64
 11  Critic_Count     8137 non-null   float64
 12  User_Score       10015 non-null  object 
 13  User_Count       7590 non-null   float64
 14  Developer        10096 non-null  object 
 15  Rating           9950 non-null   object 
dtypes: float64(9), object(7)
memory usage: 2.0+ MB
```

In [8]:

```
data.describe(include='all').T
```

Out[8]:

	count	unique	top	freq	mean	s
Name	16717	11562	Need for Speed: Most Wanted	12	NaN	Na
Platform	16719	31	PS2	2161	NaN	Na
Year_of_Release	16450	NaN	NaN	NaN	2006.49	5.878
Genre	16717	12	Action	3370	NaN	Na
Publisher	16665	581	Electronic Arts	1356	NaN	Na
NA_Sales	16719	NaN	NaN	NaN	0.26333	0.8135
EU_Sales	16719	NaN	NaN	NaN	0.145025	0.5032
JP_Sales	16719	NaN	NaN	NaN	0.0776021	0.3088
Other_Sales	16719	NaN	NaN	NaN	0.0473318	0.186
Global_Sales	16719	NaN	NaN	NaN	0.533543	1.547
Critic_Score	8137	NaN	NaN	NaN	68.9677	13.93
Critic_Count	8137	NaN	NaN	NaN	26.3608	18.98
User_Score	10015	96	tbd	2425	NaN	Na
User_Count	7590	NaN	NaN	NaN	162.23	561.2
Developer	10096	1696	Ubisoft	204	NaN	Na
Rating	9950	8	E	3991	NaN	Na

In the first part we will do some visualization of the data, so we will not remove Na value because we might remove some data we might visualize we will remove na data befor analysis

First lets try to visualize based on year of release

Lets visualize the year of release and sales

In [9]:

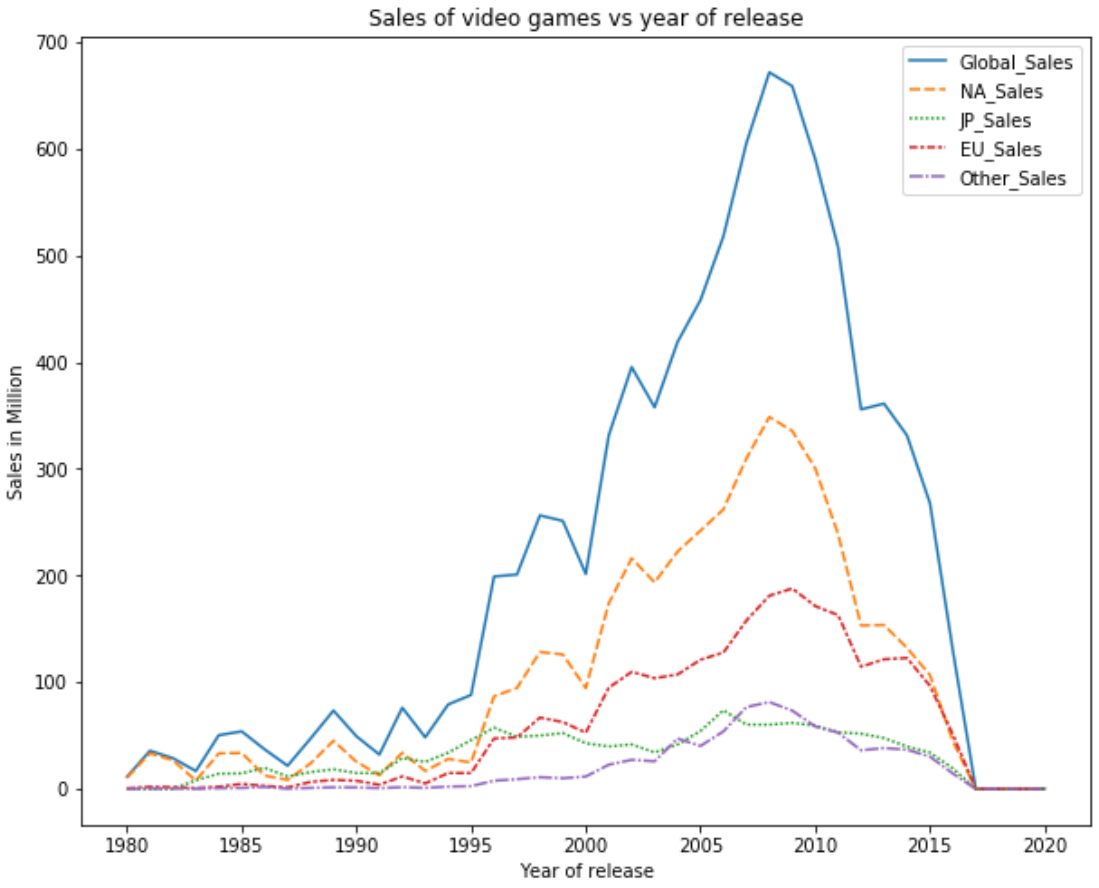
```
#### Extraxting only the required colums as grouping as necessary
temp = data[['Year_of_Release','Global_Sales','NA_Sales','JP_Sales'
,'EU_Sales','Other_Sales']]
temp = temp.groupby('Year_of_Release').sum()# To get the sum of the
sales value
```

In [10]:

```
### Plotting differnt types of sales based on the year of release
plt.figure(figsize=(10,8))
sns.lineplot(data=temp)
plt.ylabel('Sales in Million')
plt.xlabel('Year of release')
plt.title('Sales of video games vs year of release')
```

Out[10]:

```
Text(0.5, 1.0, 'Sales of video games vs year of releas
e')
```



We notice from the above plot that the global sales or sales in all parts were high from the period around 2005-2010, tentative.

Our data have the game released, by aggegrating the number of games released lets look at global sales vs number of games released

In [11]:

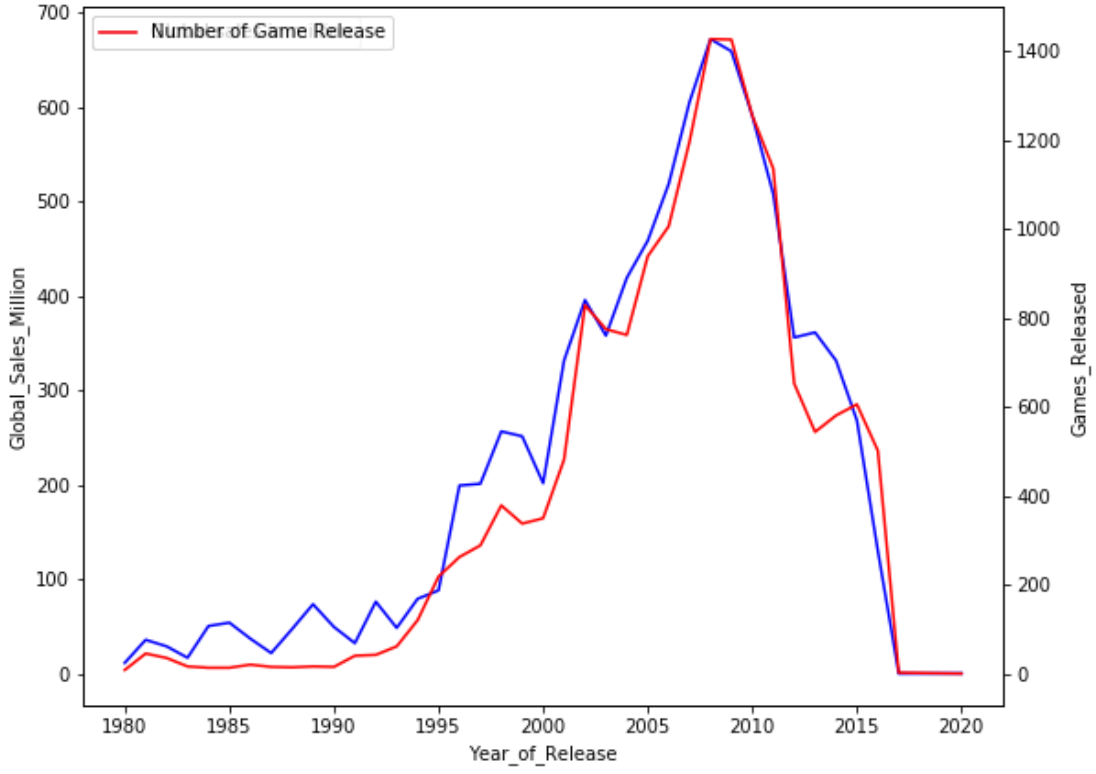
```
temp = data[['Year_of_Release','Global_Sales']] ### extracting nece
ssary colums
temp =temp.groupby('Year_of_Release')['Global_Sales'].agg(Global_Sa
les_Million='sum', Games_Released='count').reset_index()
### Above code does the sum of the global sales and adds the number
of releases each year
```

In [12]:

```
### lets plot these two things together number of release and globa
l sales
plt.figure(figsize=(9,7))
sns.lineplot(data=temp,x='Year_of_Release',y='Global_Sales_Million'
, color='b', label="Global sales in million")
ax=plt.twinx()
sns.lineplot(data=temp,x='Year_of_Release',y='Games_Released', colo
r='r', label="Number of Game Release",ax=ax)
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1872846498
8>
```



It can be seen that number of global sale is high when there were more games released or may be more games released more sales

Next lets visualized based on some top platform for playing games

There are high number of platform so reducing them to top 20 will all less frequent ones as other

In [13]:

```
#temporary data to visulaize based on platform
temp=data['Platform'].value_counts() ### Adding all the games relea
sed of same category
temp1=temp[:19].index ### Select top 19 platform with high frequenc
y
data['Platform'] = data['Platform'].where(data['Platform'].isin(tem
p1),'Other')# less frequent as other
```

We want to visualize global sales based on platform and number of games released side by side

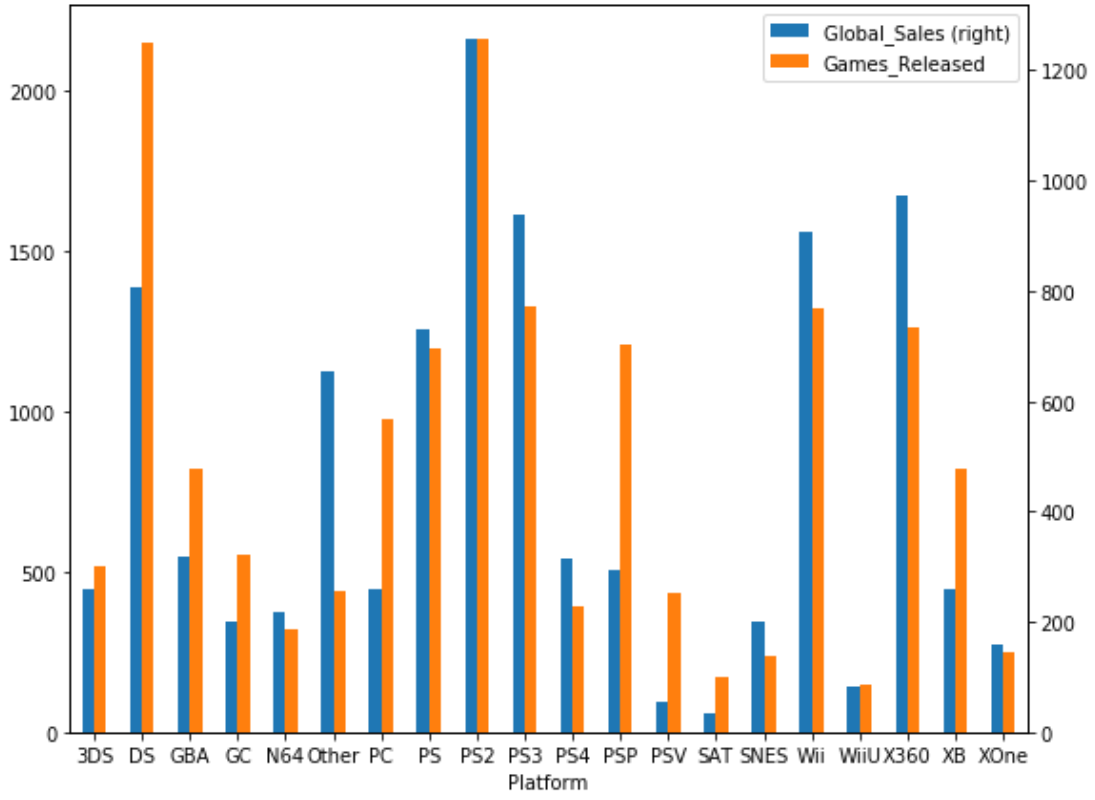
In [14]:

```
# Selecting platform and Global sales and doing sum of global sales
and counting games released per Platform
temp = data[['Platform','Global_Sales']]
temp =temp.groupby('Platform')['Global_Sales'].agg(Global_Sales='su
m', Games_Released='count')
```

Plotting Global sales and the number of games released side by side

In [15]:

```
temp.plot( kind= 'bar' , secondary_y= 'Global_Sales' , rot= 0,figsi
ze=(9,7) )
plt.show()
```



Ps2 platform had the number of games released and sale high or in porportion, DS has high number of games released but relatively less number of sales. Some notable mentions X360, PS3 and DS

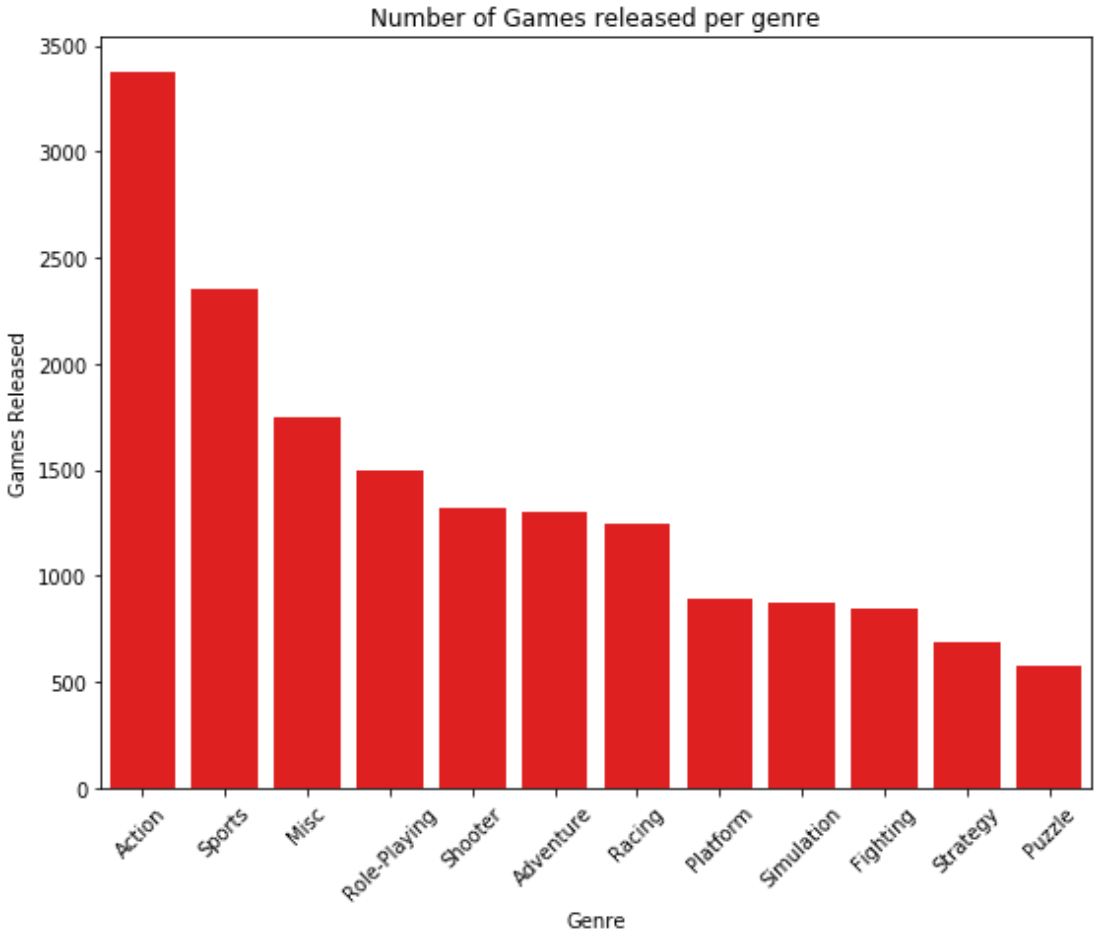
Lets visualize some data based on genre

In [16]:

```
### Barplot of genre versus number of games released
plt.figure(figsize=(9,7))
sns.countplot(x=data['Genre'], order = data['Genre'].value_counts()
.index,color='r')
plt.ylabel('Games Released')
plt.title('Number of Games released per genre')
plt.xticks(rotation=45)
```

Out[16]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 1
1]),
 <a list of 12 Text xticklabel objects>)
```



High number of games released related to action, sports genre compared to other

Next lets see the number of global sales based on genre

In [17]:

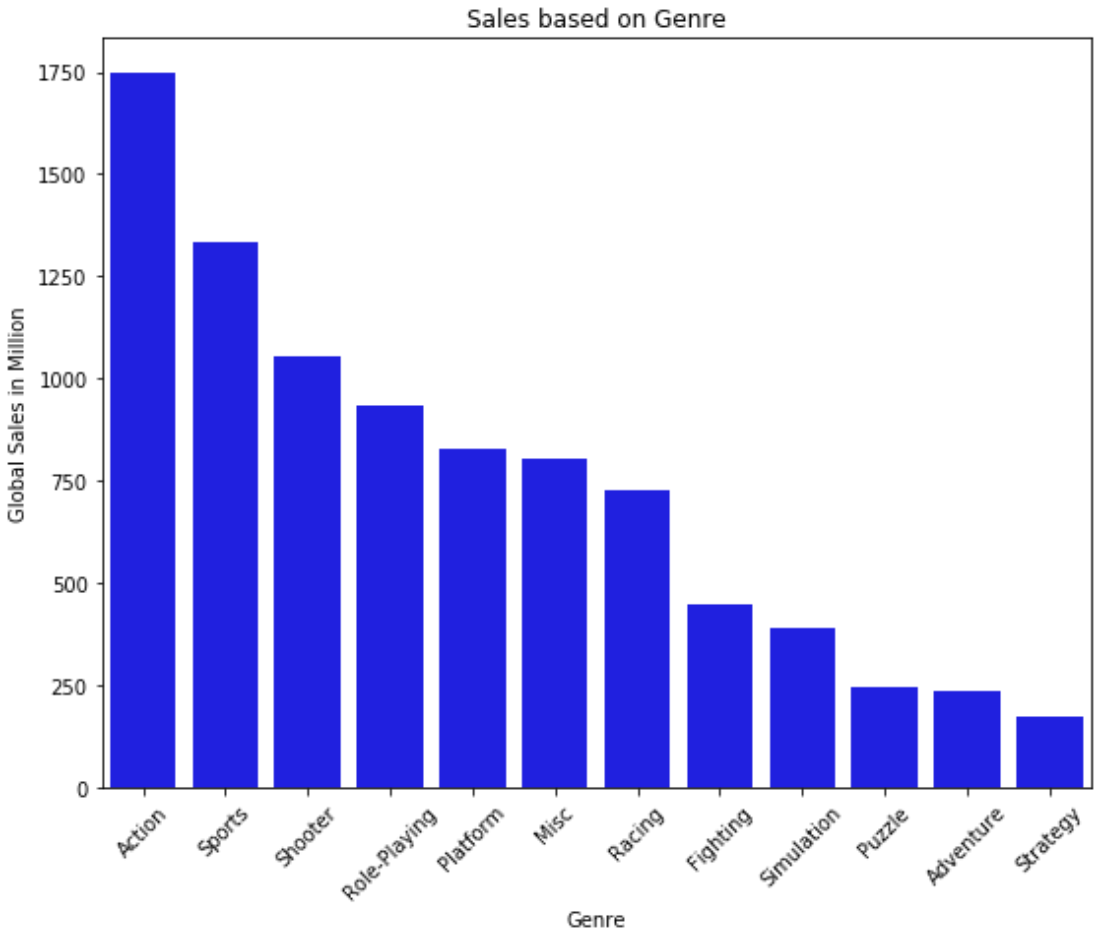
```
## we need to add up the global sales so creating temporary files
temp = data[['Genre','Global_Sales']]
temp = temp.groupby('Genre').sum().reset_index() ### adding global
 sales based on genre
temp= temp.sort_values('Global_Sales',ascending=False) # Ascending
 order for good barplot
```

In [18]:

```
#Barplot of globa sales based on genre
plt.figure(figsize=(9,7))
sns.barplot(x=temp['Genre'],y=temp['Global_Sales'],color='b')
plt.ylabel('Global Sales in Million')
plt.title('Sales based on Genre')
plt.xticks(rotation=45)
```

Out[18]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 1
1]),
 <a list of 12 Text xticklabel objects>)
```


Sales based on Genre
Global Sales in Million
1750
1500
1250
1000
750
500
250
0
Action
Sports
Shooter
Role-Playing
Platform
Misc
Racing
Fighting
Simulation
Puzzle
Adventure
Strategy
Genre

More number of games released for action and sports and more global sales, Interstingly enough shooter and role playing also have high number of sales even though less games released

Lets try to visulasize based on publisher

In [19]:

```
data['Publisher'].describe()
```

Out[19]:

```
count                16665
unique                 581
top       Electronic Arts
freq                  1356
Name: Publisher, dtype: object
```

From the describtion above there are more than 581 publisher, we will just look into 30 high frequency publisher

In [20]:

```
### Decreasing the number of categories for making analysis simple
temp=data['Publisher'].value_counts() ### Counts each category of d
ata
temp1=temp[:29].index ### Select top 29 frequency
data['Publisher'] = data['Publisher'].where(data['Publisher'].isin(
temp1),'Other') ## mark other as 'other'
```

In [21]:

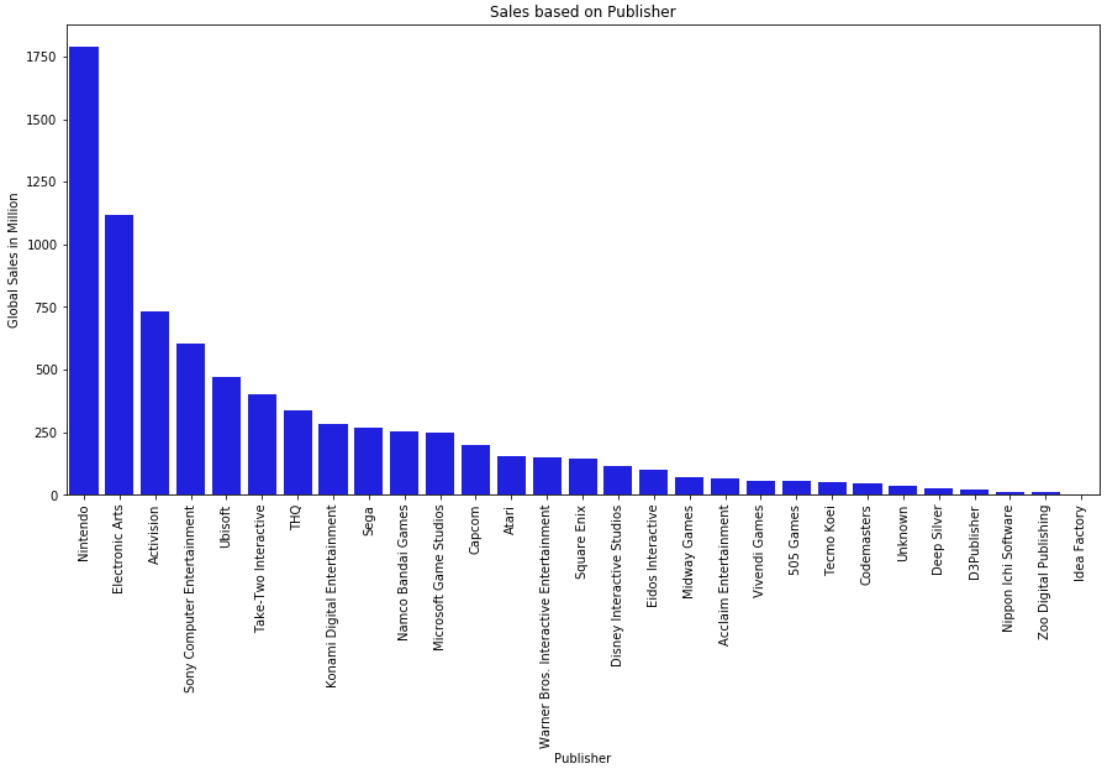
```
### temporary files for the purpose of visulaization
temp=data[data['Publisher']!='Other'] # Also removing Other for vis
ualization only
temp=temp[['Global_Sales','Publisher']] ## Sales and publisher data
subset
temp = temp.groupby('Publisher').sum().reset_index() ## Adding glob
al sales by publisher
temp=temp.sort_values('Global_Sales',ascending=False) ## ascending
 order for visualization
```

In [22]:

```
### Plotting the number of sales based on publisher
plt.figure(figsize=(15,7))
sns.barplot(x=temp['Publisher'],y=temp['Global_Sales'],color='b')
plt.ylabel('Global Sales in Million')
plt.title('Sales based on Publisher')
plt.xticks(rotation=90)
```

Out[22]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 1
1, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 2
8]),
 <a list of 29 Text xticklabel objects>)
```



Nintendo, electronic arts , Activision has highest number of sales globally

In [23]:

```
### Lets look at the number of sales per number of games released b
y publisher: kind of average
```

In [24]:

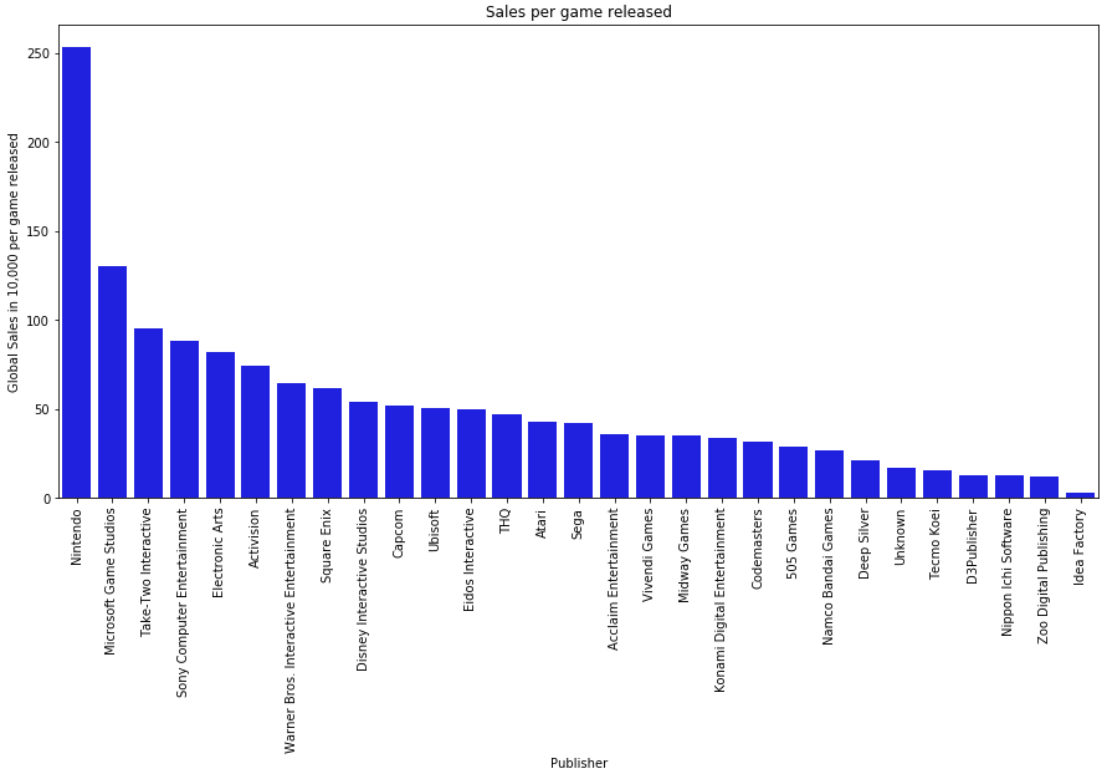
```
temp=data[data['Publisher']!='Other'] ### removing other less frequ
ent publishers
temp=temp[['Global_Sales','Publisher']] ## Temporary subset
temp =temp.groupby('Publisher')['Global_Sales'].agg(Global_Sales='s
um', Games_Released='count').reset_index()
## The line above add the global sales and counts the number per pu
blisher
temp['Sales_10000_per_game_released']=temp['Global_Sales']*100/temp
['Games_Released']
#Sales in 10,000 per the number of games released (Per million give
s low number)
temp=temp.sort_values('Sales_10000_per_game_released',ascending=Fal
se)
```

In [25]:

```
### plotting sales in 10,000 per the number of games released by publisher
plt.figure(figsize=(15,7))
sns.barplot(x=temp['Publisher'],y=temp['Sales_10000_per_game_released'],color='b')
plt.ylabel('Global Sales in 10,000 per game released')
plt.title('Sales per game released')
plt.xticks(rotation=90)
```

Out[25]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]),
 <a list of 29 Text xticklabel objects>)
```



Lets see top 10 games released based on total global sales

In [26]:

```
### Notable metioned of released gaes which have highest sales
data.sort_values('Global_Sales', ascending=False).head(10).Name
```

Out[26]:

```
0                    Wii Sports
1             Super Mario Bros.
2                Mario Kart Wii
3             Wii Sports Resort
4     Pokemon Red/Pokemon Blue
5                        Tetris
6         New Super Mario Bros.
7                     Wii Play
8    New Super Mario Bros. Wii
9                   Duck Hunt
Name: Name, dtype: object
```

Next lets see if we can somehow visulaize critic and user scores

Lets see the relation between user score and critic score

In [27]:

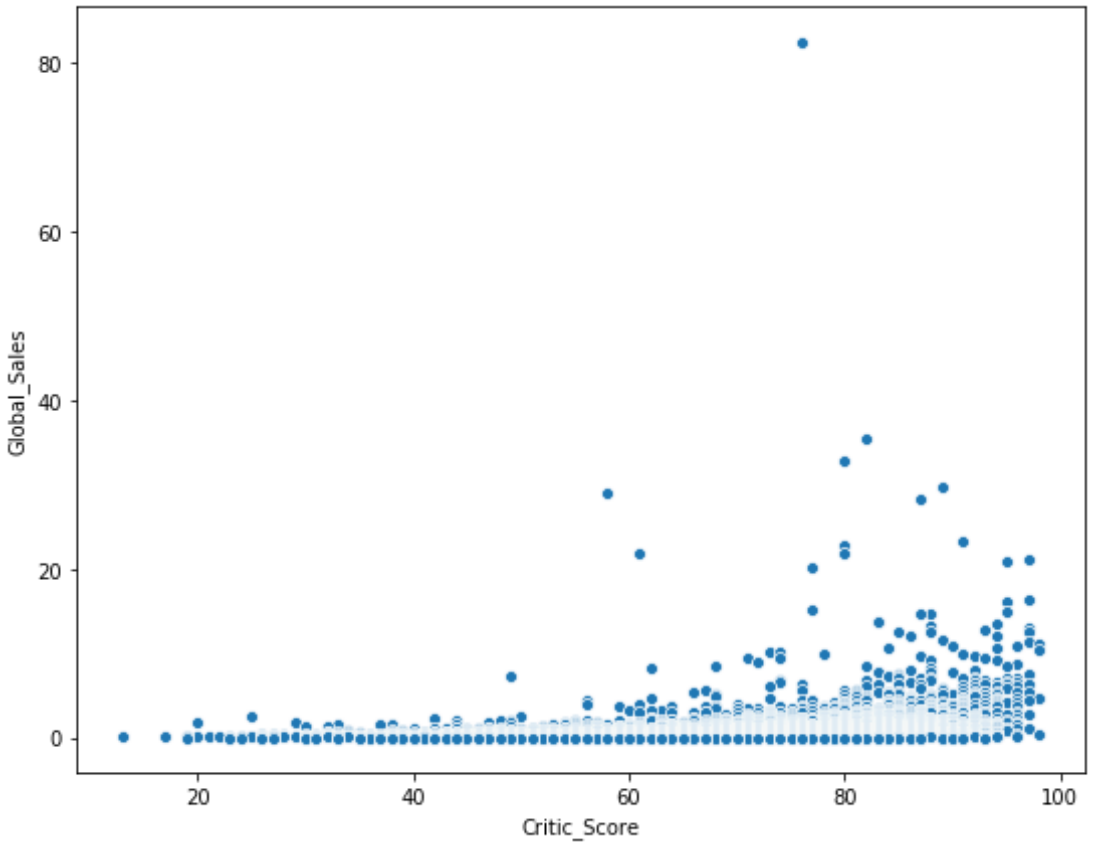
```
### Lets create a temporary files of subset
temp=data[['User_Score','Critic_Score','Global_Sales','Platform','G
enre']]
temp=temp.dropna() ## To plot scatter plot lets just remove na of n
umeric values
```

In [28]:

```
### Scatterplot of critic score vs global sales
plt.figure(figsize=(9,7))
sns.scatterplot(x=temp['Critic_Score'],y=temp['Global_Sales'])
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1872e5bb40
8>
```



Looks like good critic score means high global sales in general, but still there are some games with good critic score and less sales as well, gneral increasing trend in average

Lets look at user score and critic score as well

There are some user scores with tbd and its making the category as string type, lets remove them and look at relationship

In [29]:

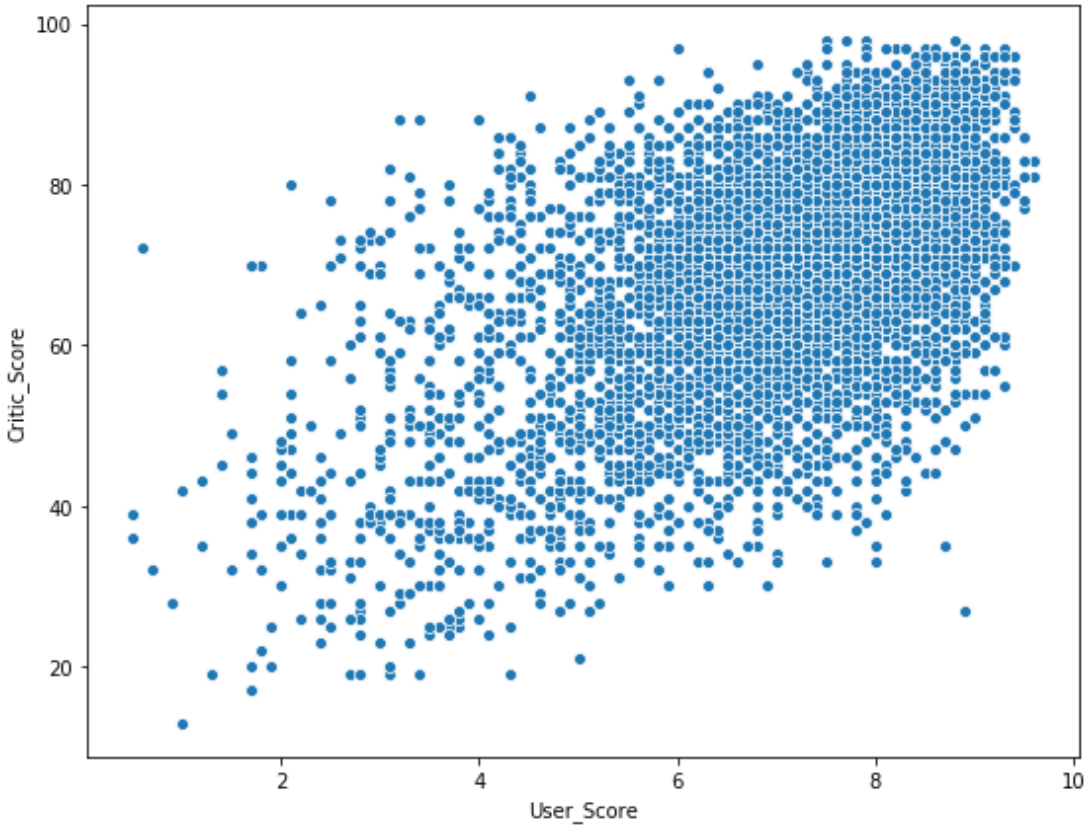
```
temp=temp[temp['User_Score']!='tbd'] ## to be determinde user score
s
temp['User_Score']=temp['User_Score'].astype('float')## making the
 variable numeric
```

In [30]:

```
##Scatterplot of user and critic score
plt.figure(figsize=(9,7))
sns.scatterplot(x=temp['User_Score'],y=temp['Critic_Score'])
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1872e8515c
8>
```



On average critic score increases user score also increases although there are some with complete opposite of one another, so the relation of user score and slaes is also somewhat correlated

Next we shall look at average critic score for game release over plat form

In [31]:

```
## Sum of slaes over platform and counting games released per platf
orm
temp1 =temp.groupby('Platform')['Critic_Score'].agg(Critic_Score='s
um', Games_Released='count').reset_index()
```

In [32]:

```
#Averaging the critic score per platform
temp1['Average_score_Platform']=temp1['Critic_Score']/temp1['Games_
Released']
```

In [33]:

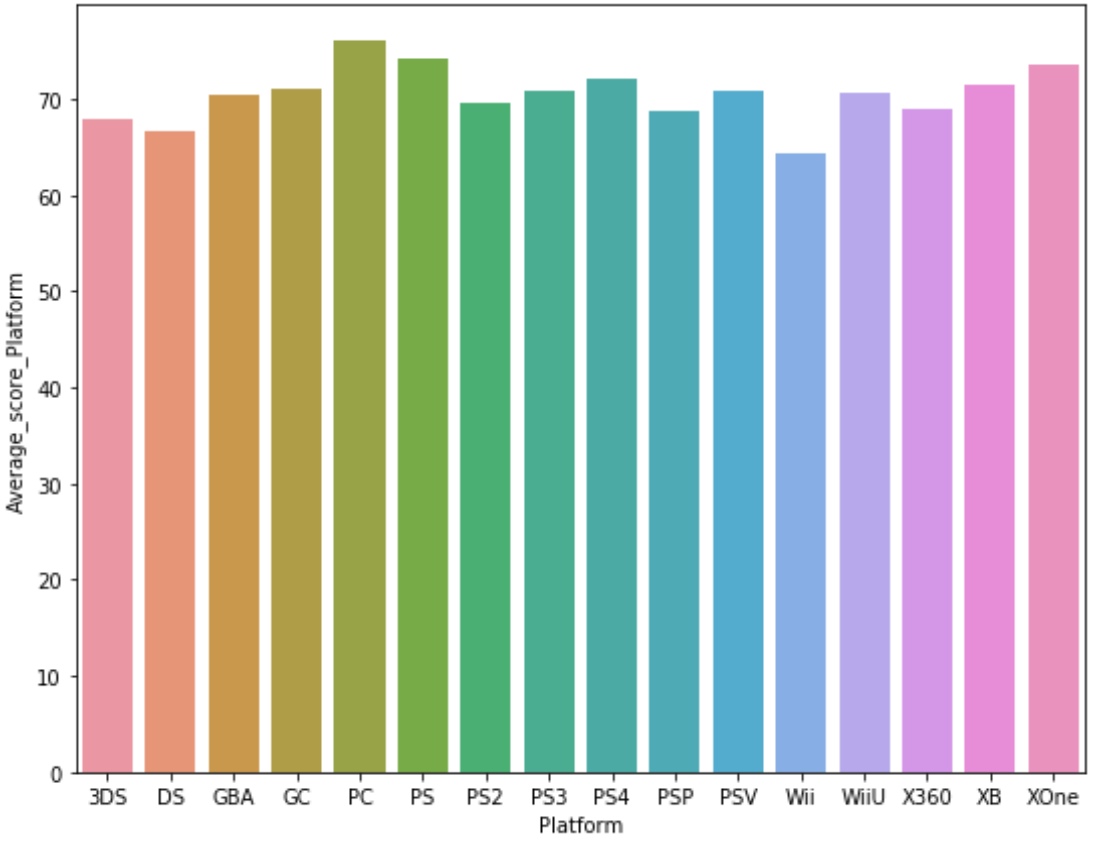
```
temp1=temp1[temp1['Platform']!='Other'] ### Not using the other pla
tform not frequent one
```

In [34]:

```
### barplot for the critic score across game released by platform
plt.figure(figsize=(9,7))
sns.barplot(x=temp1['Platform'],y=temp1['Average_score_Platform'])
```

Out[34]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1872e20b54
8>
```



Not much of the intuiative visualization as expected

We have done enough visualization, now lets manipulate and remove na (for analysis purposes)

Looks like infering or extrapolating data might not work well so we remove Na in data

In [35]:

```
data.shape
```

Out[35]:

```
(16719, 16)
```

In [36]:

```
data=data.dropna()
data.shape
```

Out[36]:

```
(6826, 16)
```

Lets look at data descrption

In [37]:

```
data.describe(include='all').T
```

Out[37]:

	count	unique	top	freq	mean	std
Name	6826	4378	Need for Speed: Most Wanted	8	NaN	NaN
Platform	6826	17	PS2	1140	NaN	NaN
Year_of_Release	6826	NaN	NaN	NaN	2007.44	4.21116
Genre	6826	12	Action	1630	NaN	NaN
Publisher	6826	30	Other	1128	NaN	NaN
NA_Sales	6826	NaN	NaN	NaN	0.394435	0.967323
EU_Sales	6826	NaN	NaN	NaN	0.236069	0.687282
JP_Sales	6826	NaN	NaN	NaN	0.0641488	0.28755
Other_Sales	6826	NaN	NaN	NaN	0.0826648	0.269853
Global_Sales	6826	NaN	NaN	NaN	0.777499	1.96331
Critic_Score	6826	NaN	NaN	NaN	70.2687	13.8704
Critic_Count	6826	NaN	NaN	NaN	28.9311	19.2228
User_Score	6826	89	7.8	294	NaN	NaN
User_Count	6826	NaN	NaN	NaN	174.748	587.389
Developer	6826	1289	EA Canada	149	NaN	NaN
Rating	6826	7	T	2378	NaN	NaN

Lets change some variable as categorical variable

In [38]:

```
data['Platform']=data['Platform'].astype('category')
data['Genre']=data['Genre'].astype('category')
data['Publisher']=data['Publisher'].astype('category')
data['Developer']=data['Developer'].astype('category')
data['Rating']=data['Rating'].astype('category')
```

Lets convert user score in numeric variable

There was variable user score 'tbd' previously lets check if any left

In [39]:

```
temp=data[data['User_Score']=='tbd']
```

In [40]:

```
temp.shape
```

Out[40]:

```
(0, 16)
```

All are removed with na removal lets convert to numeric

In [41]:

```
data['User_Score']=data['User_Score'].astype(float)
```

In [42]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6826 entries, 0 to 16706
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Name             6826 non-null   object
 1   Platform         6826 non-null   category
 2   Year_of_Release  6826 non-null   float64
 3   Genre            6826 non-null   category
 4   Publisher        6826 non-null   category
 5   NA_Sales         6826 non-null   float64
 6   EU_Sales         6826 non-null   float64
 7   JP_Sales         6826 non-null   float64
 8   Other_Sales      6826 non-null   float64
 9   Global_Sales     6826 non-null   float64
 10  Critic_Score     6826 non-null   float64
 11  Critic_Count     6826 non-null   float64
 12  User_Score       6826 non-null   float64
 13  User_Count       6826 non-null   float64
 14  Developer        6826 non-null   category
 15  Rating           6826 non-null   category
dtypes: category(5), float64(10), object(1)
memory usage: 733.0+ KB
```

In [43]:

```
data.describe(include='all').T
```

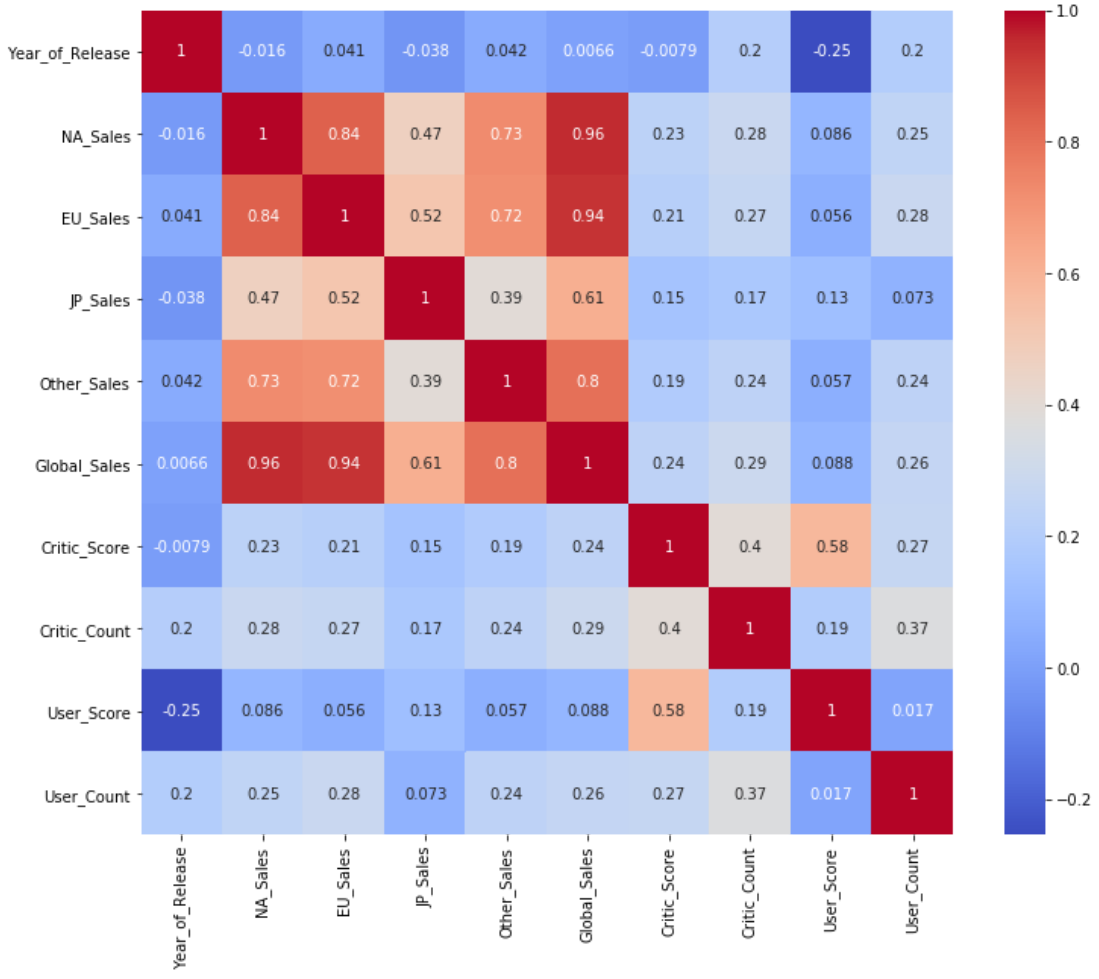
Out[43]:

	count	unique	top	freq	mean	std
Name	6826	4378	Need for Speed: Most Wanted	8	NaN	NaN
Platform	6826	17	PS2	1140	NaN	NaN
Year_of_Release	6826	NaN	NaN	NaN	2007.44	4.21116
Genre	6826	12	Action	1630	NaN	NaN
Publisher	6826	30	Other	1128	NaN	NaN
NA_Sales	6826	NaN	NaN	NaN	0.394435	0.967323
EU_Sales	6826	NaN	NaN	NaN	0.236069	0.687282
JP_Sales	6826	NaN	NaN	NaN	0.0641488	0.28755
Other_Sales	6826	NaN	NaN	NaN	0.0826648	0.269853
Global_Sales	6826	NaN	NaN	NaN	0.777499	1.96331
Critic_Score	6826	NaN	NaN	NaN	70.2687	13.8704
Critic_Count	6826	NaN	NaN	NaN	28.9311	19.2228
User_Score	6826	NaN	NaN	NaN	7.18501	1.44073
User_Count	6826	NaN	NaN	NaN	174.748	587.389
Developer	6826	1289	EA Canada	149	NaN	NaN
Rating	6826	7	T	2378	NaN	NaN

Machine learning can take any type of data so we do not further edit just look at correlation and some overall plots

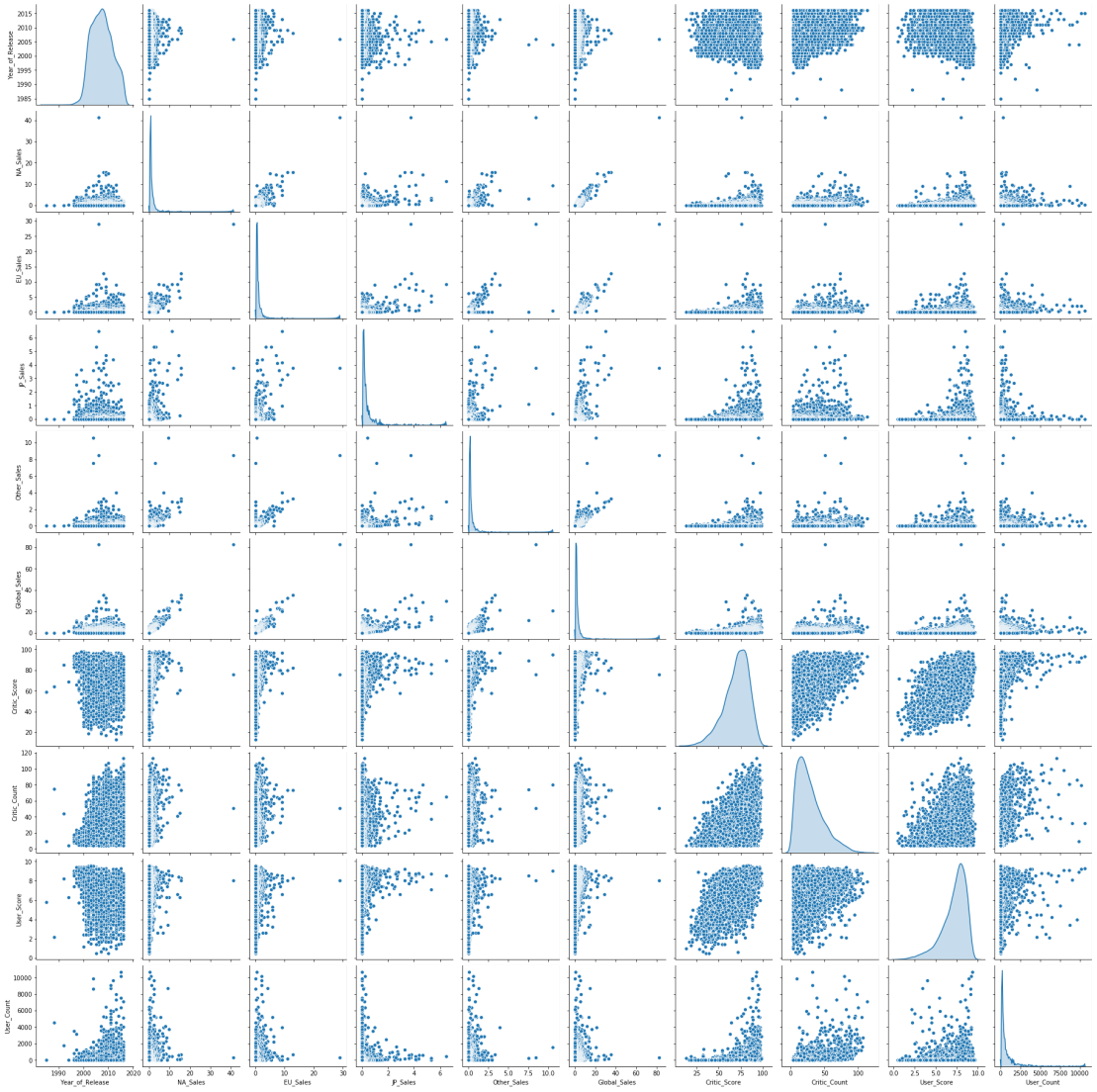
In [44]:

```
#### Coreelation heatmap
plt.figure(figsize=(12,10))
sns.heatmap(data.corr(), cmap='coolwarm',annot=True)
plt.show()
```



In [45]:

```
sns.pairplot(data, diag_kind='kde');
```



Lets us procced for the linear regression analysis

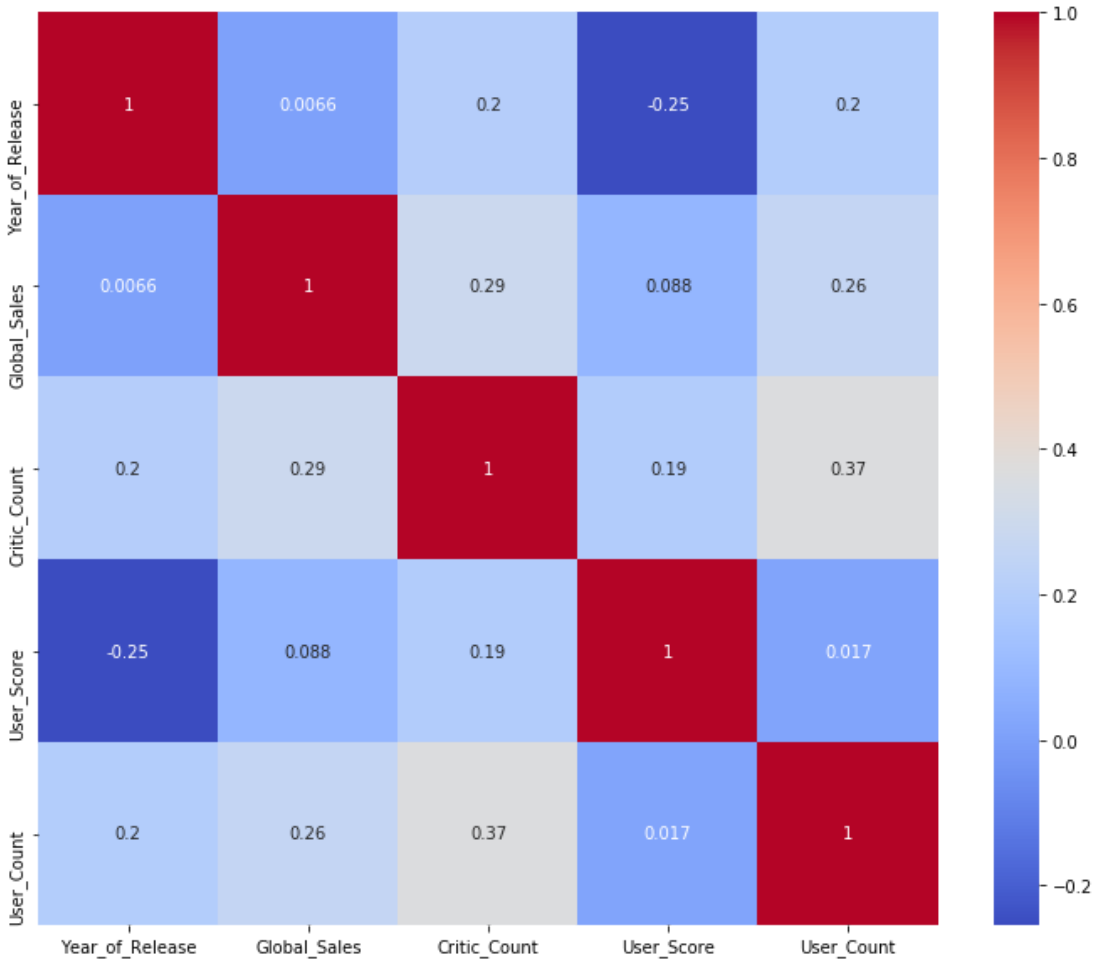
Lets remove some of the observation which are correlated with each other in the first step and the Name Variabe which is unique and developer is also extra large we can drop it.

In [46]:

```
data.drop(['NA_Sales','EU_Sales','Other_Sales','JP_Sales','Critic_Score','Name'],axis=1,inplace=True)
```

In [47]:

```
plt.figure(figsize=(12,10))
sns.heatmap(data.corr(), cmap='coolwarm',annot=True)
plt.show()
```



Now looking at uncorrelated data to proceed

In [48]:

```
data.describe(include='all')
```

Out[48]:

	Platform	Year_of_Release	Genre	Publisher	Global_Sales
count	6826	6826.000000	6826	6826	6826.000000
unique	17	NaN	12	30	NaN
top	PS2	NaN	Action	Other	NaN
freq	1140	NaN	1630	1128	NaN
mean	NaN	2007.437299	NaN	NaN	0.777499
std	NaN	4.211160	NaN	NaN	1.963313
min	NaN	1985.000000	NaN	NaN	0.010000
25%	NaN	2004.000000	NaN	NaN	0.110000
50%	NaN	2007.000000	NaN	NaN	0.290000
75%	NaN	2011.000000	NaN	NaN	0.750000
max	NaN	2016.000000	NaN	NaN	82.530000

In [49]:

```
data.head(15)
```

Out[49]:

	Platform	Year_of_Release	Genre	Publisher	Global_Sales	Cı
0	Wii	2006.0	Sports	Nintendo	82.53	
2	Wii	2008.0	Racing	Nintendo	35.52	
3	Wii	2009.0	Sports	Nintendo	32.77	
6	DS	2006.0	Platform	Nintendo	29.80	
7	Wii	2006.0	Misc	Nintendo	28.92	
8	Wii	2009.0	Platform	Nintendo	28.32	
11	DS	2005.0	Racing	Nintendo	23.21	
13	Wii	2007.0	Sports	Nintendo	22.70	
14	X360	2010.0	Misc	Microsoft Game Studios	21.81	
15	Wii	2009.0	Sports	Nintendo	21.79	
16	PS3	2013.0	Action	Take-Two Interactive	21.04	
17	PS2	2004.0	Action	Take-Two Interactive	20.81	
19	DS	2005.0	Misc	Nintendo	20.15	
23	X360	2013.0	Action	Take-Two Interactive	16.27	
24	PS2	2002.0	Action	Take-Two Interactive	16.15	

In [50]:

```
data1 = data.copy() ### Data for other random forest regression ana
lysis which does not require log-transformations
data2 =data.copy() #### Data for KNN
data3= data.copy() ### Data for SVM
```

In [51]:

```
#The inclusion of Developer makes the data more unstable leading to
e^9 MSE in linear regression so revoving it for linear regression m
odel only
data = data.drop(['Developer'],axis=1)
```

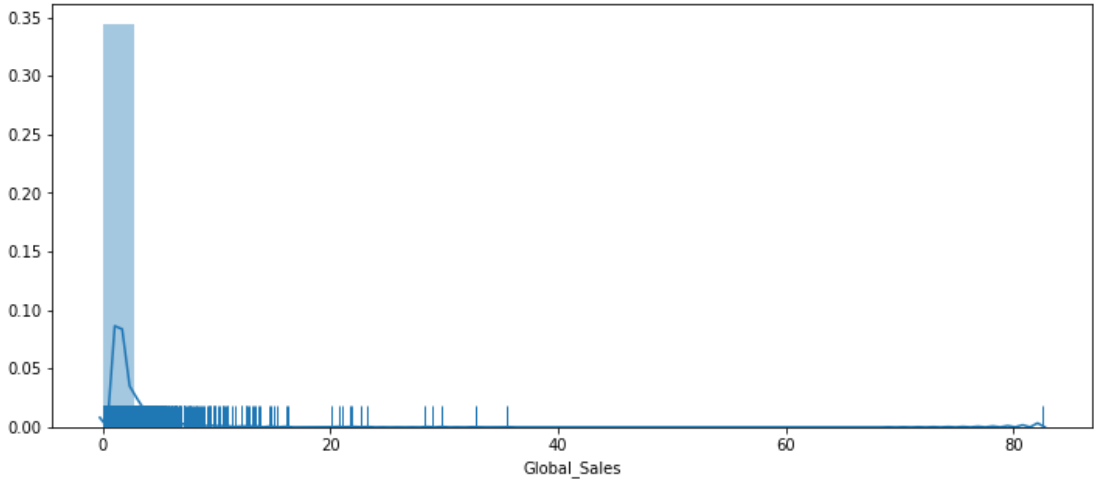
The following procedure is for for all the regression

In [52]:

```
### Lets look at the y-varaible
```

In [53]:

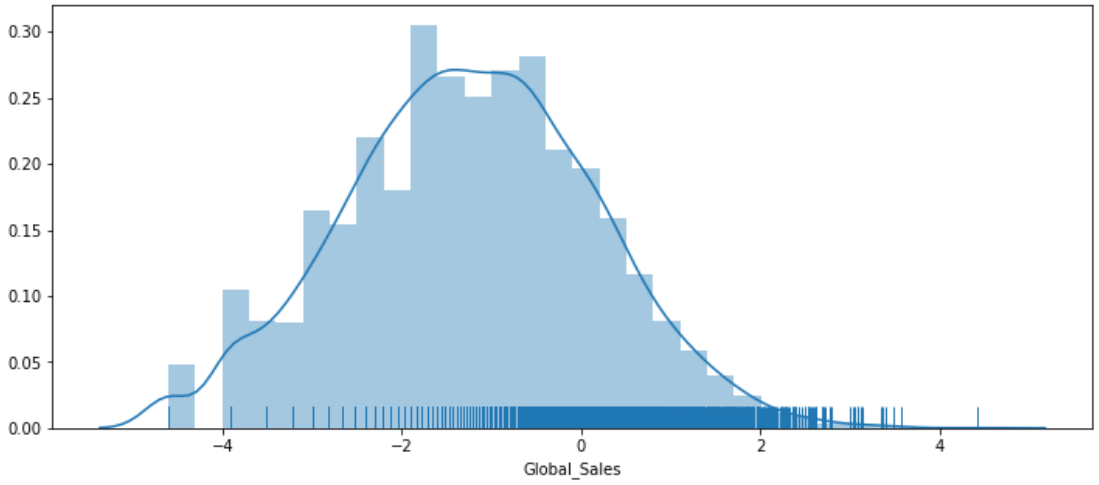
```
plt.figure(figsize=(12,5))
sns.distplot(data['Global_Sales'], bins=30,rug=True)
plt.show()
```



We need to log -transform the y-variable

In [54]:

```
plt.figure(figsize=(12,5))
data['Global_Sales']=np.log(data['Global_Sales'])
sns.distplot(data['Global_Sales'],bins=30, rug= True)
plt.show()
```



In [55]:

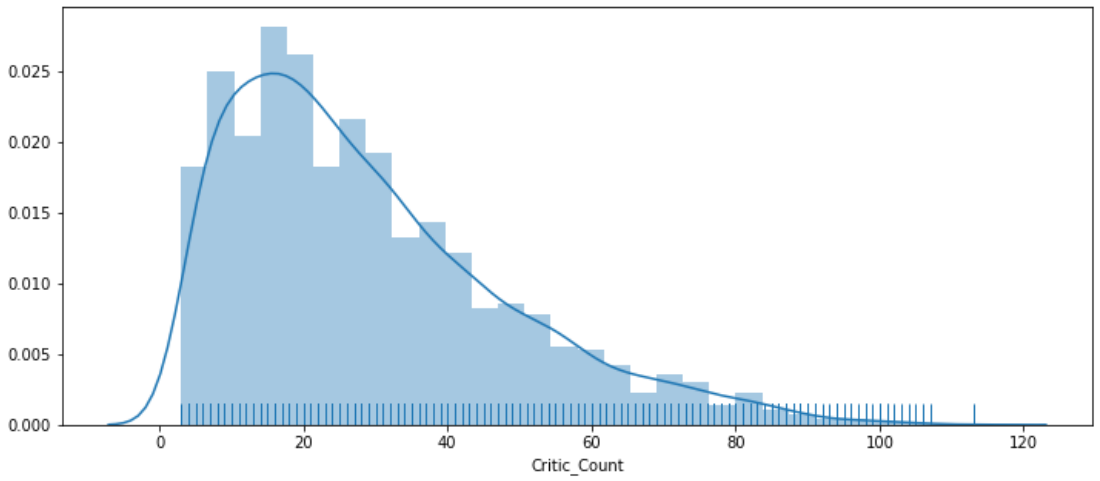
```
### Lets Remove some outliers
data = data[data['Global_Sales']<3.5]
```

The data looks fairly fine with only some outliers but still looks good enough and my intituion is not to remove the y-variable easily

Lets look at variables in feature space as well

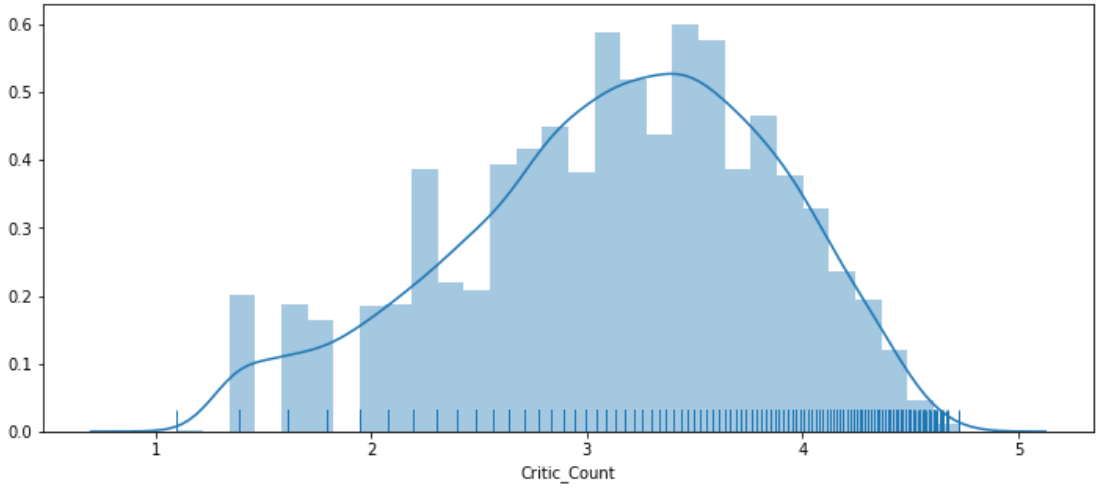
In [56]:

```
## first Critic_Count
plt.figure(figsize=(12,5))
sns.distplot(data['Critic_Count'], bins=30,rug=True)
plt.show()
```



In [57]:

```
plt.figure(figsize=(12,5))
data['Critic_Count']=np.log(data['Critic_Count'])
sns.distplot(data['Critic_Count'],bins=30, rug= True)
plt.show()
```



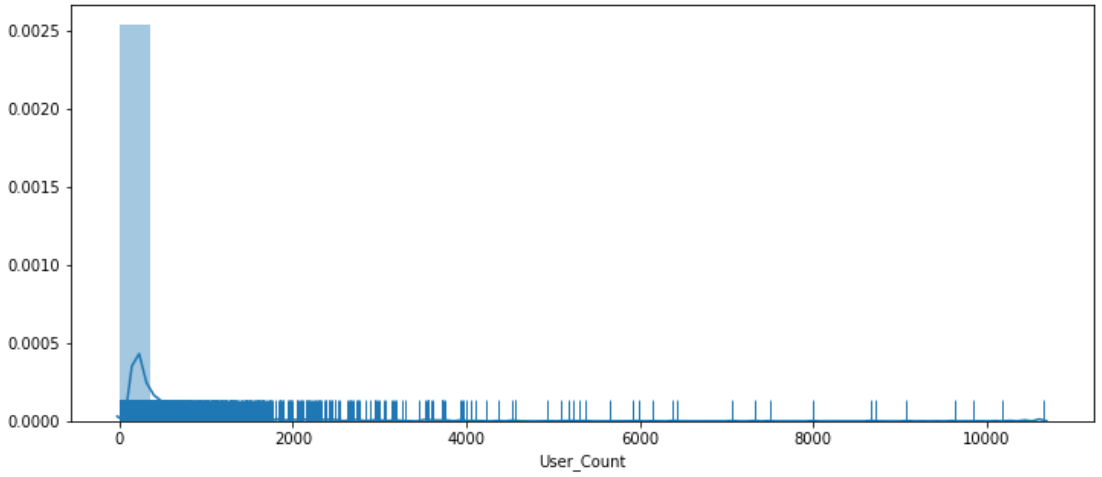
In [58]:

```
### Looks good after log transsformation
```

Next for user count

In [59]:

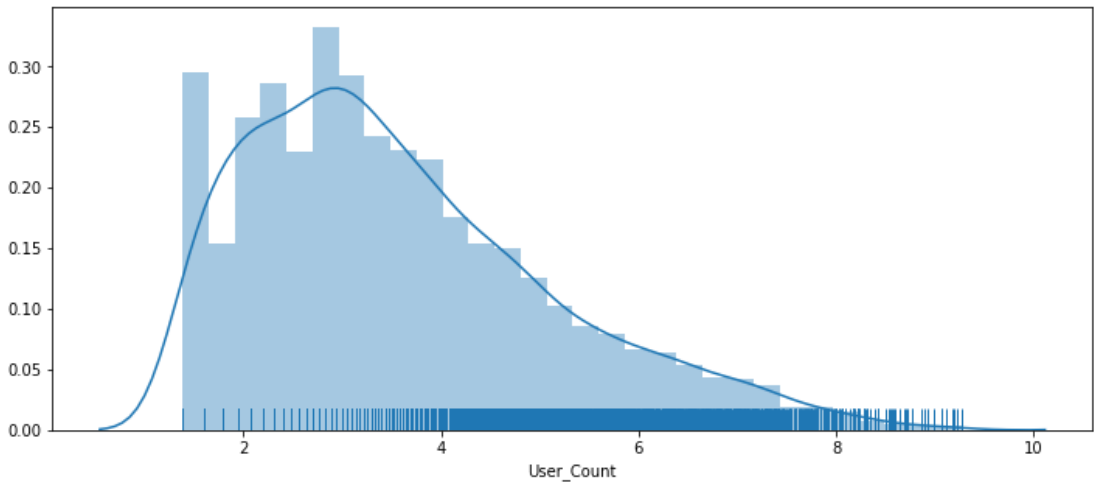
```
plt.figure(figsize=(12,5))
sns.distplot(data['User_Count'], bins=30,rug=True)
plt.show()
```



There seems to be alot of outliers lets log-transformmed the data

In [60]:

```
plt.figure(figsize=(12,5))
data['User_Count']=np.log(data['User_Count'])
sns.distplot(data['User_Count'],bins=30, rug= True)
plt.show()
```



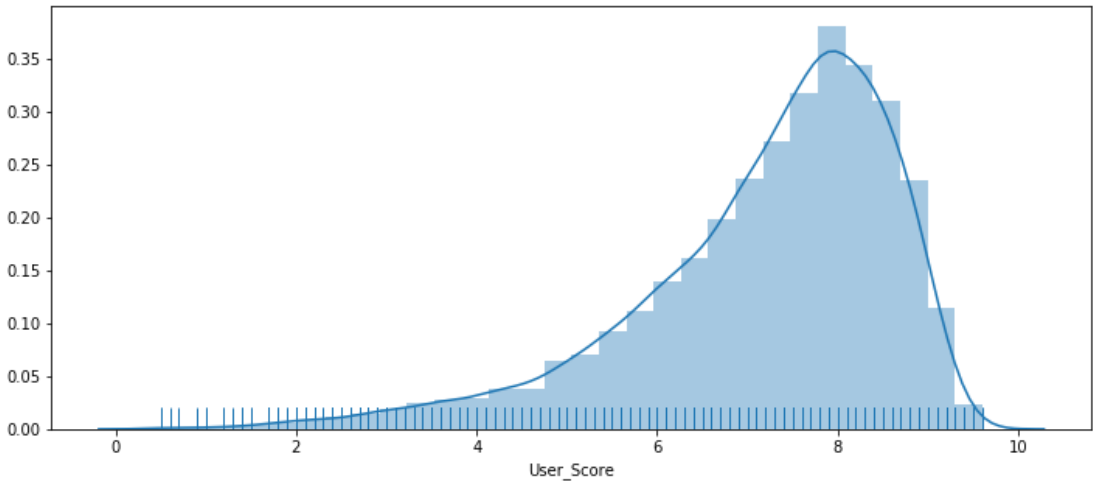
In [61]:

```
### Lets remove some observations which are clear outlier
data = data[data['User_Count']<9.5]
```

Next let us look at the User_Score variable

In [62]:

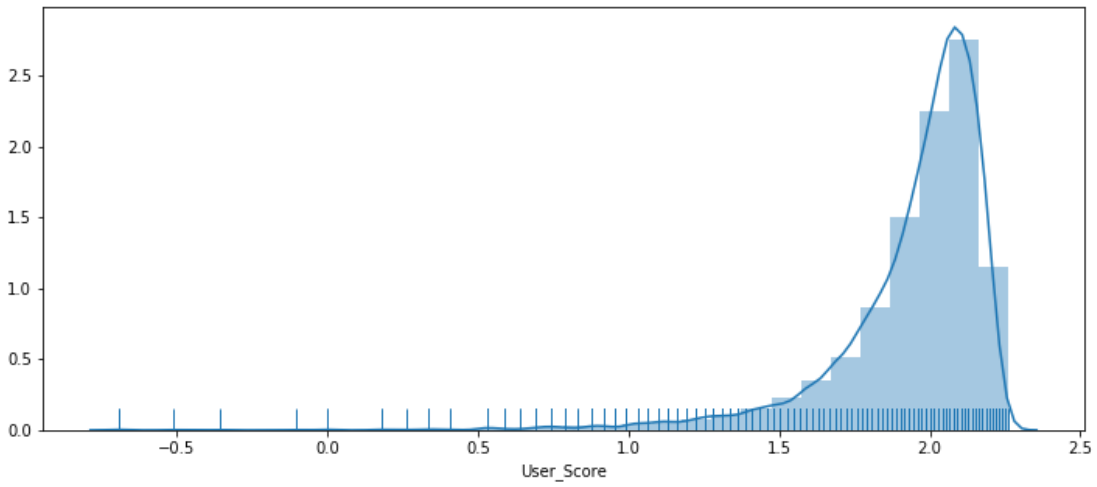
```
plt.figure(figsize=(12,5))
sns.distplot(data['User_Score'], bins=30,rug=True)
plt.show()
```



User Score are also some what skewed lets log-transform them as well

In [63]:

```
plt.figure(figsize=(12,5))
data['User_Score']=np.log(data['User_Score'])
sns.distplot(data['User_Score'],bins=30, rug= True)
plt.show()
```



In [64]:

```
### Lets us remove some outliers
data = data[data['User_Score']>0.5]
```

In [65]:

```
data.describe(include='all')
```

Out[65]:

	Platform	Year_of_Release	Genre	Publisher	Global_Sales
count	6809	6809.000000	6809	6809	6809.000000
unique	17	NaN	12	30	NaN
top	PS2	NaN	Action	Other	NaN
freq	1140	NaN	1628	1124	NaN
mean	NaN	2007.435306	NaN	NaN	-1.232203
std	NaN	4.211606	NaN	NaN	1.396467
min	NaN	1985.000000	NaN	NaN	-4.605170
25%	NaN	2004.000000	NaN	NaN	-2.207275
50%	NaN	2007.000000	NaN	NaN	-1.237874
75%	NaN	2011.000000	NaN	NaN	-0.287682
max	NaN	2016.000000	NaN	NaN	3.489513

In [66]:

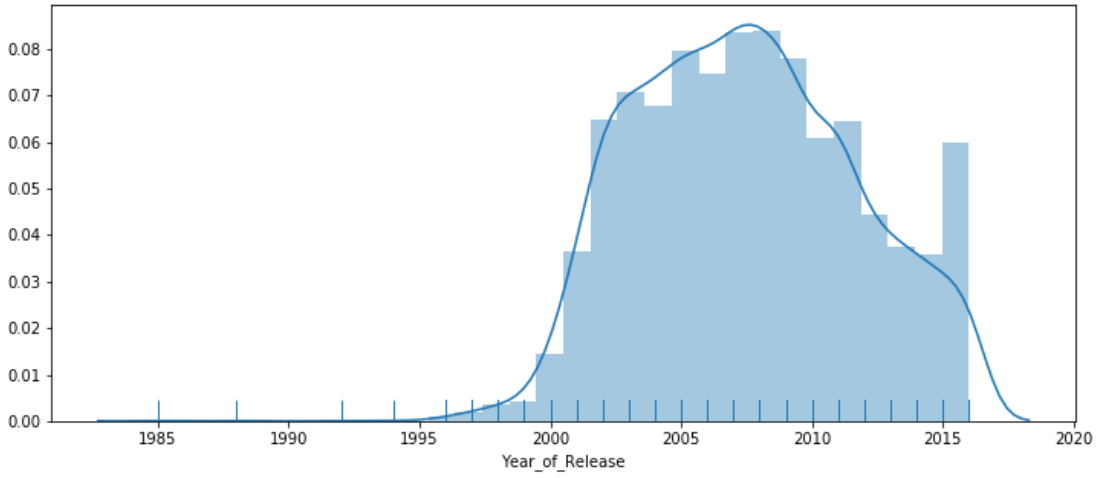
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6809 entries, 3 to 16706
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype   
---  ------           --------------  -----   
 0   Platform         6809 non-null   category
 1   Year_of_Release  6809 non-null   float64 
 2   Genre            6809 non-null   category
 3   Publisher        6809 non-null   category
 4   Global_Sales     6809 non-null   float64 
 5   Critic_Count     6809 non-null   float64 
 6   User_Score       6809 non-null   float64 
 7   User_Count       6809 non-null   float64 
 8   Rating           6809 non-null   category
dtypes: category(4), float64(5)
memory usage: 348.8 KB
```

lets lokk at the year variable

In [67]:

```
plt.figure(figsize=(12,5))
sns.distplot(data['Year_of_Release'], bins=30,rug=True)
plt.show()
```



Lets create some dummy variables

In [68]:

```
data= pd.get_dummies(data,drop_first=True) ### Creating dummies for categorical variables
```

Defining the variables and splitiing the data

In [69]:

```
y = data['Global_Sales']
X = data.drop('Global_Sales', axis=1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.2, random_state=rand_state)
```

In [70]:

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

In [71]:

```
sklearn_reg = LinearRegression()
```

In [72]:

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

Out[72]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jo
bs=None, normalize=False)
```

In [73]:

```
data
```

Out[73]:

	Year_of_Release	Global_Sales	Critic_Count	User_Score	Us
3	2009.0	3.489513	4.290459	2.079442	
6	2006.0	3.394508	4.174387	2.140066	
7	2006.0	3.364533	3.713572	1.887070	
8	2009.0	3.343568	4.382027	2.128232	
11	2005.0	3.144583	4.158883	2.151762	
...	...	...	...	...	
16667	2001.0	-4.605170	1.386294	0.875469	
16677	2002.0	-4.605170	2.484907	2.174752	
16696	2014.0	-4.605170	2.995732	2.028148	
16700	2011.0	-4.605170	2.484907	1.757858	
16706	2011.0	-4.605170	2.484907	1.974081	

6809 rows × 67 columns

In [74]:

```
sklearn_reg.predict(X_test)
```

Out[74]:

```
array([-2.25115016, -2.45534664, -2.8293636 , ..., -1.
41666776,
       -0.99350085, -0.3899352 ])
```

Lets see some predictions

In [75]:

```
sklearn_reg.predict(X_test)[0:4]
```

Out[75]:

```
array([-2.25115016, -2.45534664, -2.8293636 , -2.41589
988])
```

In [76]:

```
y_hat_test = sklearn_reg.predict(X_test)
predictions = pd.DataFrame({'Actuals':y_test, 'Predictions':y_hat_t
est})
predictions.head()
```

Out[76]:

	Actuals	Predictions
11302	-2.525729	-2.251150
13097	-2.995732	-2.455347
8417	-1.771957	-2.829364
11649	-2.525729	-2.415900
35	2.615204	1.595068

In [77]:

```
MSE_test = round(np.mean(np.square(predictions['Actuals'] - predict
ions['Predictions'])),6)
MSE_test
```

Out[77]:

```
0.831133
```

In [78]:

```
RMSE_test = round(np.sqrt(MSE_test),6)
RMSE_test
```

Out[78]:

```
0.911665
```

Further we can look the RMSE in actual scale as well

In [79]:

```
act_pred = np.exp(predictions)
```

In [80]:

```
act_pred
```

Out[80]:

	Actuals	Predictions
11302	0.08	0.105278
13097	0.05	0.085833
8417	0.17	0.059050
11649	0.08	0.089287
35	13.67	4.928664
...	...	...
8389	0.17	0.151199
1336	1.43	0.619536
2233	0.93	0.242521
5487	0.33	0.370278
1663	1.21	0.677101

1362 rows × 2 columns

In [81]:

```
MSE_test = round(np.mean(np.square(act_pred['Actuals'] - act_pred[
'Predictions'])),6)
MSE_test
```

Out[81]:

```
1.757885
```

In [82]:

```
RMSE_test = RMSE_test = round(np.sqrt(MSE_test),6)
RMSE_test
```

Out[82]:

```
1.325853
```

Random Forest Regression Analysis

Lets convert some of our categorical variables to be used in Random Forest regression.

In [83]:

```
data1
```

Out[83]:

	Platform	Year_of_Release	Genre	Publisher	Global_Sa
0	Wii	2006.0	Sports	Nintendo	82
2	Wii	2008.0	Racing	Nintendo	35
3	Wii	2009.0	Sports	Nintendo	32
6	DS	2006.0	Platform	Nintendo	29
7	Wii	2006.0	Misc	Nintendo	28
...	...	...	...	...	
16667	GBA	2001.0	Action	Other	0
16677	GBA	2002.0	Fighting	Midway Games	0
16696	PC	2014.0	Action	Konami Digital Entertainment	0
16700	PC	2011.0	Shooter	Other	0
16706	PC	2011.0	Strategy	Unknown	0

6826 rows × 10 columns

In [84]:

```
data1['Global_Sales']=np.log(data1['Global_Sales'])
```

In [85]:

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

for col in categorical:
    data1[col] = data1[col].astype("category").cat.codes
```

In [86]:

```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6826 entries, 0 to 16706
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Platform         6826 non-null   int8   
 1   Year_of_Release  6826 non-null   float64
 2   Genre            6826 non-null   int8   
 3   Publisher        6826 non-null   int8   
 4   Global_Sales     6826 non-null   float64
 5   Critic_Count     6826 non-null   float64
 6   User_Score       6826 non-null   float64
 7   User_Count       6826 non-null   float64
 8   Developer        6826 non-null   int16  
 9   Rating           6826 non-null   int8   
dtypes: float64(5), int16(1), int8(4)
memory usage: 360.0 KB
```

In [87]:

```
y = data1['Global_Sales']
X = data1.drop(['Global_Sales'], axis=1) # becareful inplace= False

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
=0.2, random_state=rand_state)
```

In [88]:

```
from sklearn.ensemble import RandomForestRegressor
```

In [89]:

```
RF_regressor = RandomForestRegressor(n_estimators = 100, max_featur
es='sqrt', random_state=1000)
RF_regressor.fit(X_train, y_train)
```

Out[89]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, c
riterion='mse',
                      max_depth=None, max_features='sq
rt', max_leaf_nodes=None,
                      max_samples=None, min_impurity_d
ecrease=0.0,
                      min_impurity_split=None, min_sam
ples_leaf=1,
                      min_samples_split=2, min_weight_
fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, o
ob_score=False,
                      random_state=1000, verbose=0, wa
rm_start=False)
```

In [90]:

```
y_pred_test = RF_regressor.predict(X_test)
```

MSE and RMSE in log scale for Y-variable

In [91]:

```
MSE_test = round(np.mean(np.square(y_test - y_pred_test)),2)
MSE_test
```

Out[91]:

```
0.81
```

In [92]:

```
RMSE_test = round(np.sqrt(MSE_test),2)
RMSE_test
```

Out[92]:

```
0.9
```

Now lets see if we can improve this by optimizing the model.

In [93]:

```
param_grid = {'n_estimators': [10,500], 'max_features':['sqrt'], 'max_depth':[2,10,60]} ## I did a lot of search and mentioned just a few
```

In [94]:

```
from sklearn.model_selection import GridSearchCV
```

In [95]:

```
grid= GridSearchCV(RandomForestRegressor(random_state=rand_state), param_grid, refit=True, verbose=0, cv=10)
```

In [96]:

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

Out[96]:

```
GridSearchCV(cv=10, error_score=nan,
             estimator=RandomForestRegressor(bootstrap
=True, ccp_alpha=0.0,
                                             criterion
='mse', max_depth=None,
                                             max_featu
res='auto',
                                             max_leaf_
nodes=None,
                                             max_sampl
es=None,
                                             min_impur
ity_decrease=0.0,
                                             min_impur
ity_split=None,
                                             min_sampl
es_leaf=1,
                                             min_sampl
es_split=2,
                                             min_weigh
t_fraction_leaf=0.0,
                                             n_estimat
ors=100, n_jobs=None,
                                             oob_score
=False, random_state=1000,
                                             verbose=
0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [2, 10, 60], 'ma
x_features': ['sqrt'],
                         'n_estimators': [10, 500]},
             pre_dispatch='2*n_jobs', refit=True, retu
rn_train_score=False,
             scoring=None, verbose=0)
```

In [97]:

```
grid.best_params_
```

Out[97]:

```
{'max_depth': 60, 'max_features': 'sqrt', 'n_estimator
s': 500}
```

In [98]:

```
grid.best_estimator_
```

Out[98]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, c
riterion='mse',
                      max_depth=60, max_features='sqr
t', max_leaf_nodes=None,
                      max_samples=None, min_impurity_d
ecrease=0.0,
                      min_impurity_split=None, min_sam
ples_leaf=1,
                      min_samples_split=2, min_weight_
fraction_leaf=0.0,
                      n_estimators=500, n_jobs=None, o
ob_score=False,
                      random_state=1000, verbose=0, wa
rm_start=False)
```

In [99]:

```
y_pred_test_optimized = grid.predict(X_test)
```

In [100]:

```
MSE_test = round(np.mean(np.square(y_test - y_pred_test_optimized
)),2)
MSE_test
```

Out[100]:

```
0.79
```

In [101]:

```
RMSE_test = round(np.sqrt(MSE_test),2)
RMSE_test
```

Out[101]:

```
0.89
```

We were able to improve the model slightly from our un-optimized model. The test RMSE for the un-optimized was 0.81 and with the optimized model the RMSE is 0.79.

Lets look now at the feature importance.

In [102]:

```
features= list(X_train.columns)
features
```

Out[102]:

```
['Platform',
 'Year_of_Release',
 'Genre',
 'Publisher',
 'Critic_Count',
 'User_Score',
 'User_Count',
 'Developer',
 'Rating']
```

In [103]:

```
RF_regressor = RandomForestRegressor(n_estimators = 500, max_featur
es='sqrt', max_depth=60)
RF_regressor.fit(X_train, y_train)
```

Out[103]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, c
riterion='mse',
                      max_depth=60, max_features='sqr
t', max_leaf_nodes=None,
                      max_samples=None, min_impurity_d
ecrease=0.0,
                      min_impurity_split=None, min_sam
ples_leaf=1,
                      min_samples_split=2, min_weight_
fraction_leaf=0.0,
                      n_estimators=500, n_jobs=None, o
ob_score=False,
                      random_state=None, verbose=0, wa
rm_start=False)
```

In [104]:

```
importances= RF_regressor.feature_importances_
importances
```

Out[104]:

```
array([0.14231262, 0.09677104, 0.05973712, 0.08097714,
0.17042444,
       0.08223388, 0.23940676, 0.08584348, 0.0422935
1])
```

In [105]:

```
FIM = pd.DataFrame({'Features': features , 'Feature_importance':imp
ortances})
FIM=FIM.sort_values(by=['Feature_importance'])
FIM
```

Out[105]:

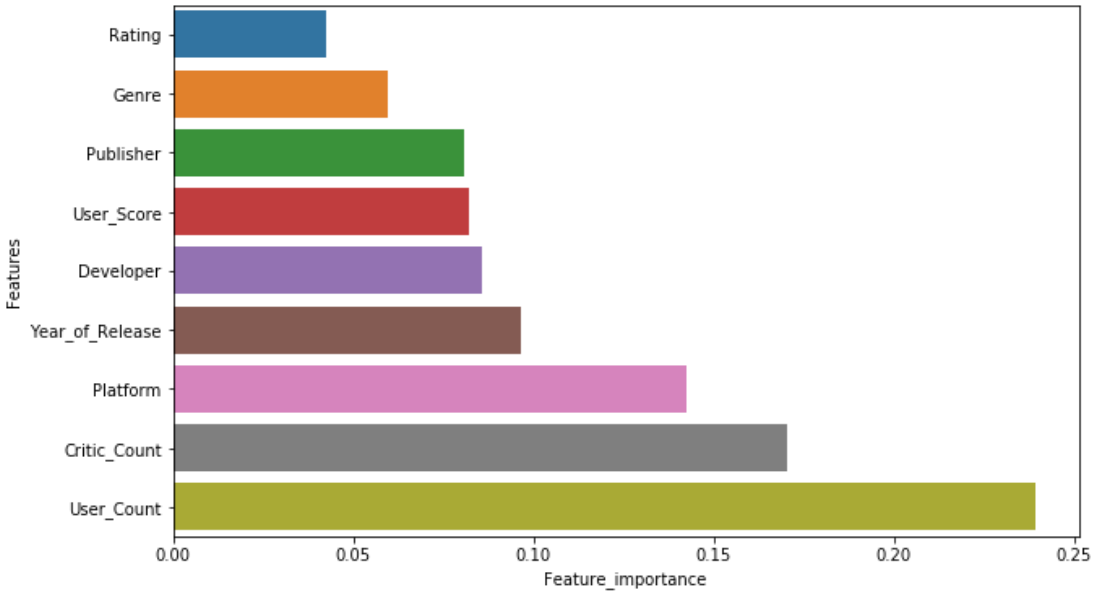
	Features	Feature_importance
8	Rating	0.042294
2	Genre	0.059737
3	Publisher	0.080977
5	User_Score	0.082234
7	Developer	0.085843
1	Year_of_Release	0.096771
0	Platform	0.142313
4	Critic_Count	0.170424
6	User_Count	0.239407

In [106]:

```
plt.figure(figsize=(10,6))
sns.barplot(y='Features', x='Feature_importance', data=FIM)
```

Out[106]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18733948a4
8>
```



I wasn't expecting User Count to be the top feature, but it isn't a total surprise either. The other most important features all make sense to me as well.

Lets look at the RMSE_test from the cross validation

In [107]:

```
from sklearn.model_selection import cross_val_score
import sklearn.metrics
```

In [108]:

```
RMSE_CV=[]
K = (5,10)
for i in K:
    NMSE = cross_val_score(estimator = RF_regressor, X = X_train, y
= y_train, cv = i , scoring="neg_mean_squared_error" )
    RMSE_CV.append(round(np.sqrt(-NMSE).mean(),2))

RMSE_CV_df = pd.DataFrame(K, columns=['K'])
RMSE_CV_df['RMSE_CV']=RMSE_CV

RMSE_CV_df
```

Out[108]:

	K	RMSE_CV
0	5	0.89
1	10	0.88

Those were the results from cross-fold five nad cross-fold 1o cross-validation.

SVR Regression

In [109]:

```
data3 ### Differnt data are used because we worked on differnt comp
uter at same time
```

Out[109]:

	Platform	Year_of_Release	Genre	Publisher	Global_Sa
0	Wii	2006.0	Sports	Nintendo	82
2	Wii	2008.0	Racing	Nintendo	35
3	Wii	2009.0	Sports	Nintendo	32
6	DS	2006.0	Platform	Nintendo	29
7	Wii	2006.0	Misc	Nintendo	28
...	...	...	...	...	
16667	GBA	2001.0	Action	Other	0
16677	GBA	2002.0	Fighting	Midway Games	0
16696	PC	2014.0	Action	Konami Digital Entertainment	0
16700	PC	2011.0	Shooter	Other	0
16706	PC	2011.0	Strategy	Unknown	0

6826 rows × 10 columns

In [110]:

```
data3 =data3.drop(['Developer'],axis=1) ### Kepping developed optio
n we could not get results in more than overnight
```

In [111]:

```
data3
```

Out[111]:

	Platform	Year_of_Release	Genre	Publisher	Global_Sa
0	Wii	2006.0	Sports	Nintendo	82
2	Wii	2008.0	Racing	Nintendo	35
3	Wii	2009.0	Sports	Nintendo	32
6	DS	2006.0	Platform	Nintendo	29
7	Wii	2006.0	Misc	Nintendo	28
...	...	...	...	...	
16667	GBA	2001.0	Action	Other	0
16677	GBA	2002.0	Fighting	Midway Games	0
16696	PC	2014.0	Action	Konami Digital Entertainment	0
16700	PC	2011.0	Shooter	Other	0
16706	PC	2011.0	Strategy	Unknown	0

6826 rows × 9 columns

In [112]:

```
data3['Global_Sales']=np.log(data3['Global_Sales'])
```

In [113]:

```
data3= pd.get_dummies(data3,drop_first=True) ### Creating dummies f
or categorical variables
```

Splitting the data

In [114]:

```
y = data3['Global_Sales']
X = data3.drop('Global_Sales', axis=1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
=0.2, random_state=rand_state)
```

Standarizing the data

In [115]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [116]:

```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
```

In [117]:

```
SVM_regression = SVR(C=1, kernel='linear')
SVM_regression.fit(X_train, y_train)
```

Out[117]:

```
SVR(C=1, cache_size=200, coef0=0.0, degree=3, epsilon=
0.1, gamma='scale',
    kernel='linear', max_iter=-1, shrinking=True, tol=
0.001, verbose=False)
```

In [118]:

```
y_pred_test_SVR = SVM_regression.predict(X_test)
```

In [119]:

```
MSE_test = round(np.mean(np.square(y_test - y_pred_test_SVR)),2)
MSE_test
```

Out[119]:

```
1.1
```

In [120]:

```
RMSE_test = round(np.sqrt(MSE_test),2)
RMSE_test
```

Out[120]:

```
1.05
```

We could not do intensive grid search due to computational limitations, our grid search gave not much inproved data.

In [121]:

```
param_grid = {'C': [1,10], 'gamma': [1,0.1], 'kernel': ['linear']}
```

In [122]:

```
grid = GridSearchCV(SVR(),param_grid,refit=True,verbose=0, cv=5)
```

In [123]:

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

Out[123]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=SVR(C=1.0, cache_size=200, coef
0=0.0, degree=3,
                           epsilon=0.1, gamma='scale',
kernel='rbf',
                           max_iter=-1, shrinking=Tru
e, tol=0.001,
                           verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [1, 10], 'gamma': [1, 0.
1], 'kernel': ['linear']},
             pre_dispatch='2*n_jobs', refit=True, retu
rn_train_score=False,
             scoring=None, verbose=0)
```

In [124]:

```
grid.best_params_
```

Out[124]:

```
{'C': 1, 'gamma': 1, 'kernel': 'linear'}
```

In [125]:

```
grid.best_estimator_
```

Out[125]:

```
SVR(C=1, cache_size=200, coef0=0.0, degree=3, epsilon=
0.1, gamma=1,
    kernel='linear', max_iter=-1, shrinking=True, tol=
0.001, verbose=False)
```

In [126]:

```
y_pred_test_optimized_SVR = grid.predict(X_test)
```

In [127]:

```
MSE_test = round(np.mean(np.square(y_test - y_pred_test_optimized_S
VR)),2)
MSE_test
```

Out[127]:

1.1

In [128]:

```
RMSE_test = round(np.sqrt(MSE_test),2)
RMSE_test
```

Out[128]:

1.05

KNN Regression

In [129]:

```
data2
```

Out[129]:

	Platform	Year_of_Release	Genre	Publisher	Global_Sa
0	Wii	2006.0	Sports	Nintendo	82
2	Wii	2008.0	Racing	Nintendo	35
3	Wii	2009.0	Sports	Nintendo	32
6	DS	2006.0	Platform	Nintendo	29
7	Wii	2006.0	Misc	Nintendo	28
...	...	...	...	...	
16667	GBA	2001.0	Action	Other	0
16677	GBA	2002.0	Fighting	Midway Games	0
16696	PC	2014.0	Action	Konami Digital Entertainment	0
16700	PC	2011.0	Shooter	Other	0
16706	PC	2011.0	Strategy	Unknown	0

6826 rows × 10 columns

Putting Developer variable made computation slow for our computer

In [130]:

```
#drop Developer variable
data2.drop(['Developer'],axis=1,inplace=True)
```

In [131]:

```
data2.describe(include='all')
```

Out[131]:

	Platform	Year_of_Release	Genre	Publisher	Global_Sales
count	6826	6826.000000	6826	6826	6826.000000
unique	17	NaN	12	30	NaN
top	PS2	NaN	Action	Other	NaN
freq	1140	NaN	1630	1128	NaN
mean	NaN	2007.437299	NaN	NaN	0.777499
std	NaN	4.211160	NaN	NaN	1.963313
min	NaN	1985.000000	NaN	NaN	0.010000
25%	NaN	2004.000000	NaN	NaN	0.110000
50%	NaN	2007.000000	NaN	NaN	0.290000
75%	NaN	2011.000000	NaN	NaN	0.750000
max	NaN	2016.000000	NaN	NaN	82.530000

In [132]:

```
##Log transormation for y-variable
data2['Global_Sales']=np.log(data2['Global_Sales'])
```

In [133]:

```
data2.describe(include='all')
```

Out[133]:

	Platform	Year_of_Release	Genre	Publisher	Global_Sales
count	6826	6826.000000	6826	6826	6826.000000
unique	17	NaN	12	30	NaN
top	PS2	NaN	Action	Other	NaN
freq	1140	NaN	1630	1128	NaN
mean	NaN	2007.437299	NaN	NaN	-1.233494
std	NaN	4.211160	NaN	NaN	1.399159
min	NaN	1985.000000	NaN	NaN	-4.605170
25%	NaN	2004.000000	NaN	NaN	-2.207275
50%	NaN	2007.000000	NaN	NaN	-1.237874
75%	NaN	2011.000000	NaN	NaN	-0.287682
max	NaN	2016.000000	NaN	NaN	4.413162

In [134]:

```
#create dummies
data2= pd.get_dummies(data2,drop_first=True)
```

In [135]:

```
rand_state=1000
```

Spliting the data

In [136]:

```
y = data2['Global_Sales']
X = data2.drop('Global_Sales', axis=1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
=0.2, random_state=rand_state)
```

Scaling the data

In [137]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [138]:

```
#starting with KNNregression
from sklearn.neighbors import KNeighborsRegressor
```

In [139]:

```
# Fitting KNN regression to the Training set
KNN_regression = KNeighborsRegressor(n_neighbors=54)
KNN_regression.fit(X_train, y_train)
```

Out[139]:

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, me
tric='minkowski',
                    metric_params=None, n_jobs=None, n
_neighbors=54, p=2,
                    weights='uniform')
```

In [140]:

```
# Predicting the Test set results
y_pred_test = KNN_regression.predict(X_test)
```

In [141]:

```
predictions = pd.DataFrame({ 'y_actual_test':y_test,'y_pred_test':y
_pred_test, 'resid':y_test - y_pred_test})
predictions.head()
```

Out[141]:

	y_actual_test	y_pred_test	resid
14759	-3.506558	-0.971635	-2.534923
9627	-2.040221	-1.453372	-0.586849
1632	0.207014	-0.541862	0.748877
5056	-0.967584	-1.164247	0.196663
14529	-3.506558	-2.195253	-1.311305

In [142]:

```
##evaluating on test dataset
MSE_test = round(np.mean(np.square(y_test - y_pred_test)),2)
MSE_test
```

Out[142]:

1.38

In [143]:

```
RMSE_test = round(np.sqrt(MSE_test),2)
RMSE_test
```

Out[143]:

1.17

In [144]:

```
##Cross Validation 10 fold
from sklearn.model_selection import cross_val_score
```

In [145]:

```
NMSE = cross_val_score(estimator = KNN_regression, X = X_train, y = y_train, cv = 10 , scoring="neg_mean_squared_error" )
```

In [146]:

```
CV_MSE = round(np.mean(-NMSE),2)
CV_MSE
```

Out[146]:

```
1.33
```

In [147]:

```
CV_RMSE = round(np.sqrt(CV_MSE))
CV_RMSE
```

Out[147]:

```
1.0
```

In [148]:

```
CV_RMSE=[]
test_RMSE = []

k=100

for i in range(1,k):
    KNN_i = KNeighborsRegressor(n_neighbors=i)
    KNN_i.fit(X_train, y_train)
    RMSE_i = np.sqrt(np.mean(-1*cross_val_score(estimator = KNN_i,
X = X_train, y = y_train, cv = 10 , scoring="neg_mean_squared_erro
r" )))
    CV_RMSE.append(RMSE_i)

    test_RMSE.append(np.sqrt(np.mean(np.square(y_test - KNN_i.predi
ct(X_test)))))

optimal_k = pd.DataFrame({'CV_RMSE': np.round(CV_RMSE,2), 'test_RMS
E':np.round(test_RMSE,2)}, index=range(1,k))
```

In [149]:

```
optimal_k
```

Out[149]:

	CV_RMSE	test_RMSE
1	1.35	1.35
2	1.21	1.18
3	1.16	1.16
4	1.14	1.15
5	1.13	1.14
...	...	...
95	1.18	1.19
96	1.18	1.19
97	1.18	1.19
98	1.18	1.19
99	1.18	1.19

99 rows × 2 columns

Lets find the minimum for each column for now

In [151]:

```
print(optimal_k[optimal_k.CV_RMSE == optimal_k.CV_RMSE.min()])
```

```
   CV_RMSE  test_RMSE
8     1.11       1.12
```

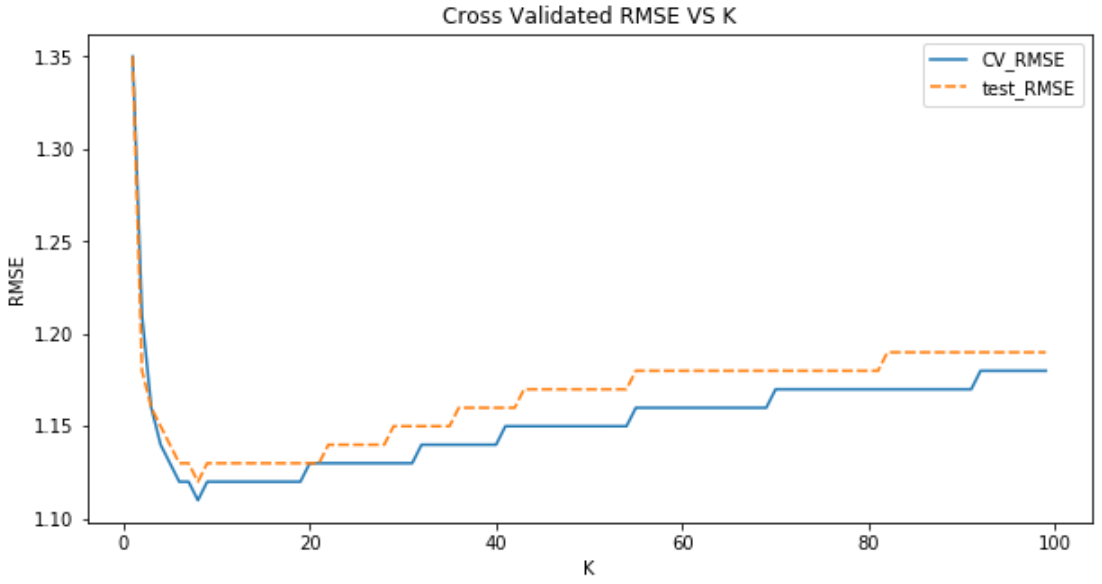
In [152]:

```
print(optimal_k[optimal_k.test_RMSE == optimal_k.test_RMSE.min()])
```

```
   CV_RMSE  test_RMSE
8     1.11       1.12
```

In [153]:

```
plt.figure(figsize=(10,5))
sns.lineplot(data=optimal_k)
plt.title('Cross Validated RMSE VS K')
plt.xlabel('K')
plt.ylabel('RMSE')
plt.show()
```



We have consensus from both CV_rmse and test_rmse , since we have test set observable we can use test set rmse to get the better result, further we could have done CV for test set.

Running the KNN again with the optimized number of K

In [154]:

```
# Fitting KNN regression to the Training set
KNN_regression = KNeighborsRegressor(n_neighbors=8)
KNN_regression.fit(X_train, y_train)
```

Out[154]:

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, me
tric='minkowski',
                    metric_params=None, n_jobs=None, n
_neighbors=8, p=2,
                    weights='uniform')
```

In [155]:

```
# Predicting the Test set results
y_pred_test = KNN_regression.predict(X_test)
```

In [156]:

```
predictions = pd.DataFrame({ 'y_actual_test':y_test,'y_pred_test':y
_pred_test, 'resid':y_test - y_pred_test})
predictions.head()
```

Out[156]:

	y_actual_test	y_pred_test	resid
14759	-3.506558	-1.903552	-1.603006
9627	-2.040221	-1.374752	-0.665469
1632	0.207014	-0.361318	0.568333
5056	-0.967584	0.037517	-1.005101
14529	-3.506558	-2.338352	-1.168206

In [157]:

```
MSE_test = round(np.mean(np.square(y_test - y_pred_test)),2)
MSE_test
```

Out[157]:

1.26

In [159]:

```
RMSE_test = round(np.sqrt(MSE_test),2)
RMSE_test
```

Out[159]:

1.12

Lets summarize the models

```
In Summary
Linear regression ---- 0.91
Randomm Forest regression --- 0.89
SVR Rgerssion ------ 1.05
KNN Regression ----- 1.12
```

Looks like random forest does the better job for predicting the number of Global_Sales per million.