App Rating Prediction

To make a model to predict the app rating, with other information about the app provided

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
data = pd.read_csv("googleplaystore.csv")
```

In [3]:
```
data.shape
```

Out[3]:
```
(10841, 13)
```

In [4]:
```
data.columns
```

Out[4]:
```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type',
       'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver',
       'Android Ver'],
      dtype='object')
```

In [5]:
```
data.head()
```

Out[5]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art & Design	January 7, 2018	1.0.0	4.0.3 and up
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	0	Everyone	Art & Design;Pretend Play	January 15, 2018	2.0.0	4.0.3 and up
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+	Free	0	Everyone	Art & Design	August 1, 2018	1.2.4	4.0.3 and up
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000,000+	Free	0	Teen	Art & Design	June 8, 2018	Varies with device	4.2 and up
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100,000+	Free	0	Everyone	Art & Design;Creativity	June 20, 2018	1.1	4.4 and up

Check for null values in the data. Get the number of null values for each column.

Drop records with nulls in any of the columns.

Variables seem to have incorrect type and inconsistent formatting. You need to fix them:

Size column has sizes in Kb as well as Mb. To analyze, you'll need to convert these to numeric.

Extract the numeric value from the column

Multiply the value by 1,000, if size is mentioned in Mb

Reviews is a numeric field that is loaded as a string field. Convert it to numeric (int/float).

Installs field is currently stored as string and has values like 1,000,000+.

Treat 1,000,000+ as 1,000,000

remove '+', ',' from the field, convert it to integer

Price field is a string and has symbol. Remove '' sign

In [6]:
```
data.isnull().sum()  #checking for any null values
```

Out[6]:
```
App                  0
Category             0
Rating            1474
Reviews              0
Size                 0
Installs             0
Type                 1
Price                0
Content Rating       1
Genres               0
Last Updated         0
Current Ver          8
Android Ver          3
dtype: int64
```

In [7]:
```
data1 = data.dropna()
```

In [8]:
```
data1.shape
```

Out[8]:
```
(9360, 13)
```

In [9]:
```
data1.isnull().sum()  #so all the null values are dropped
```

Out[9]:
```
App               0
Category          0
Rating            0
Reviews           0
Size              0
Installs          0
Type              0
Price             0
Content Rating    0
Genres            0
Last Updated      0
Current Ver       0
Android Ver       0
dtype: int64
```

In [10]:
```
data1["Size"]
```

Out[10]:
```
0                       19M
1                       14M
2                      8.7M
3                       25M
4                      2.8M
                ...        
10834                  2.6M
10836                   53M
10837                  3.6M
10839    Varies with device
10840                   19M
Name: Size, Length: 9360, dtype: object
```

In [11]:
```
data1 = data1[-data1["Size"].str.contains("Var")]         #removing the part of "Size" column where it contains
# here negative sign i.e., before data1 is applied for removing the "Var" rows from dataset
#or
#data1 = data1[data1['Size'].str.contains('Varies with devices') == False]   #this code will do the same operat
```

In [12]:
```
data1["Size"]
```

Out[12]:
```
0         19M
1         14M
2        8.7M
3         25M
4        2.8M
         ... 
10833    619k
10834    2.6M
10836     53M
10837    3.6M
10840     19M
Name: Size, Length: 7723, dtype: object
```

In [13]:
```
data1.loc[:,"SizeNum"] = data1.Size.str.rstrip("Mk+")

#created a new column "SizeNum" with "Size" column values which doesnot has "M","k",or"+""
```

In [14]:
```
data1["SizeNum"]
```

Out[14]:
```
0         19
1         14
2        8.7
3         25
4        2.8
        ... 
10833    619
10834    2.6
10836     53
10837    3.6
10840     19
Name: SizeNum, Length: 7723, dtype: object
```

In [15]:
```
data1.SizeNum = pd.to_numeric(data1["SizeNum"])
```

In [16]:
```
data1["SizeNum"].dtype
```

Out[16]:
```
dtype('float64')
```

In [17]:
```
import numpy as np
```

Multiply the value by 1,000, if size is mentioned in Mb

In [18]:
```
data1["SizeNum"] =np.where(data1["Size"].str.contains("M"),data1["SizeNum"]*1000,data1.SizeNum)
```

In [19]:
```
data1["SizeNum"]
```

Out[19]:
```
0        19000.0
1        14000.0
2         8700.0
3        25000.0
4         2800.0
          ...   
10833      619.0
10834     2600.0
10836    53000.0
10837     3600.0
10840    19000.0
Name: SizeNum, Length: 7723, dtype: float64
```

In [20]:
```
data1.Size = data1.SizeNum
data1.drop("SizeNum",axis=1,inplace=True)

#replacing the values of "Size" column with "SizeNum" column and dropping the "SizeNum" column
```

Reviews is a numeric field that is loaded as a string field. Convert it to numeric (int/float)

In [21]:
```
data1.Reviews = pd.to_numeric(data1.Reviews)
```

In [22]:
```
data1["Reviews"].dtype
```

Out[22]:
```
dtype('int64')
```

"Installs" field is currently stored as string and has values like 1,000,000+.

Treat 1,000,000+ as 1,000,000

remove '+', ',' from the field, convert it to integer

Price field is a string and has $ symbol.

Remove '$' sign, and convert it to numeric.

In [23]:
```
data1["Installs"]=data1["Installs"].str.replace("+","")  #replacing "+" by ""  (blank)
```

```
<ipython-input-23-ca8095dd9749>:1: FutureWarning: The default value of regex will change from True to False in a future version. In addition, single character regular expressions will*not* be treated as literal strings whe
n regex=True.
  data1["Installs"]=data1["Installs"].str.replace("+","")  #replacing "+" by ""  (blank)
```

In [24]:
```
data1["Installs"]=data1["Installs"].str.replace(",","")   #replacing "," by ""  (blank)
```

In [25]:
```
data1["Installs"] = pd.to_numeric(data1.Installs)  #converting installs dtype to numberic
```

In [26]:
```
data1["Installs"].dtype
```

Out[26]:
```
dtype('int64')
```

This code also do the same treatment, here we used the lambda function

data1['Installs'] = data1['Installs'].map(lambda x: str(x).replace('+','')) #x iterator replace the + with nothing

data1['Installs'] = data1['Installs'].map(lambda x: str(x).replace(',','')) #x iterator replace the , with nothing

data1['Installs'] = data1['Installs'].astype(int)

In [27]:
```
data1["Installs"]
```

Out[27]:
```
0           10000
1          500000
2         5000000
3        50000000
4          100000
           ...   
10833        1000
10834         500
10836        5000
10837         100
10840    10000000
Name: Installs, Length: 7723, dtype: int64
```

In [28]:
```
data1["Price"]=data1["Price"].str.replace("$","")   #replacing "$" by ""
```

```
<ipython-input-28-30317f0f4994>:1: FutureWarning: The default value of regex will change from True to False in a future version. In addition, single character regular expressions will*not* be treated as literal strings whe
n regex=True.
  data1["Price"]=data1["Price"].str.replace("$","")   #replacing "$" by ""
```

In [29]:
```
data1["Price"] = pd.to_numeric(data1.Price)
```

In [30]:
```
data1.Price.dtype
```

Out[30]:
```
dtype('float64')
```

Sanity checks:

Average rating should be between 1 and 5 as only these values are allowed on the play store. Drop the rows that have a value outside this range.

Reviews should not be more than installs as only those who installed can review the app. If there are any such records, drop them.

For free apps (type = "Free"), the price should not be >0. Drop any such rows

In [31]:
```
data1 = data1[(data1.Rating>=1) & (data1.Rating<=5)]
```

In [32]:
```
data1["Rating"]
```

Out[32]:
```
0        4.1
1        3.9
2        4.7
3        4.5
4        4.3
        ... 
10833    4.8
10834    4.0
10836    4.5
10837    5.0
10840    4.5
Name: Rating, Length: 7723, dtype: float64
```

In [33]:
```
len(data1.index)
```

Out[33]:
```
7723
```

In [34]:
```
data1.drop(data1.index[data1.Reviews > data1.Installs],axis=0,inplace=True)

#dropping the records where "Reviews" are more than "Installs"
```

In [35]:
```
len(data1.index)
```

Out[35]:
```
7717
```

In [36]:
```
import warnings
warnings.filterwarnings('ignore')
```

For free apps (type = "Free"), the price should not be >0. Drop any such rows

In [37]:
```
data1[(data1["Type"]=="Free") & (data1["Price"]>0)]

#checking if any "Free" apps with price greater than 0
```

Out[37]:

App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver

There are no such records

Performing univariate analysis:

-BOXPLOT for "Price"

Are there any outliers? Think about the price of usual apps on Play Store.

-BOXPLOT for "Reviews"

Are there any apps with very high number of reviews? Do the values seem right?

-HISTOGRAM for "Rating"

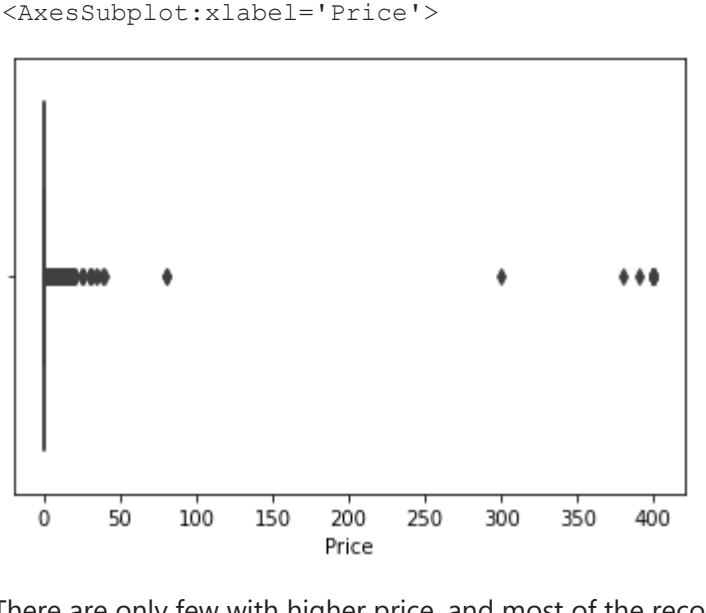
How are the ratings distributed? Is it more toward higher ratings?

-HISTOGRAM for "Size"

Note down your observations for the plots made above. Which of these seem to have outliers?

In [38]:
```
sns.boxplot(x="Price",data=data1)
```

Out[38]:
```
<AxesSubplot:xlabel='Price'>
```



There are only few with higher price, and most of the records falling under 100. So we can consider that, records with greater than 100 are the outliers.

In [39]:
```
std = np.std(data1.Price)  #standard deviation
```

In [40]:
```
mean =np.mean(data1.Price)   #mean
```

In [41]:
```
outlier_uplimit = mean + 3 *std
```

In [42]:
```
outlier_uplimit
```

Out[42]:
```
53.36969138940857
```

So anything beyond 53.36 can be called upper outlier

In [43]:
```
len(data1[(data1["Price"]>outlier_uplimit)])
```

Out[43]:
```
17
```

So, there are 17 outlier records

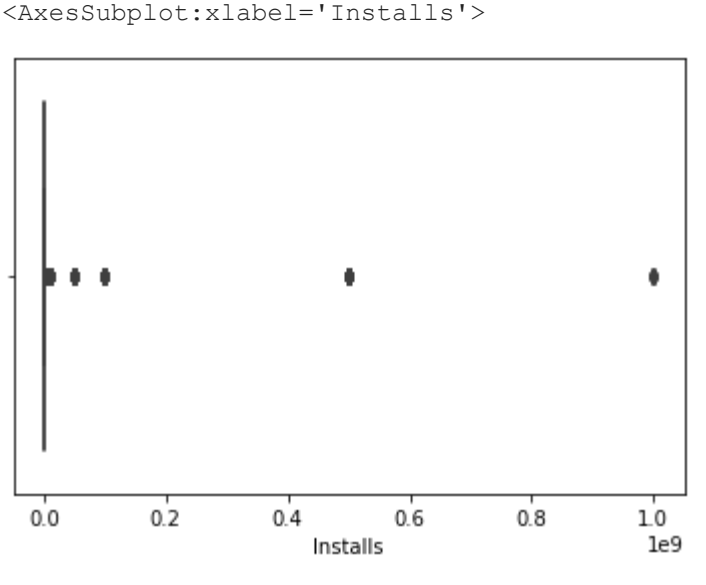
BOXPLOT for REVIEWS- Installs- There seems to be some outliers in this field too. Apps having very high number of installs should be dropped from the analysis.

Find out the different percentiles – 10, 25, 50, 70, 90, 95, 99

Decide a threshold as cutoff for outlier and drop records having values more than that

In [44]:
```
sns.boxplot(x="Installs",data=data1)
```

Out[44]:
```
<AxesSubplot:xlabel='Installs'>
```



In [45]:
```
np.percentile(data1["Installs"],10)
```

Out[45]:
```
1000.0
```

In [46]:
```
np.percentile(data1["Installs"],25)
```

Out[46]:
```
10000.0
```

In [47]:
```
np.percentile(data1["Installs"],50)
```

Out[47]:
```
100000.0
```

In [48]:
```
np.percentile(data1["Installs"],70)
```

Out[48]:
```
1000000.0
```

In [49]:
```
np.percentile(data1["Installs"],90)
```

Out[49]:
```
10000000.0
```

In [50]:
```
np.percentile(data1["Installs"],95)
```

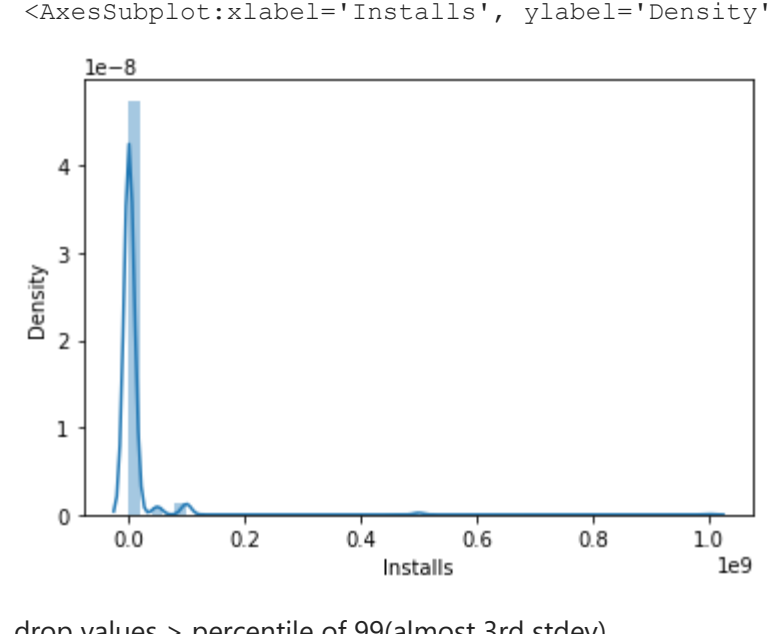
Out[50]:
```
50000000.0
```

In [51]:
```
np.percentile(data1["Installs"],99)
```

Out[51]:
```
100000000.0
```

In [52]:
```
sns.distplot(data1["Installs"])
```

Out[52]:
```
<AxesSubplot:xlabel='Installs', ylabel='Density'>
```



drop values > percentile of 99(almost 3rd stdev)

In [53]:
```
len(data1[data1.Installs>=100000000.0])
#checking again that how many values are greater than 3rd standard deviation
```

Out[53]:
```
241
```

In [54]:
```
data1.drop(data1.index[data1.Installs>=100000000.0],inplace=True)
#dropping those values
```

Bivariate analysis- Let's look at how the available predictors relate to the variable of interest, i.e., our target variable rating.

Make scatter plots (for numeric features) and box plots (for character features) to assess the relations between rating and the other features.

Make scatter plot/joinplot for Rating vs. Price

What pattern do you observe? Does rating increase with price?

Make scatter plot/joinplot for Rating vs. Size

Are heavier apps rated better?

Make scatter plot/joinplot for Rating vs. Reviews

Does more review mean a better rating always?

Make boxplot for Rating vs. Content Rating

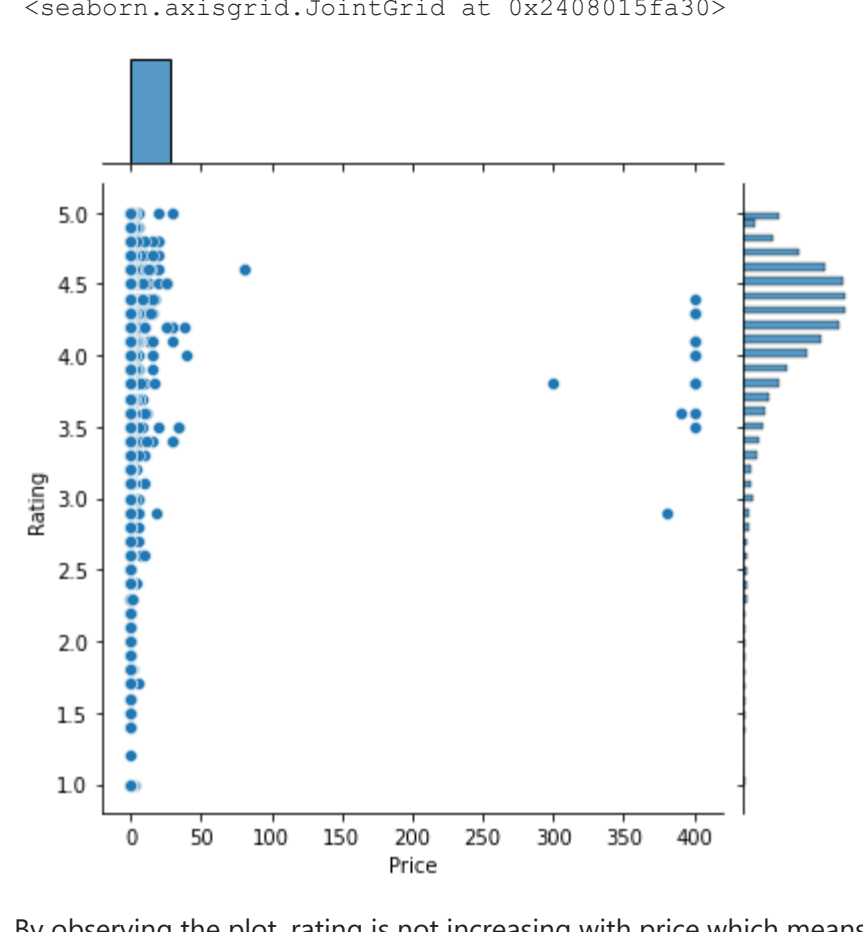
Is there any difference in the ratings? Are some types liked better?

Make boxplot for Ratings vs. Category

Which genre has the best ratings?

In [55]:
```
sns.jointplot(x="Price",y="Rating",data=data1)
```

Out[55]:
```
<seaborn.axisgrid.JointGrid at 0x2408015fa30>
```

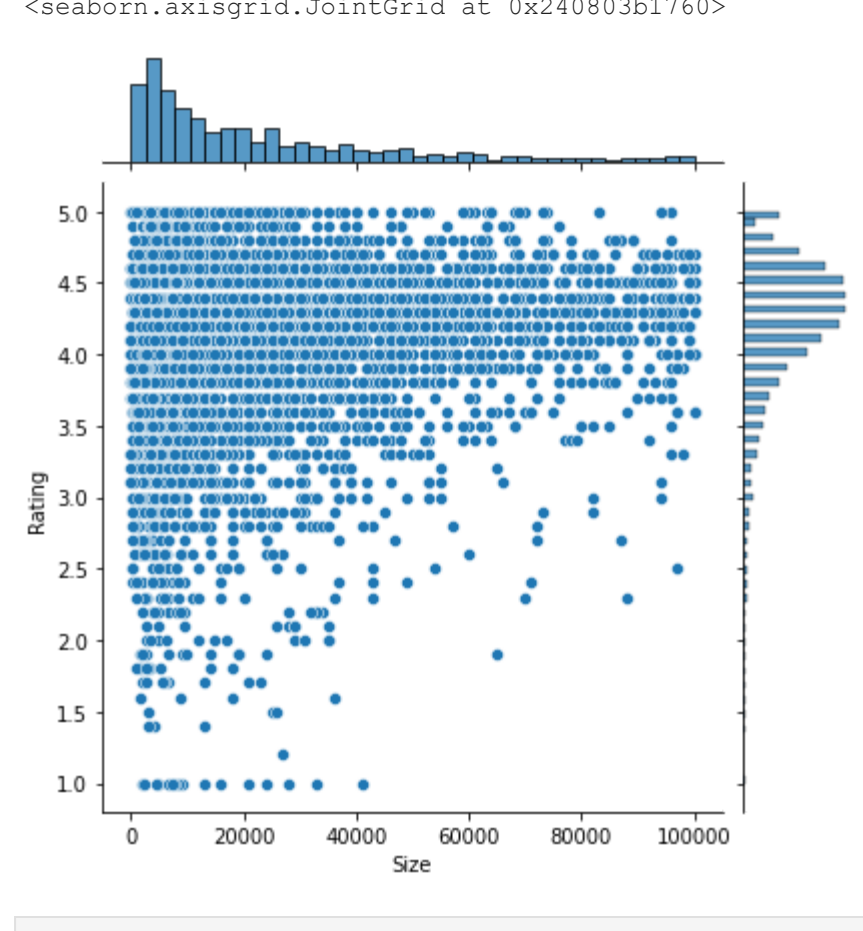


By observing the plot, rating is not increasing with price which means price has limited impact on rating.

Make scatter plot/joinplot for Rating vs. Size

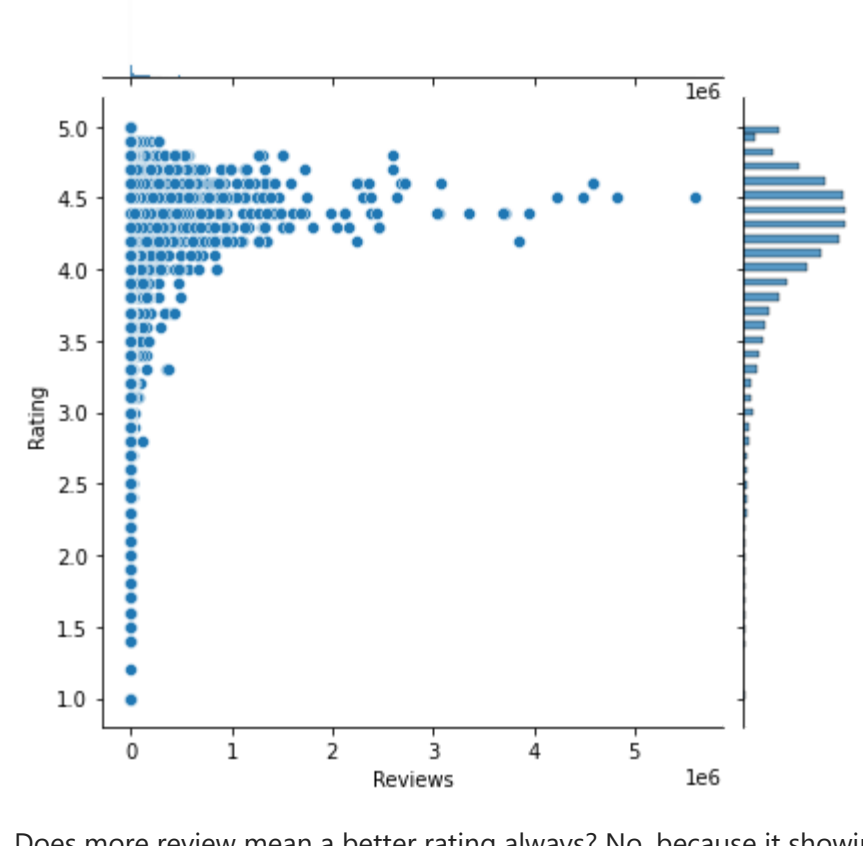
In [56]:
```
sns.jointplot(x="Size",y="Rating",data=data1)
#joint plot for SIZE and RATING
```

Out[56]:
```
<seaborn.axisgrid.JointGrid at 0x240803b1760>
```



In [57]:
```
sns.jointplot(x="Reviews",y="Rating",data=data1)
#jointplot for REVIEWS and RATING
```

Out[57]:
```
<seaborn.axisgrid.JointGrid at 0x240805b6eb0>
```



Does more review mean a better rating always? No, because it showing a weak relation in the plot

In [58]:
```
data1.corr()
```

Out[58]:

	Rating	Reviews	Size	Installs	Price
Rating	1.000000	0.116220	0.067926	0.090206	-0.020520
Reviews	0.116220	1.000000	0.217629	0.725131	-0.017533
Size	0.067926	0.217629	1.000000	0.199643	-0.024904
Installs	0.090206	0.725131	0.199643	1.000000	-0.023467
Price	-0.020520	-0.017533	-0.024904	-0.023467	1.000000

Make boxplot for Rating vs. Content Rating

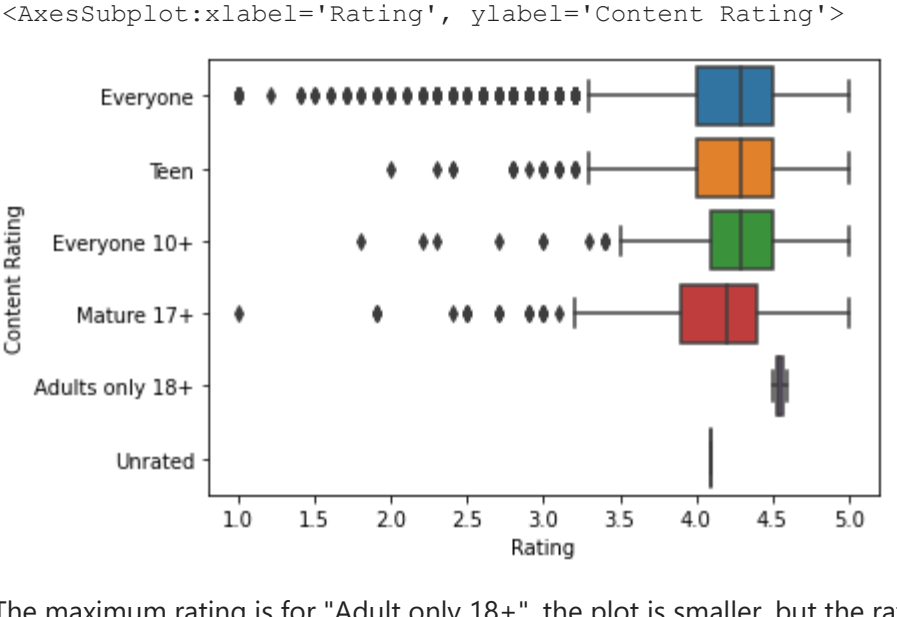
Is there any difference in the ratings? Are some types liked better?

In [59]:
```
data1["Content Rating"].unique()
```

Out[59]:
```
array(['Everyone', 'Teen', 'Everyone 10+', 'Mature 17+',
       'Adults only 18+', 'Unrated'], dtype=object)
```

In [60]:
```
sns.boxplot(x="Rating",y="Content Rating",data=data1)
```

Out[60]:
```
<AxesSubplot:xlabel='Rating', ylabel='Content Rating'>
```

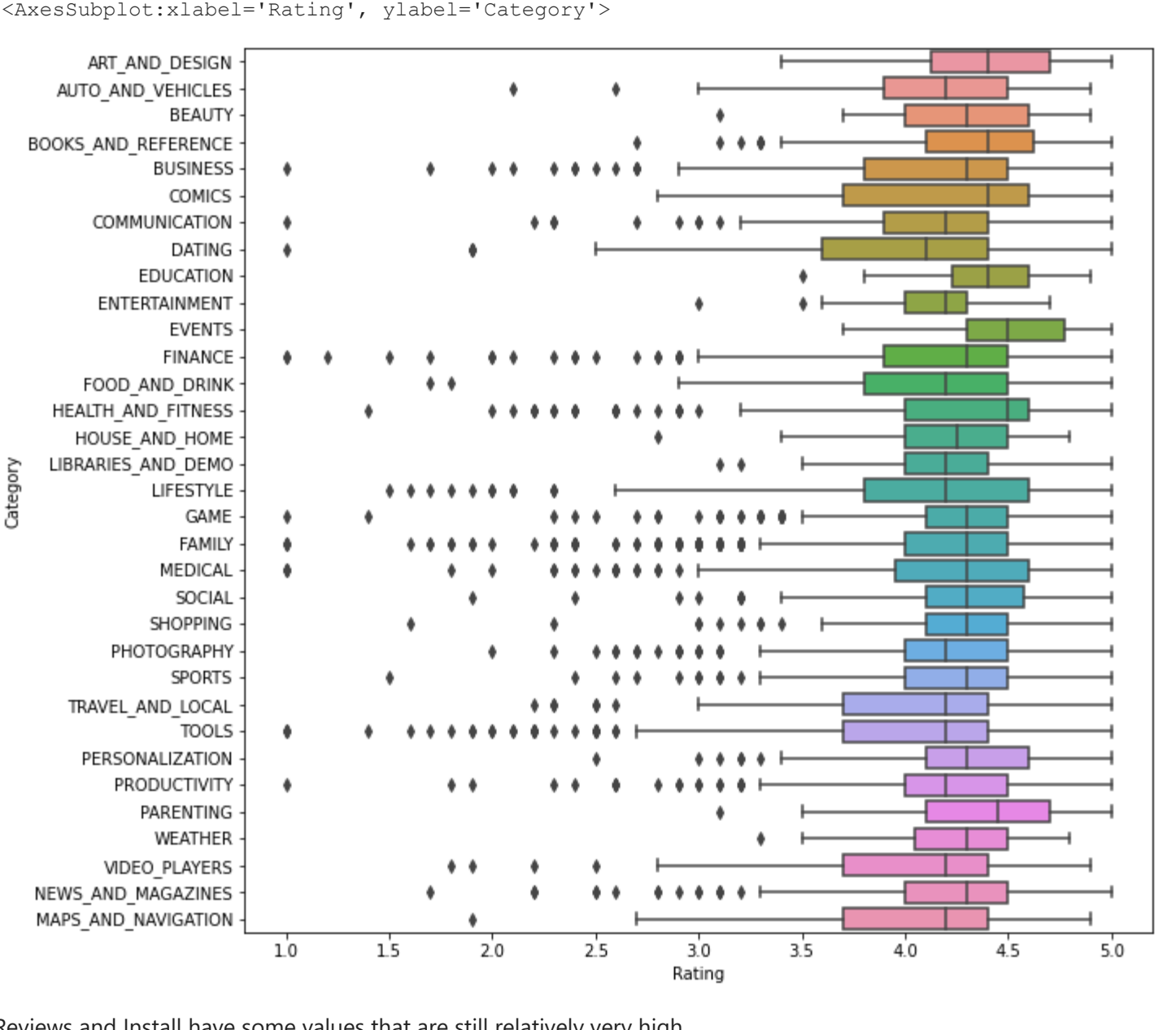


The maximum rating is for "Adult only 18+", the plot is smaller, but the rating is on higher side (around 4.5)

All the other content ratings seems to be similar.

In [61]:
```
plt.figure(figsize=(10,10))
sns.boxplot(x="Rating",y="Category",data=data1)
```

Out[61]:
```
<AxesSubplot:xlabel='Rating', ylabel='Category'>
```



Reviews and Install have some values that are still relatively very high.

Before building a linear regression model, you need to reduce the skew.

Apply log transformation (np.log1p) to Reviews and Installs.

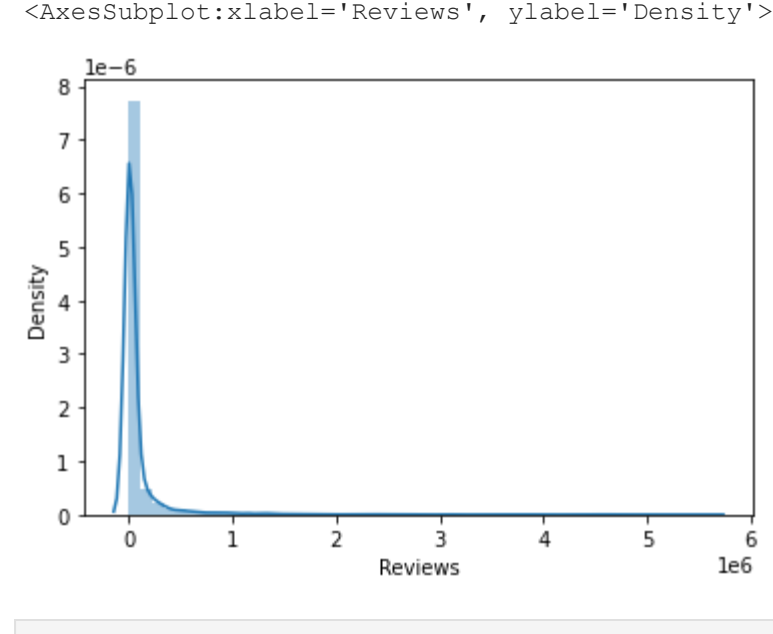
Drop columns App, Last Updated, Current Ver, and Android Ver. These variables are not useful for our task.

Get dummy columns for Category, Genres, and Content Rating. This needs to be done as the models do not understand categorical data, and all data should be numeric. Dummy encoding is one way to convert character fields to numeric. Name of dataframe should be inp2.

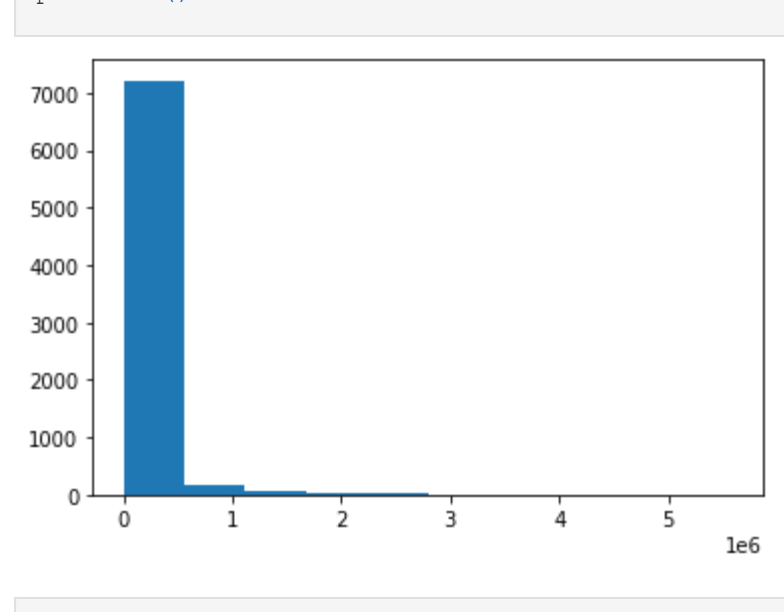
In [62]:
```
inp1 = data1.copy()
#creating a copy of the dataframe to make all the edits and naming it "inp1".
```

In [63]:
```
sns.distplot(inp1["Reviews"])
```

Out[63]:
```
<AxesSubplot:xlabel='Reviews', ylabel='Density'>
```



In [64]:
```
plt.hist(inp1[["Reviews"]])
plt.show()
```



In [65]:
```
inp1.Reviews=inp1.Reviews.apply(np.log1p)

#to remove the skewness in the data we are applying np.log1p (exponential smoothing)
```

In [66]:
```
inp1.Installs=inp1.Installs.apply(np.log1p)
```

In [67]:
```
inp1.drop(['App', 'Last Updated', 'Current Ver','Android Ver'],axis=1,inplace=True)

#dropping non useful columns i.e., App, Last Updated, Current Ver, and Android Ver
```

In [68]:
```
inp1.shape
```

Out[68]:
```
(7476, 9)
```

In [69]:
```
inp1.columns
```

Out[69]:
```
Index(['Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type', 'Price',
       'Content Rating', 'Genres'],
      dtype='object')
```

In [70]:
```
inp1["Type"].unique()
```

Out[70]:
```
array(['Free', 'Paid'], dtype=object)
```

In [71]:
```
inp2 = pd.get_dummies(inp1)
```

In [72]:
```
inp2.shape
```

Out[72]:
```
(7476, 158)
```

Train test split and apply 70-30 split. Name the new dataframes df_train and df_test.

- Separate the dataframes into X_train, y_train, X_test, and y_test.
- Model building

Use linear regression as the technique

Report the R2 on the train set

- Make predictions on test set and report R2.

In [73]:
```
inp2.head()
```

Out[73]:

	Rating	Reviews	Size	Installs	Price	Category_ART_AND_DESIGN	Category_AUTO_AND_VEHICLES	Category_BEAUTY	Category_BOOK
0	4.1	5.075174	19000.0	9.210440	0.0	1	0	0	
1	3.9	6.875232	14000.0	13.122365	0.0	1	0	0	
2	4.7	11.379520	8700.0	15.424949	0.0	1	0	0	
3	4.5	12.281389	25000.0	17.727534	0.0	1	0	0	
4	4.3	6.875232	2800.0	11.512935	0.0	1	0	0	

5 rows × 158 columns

In [74]:
```
y = inp2.iloc[:,0]       #target
```

In [75]:
```
X = inp2.iloc[:,1:]      #features
```

In [76]:
```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.3)
```

In [77]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

In [78]:
```
lr.fit(x_train,y_train)
```

Out[78]: LinearRegression()

In [79]:
```
y_pred = lr.predict(x_test)
```

In [80]:
```
from sklearn.metrics import r2_score
```

In [81]:
```
r2_score(y_test,y_pred)
```

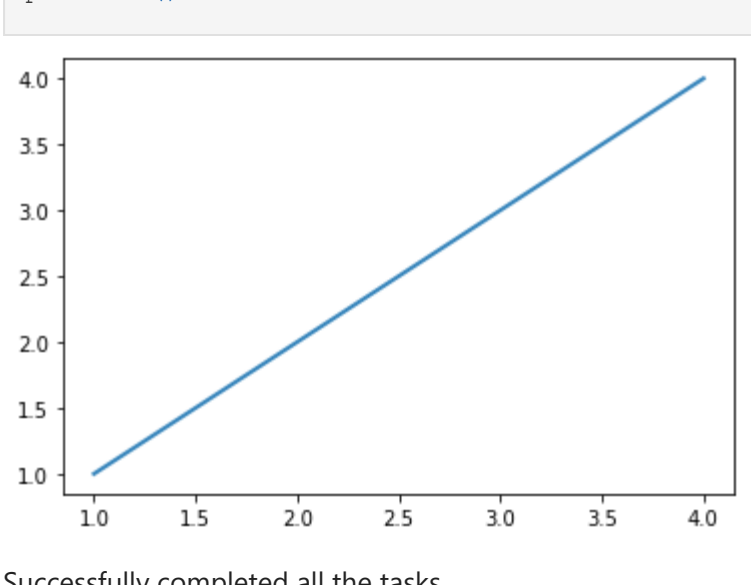
Out[81]: 0.11160331489180697

Here the accuracy is very less and reason is we applied the "get_dummies" (as asked in the task),

and it highly increases the dimension

(for better accuracy, we can use "LabelEncoder" (Scikitlearn approach))

In [82]:
```
import matplotlib.pyplot as plt
x = [1,2,3,4]
y =x
plt.plot(x,y,linewidth=2,alpha=0.9)
plt.show()
```



Successfully completed all the tasks

by Ashish Roy