In [ ]:

```
import pandas as pd
import seaborn as sns
```

In [ ]:

```
df = pd.read_csv("googleplaystore.csv")
```

In [ ]:

```
df.shape
```

Out[ ]:

```
(10841, 13)
```

In [ ]:

```
df.columns
```

Out[ ]:

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

In [ ]:

```
df.head(2)
```

Out[ ]:

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

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 $symbol$ ' sign

$.Remove$'

In [ ]:

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

Out[ ]:

```
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 [ ]:

```
df1 = df.dropna()
```

In [ ]:

```
df1.shape
```

Out[ ]:

```
(9360, 13)
```

In [ ]:

```
df1.isna().sum()
```

Out[ ]:

```
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 [ ]:

```
df1["Size"]
```

Out[ ]:

```
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 [ ]:

```
df1 = df1[-df1["Size"].str.contains("Var")]#To get everything except rows where size=stri
ng starts with Var.
```

In [ ]:

```
df1["Size"]
```

Out[ ]:

```
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 [ ]:

```
df1.loc[:,"SizeNum"] = df1.Size.str.rstrip("Mk+")
```

In [ ]:

```
df1["SizeNum"]
```

Out[ ]:

```
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 [ ]:

```
df1.SizeNum = pd.to_numeric(df1["SizeNum"])
```

In [ ]:

```
df1["SizeNum"].dtype
```

Out[ ]:

```
dtype('float64')
```

In [ ]:

```
import numpy as np
df1["SizeNum"] =np.where(df1["Size"].str.contains("M"),df1["SizeNum"]*1000,df1.SizeNum)
```

In [ ]:

```
df1["SizeNum"]
```

Out[ ]:

```
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 [ ]:

```
df1.Size = df1.SizeNum
df1.drop("SizeNum",axis=1,inplace=True)
```

In [ ]:

```
#converr reviews into numeric
```

In [ ]:

```
df1.Reviews = pd.to_numeric(df1.Reviews)
```

In [ ]:

```
df1["Reviews"].dtype
```

Out[ ]:

```
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 $symbol$ ' sign

$.Remove$
'

In [ ]:

```
df1["Installs"]=df1["Installs"].str.replace("+","")
```

```
<ipython-input-23-a6f72684b18a>: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 when regex=True.
  df1["Installs"]=df1["Installs"].str.replace("+","")
```

In [ ]:

```
df1["Installs"]=df1["Installs"].str.replace(",","")
```

In [ ]:

```
df1["Installs"] = pd.to_numeric(df1.Installs)
```

In [ ]:

```
df1["Installs"].dtype
```

Out[ ]:

```
dtype('int64')
```

In [ ]:

```
df1["Installs"]
```

Out[ ]:

```
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 [ ]:

```
df1["Price"]=df1["Price"].str.replace("$","")
```

```
<ipython-input-28-5b968cd8a683>: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 when regex=True.
  df1["Price"]=df1["Price"].str.replace("$","")
```

In [ ]:

```
df1["Price"] = pd.to_numeric(df1.Price)
```

In [ ]:

```
df1.Price.dtype
```

Out[ ]:

```
dtype('float64')
```

1. **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 [ ]:

```
df1 = df1[(df1.Rating>=1) & (df1.Rating<=5)]
```

In [ ]:

```
df1["Rating"]
```

Out[ ]:

```
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 [ ]:

```
len(df1.index)
```

Out[ ]:

```
7723
```

In [ ]:

```
df1.drop(df1.index[df1.Reviews>df1.Installs],axis=0,inplace=True)
```

In [ ]:

```
len(df1.index)
```

Out[ ]:

```
7717
```

In [ ]:

```
import warnings
warnings.filterwarnings('ignore')
```

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

In [ ]:

```
df1[(df1["Type"]=="Free") & (df1["Price"]>0)]
```

Out[ ]:

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

In [ ]:

```
#There are no free apps with price > 0.
```

1. **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

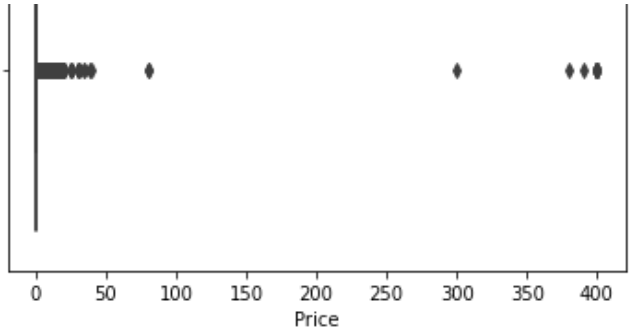
Note down your observations for the plots made above. Which of these seem to have outliers?

In [ ]:

```
sns.boxplot(x="Price",data=df1)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5091e40130>
```



In [ ]:

```
#greater than 100 might be consider as outliers
```

In [ ]:

```
std = np.std(df1.Price)
```

In [ ]:

```
mean =np.mean(df1.Price)
```

In [ ]:

```
outlier_uplimit = mean + 3 *std
```

In [ ]:

```
outlier_uplimit
```

Out[ ]:

```
53.36969138940375
```

In [ ]:

```
len(df1[(df1["Price"]>outlier_uplimit)])
```

Out[ ]:

```
17
```

Boxplot for Reviews --- same way as with price. 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

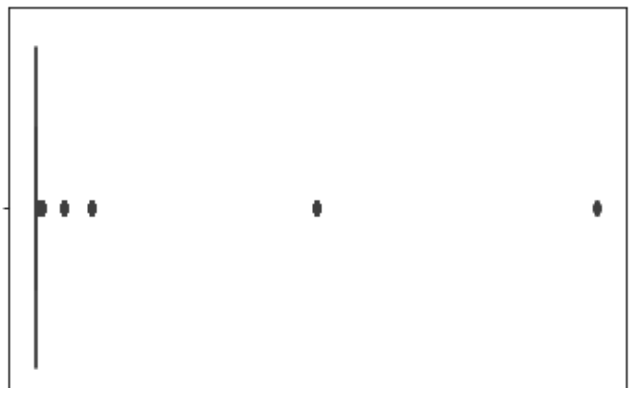
Decide a threshold as cutoff for outlier and drop records having values more than that

In [ ]:

```
sns.boxplot(x="Installs",data=df1)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5091d8ee80>
```



In [ ]:

```
np.percentile(df1["Installs"],10)
```

Out[ ]:

```
1000.0
```

In [ ]:

```
np.percentile(df1["Installs"],25)
```

Out[ ]:

```
10000.0
```

In [ ]:

```
np.percentile(df1["Installs"],50)
```

Out[ ]:

```
100000.0
```

In [ ]:

```
np.percentile(df1["Installs"],70)
```

Out[ ]:

```
1000000.0
```

In [ ]:

```
np.percentile(df1["Installs"],90)
```

Out[ ]:

```
10000000.0
```

In [ ]:

```
np.percentile(df1["Installs"],95)
```

Out[ ]:

```
50000000.0
```

In [ ]:

```
np.percentile(df1["Installs"],99)
```

Out[ ]:

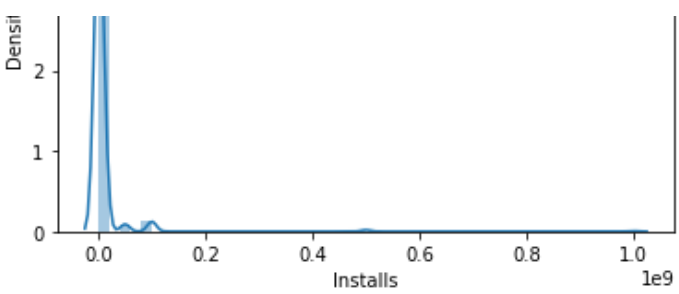
```
100000000.0
```

In [ ]:

```
sns.distplot(df1["Installs"])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5091893fd0>
```



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

In [ ]:

```
len(df1[df1.Installs>=100000000.0])
```

Out[ ]:

```
241
```

In [ ]:

```
df1.drop(df1.index[df1.Installs>=100000000.0],inplace=True)
```

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

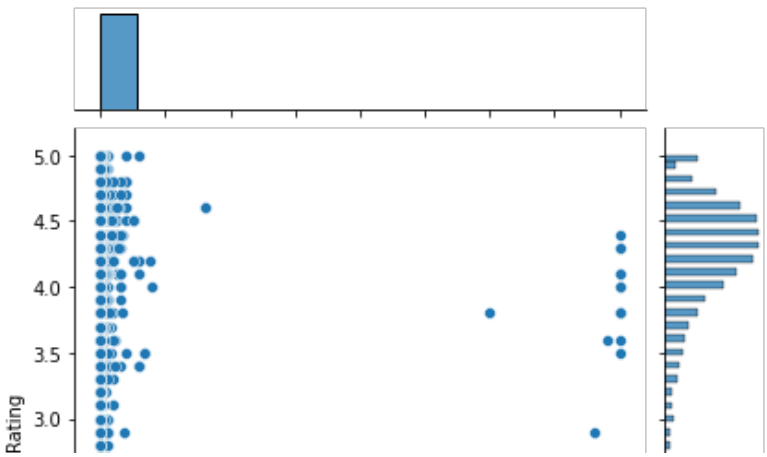
Which genre has the best ratings?

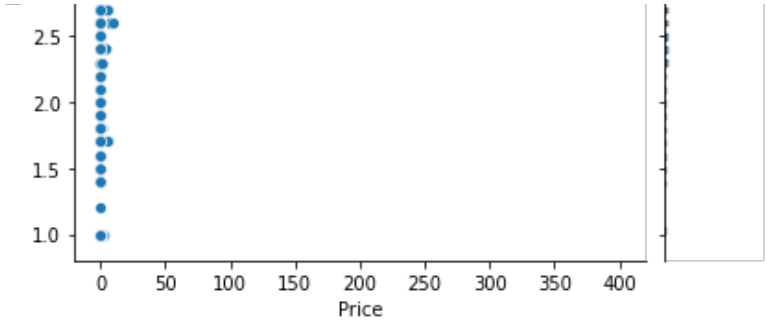
In [ ]:

```
sns.jointplot(x="Price",y="Rating",data=df1)
```

Out[ ]:

```
<seaborn.axisgrid.JointGrid at 0x7f508f7a0820>
```





Does rating increase with price? It seems like price has limited impact on rating.

Make scatter plot/joinplot for Rating vs. Size

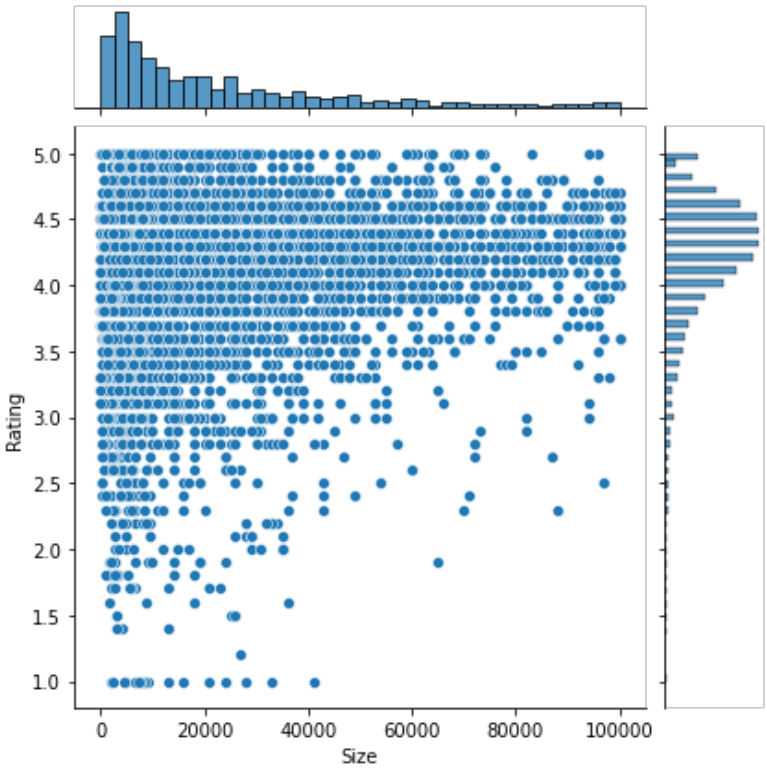
Are heavier apps rated better?

In [ ]:

```
sns.jointplot(x="Size",y="Rating",data=df1)
```

Out[ ]:

```
<seaborn.axisgrid.JointGrid at 0x7f508cda2610>
```

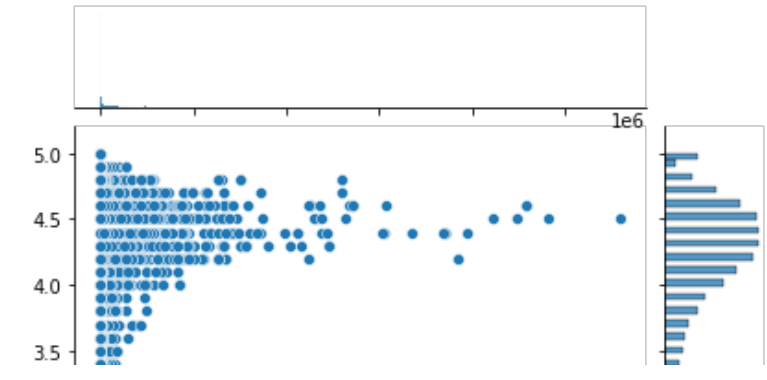


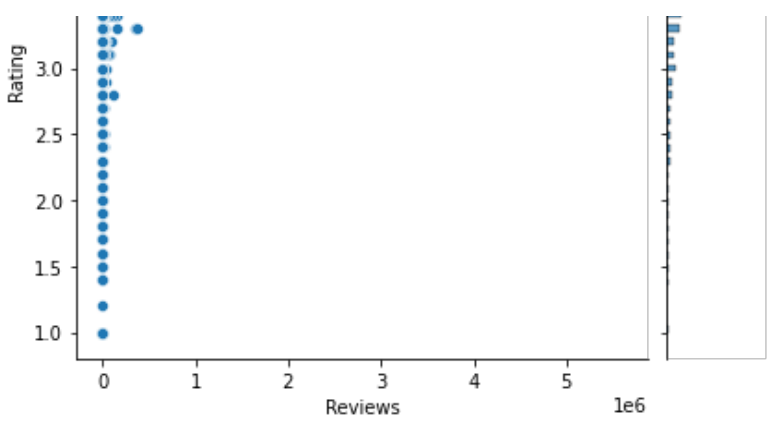
In [ ]:

```
sns.jointplot(x="Reviews",y="Rating",data=df1)
```

Out[ ]:

```
<seaborn.axisgrid.JointGrid at 0x7f508cb85d00>
```





Do more review mean a better rating always?

In [ ]:

```
df1.corr()
```

Out[ ]:

	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 [ ]:

```
df1["Content Rating"].unique()
```

Out[ ]:

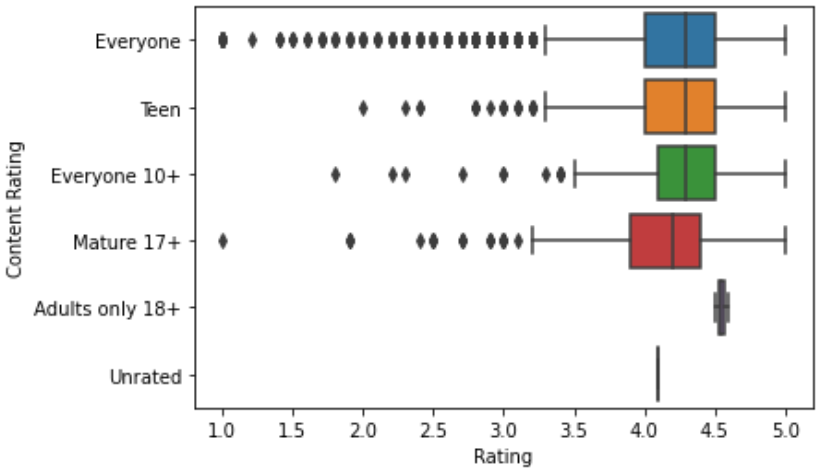
```
array(['Everyone', 'Teen', 'Everyone 10+', 'Mature 17+',
       'Adults only 18+', 'Unrated'], dtype=object)
```

In [ ]:

```
sns.boxplot(x="Rating",y="Content Rating",data=df1)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f508a274fd0>
```

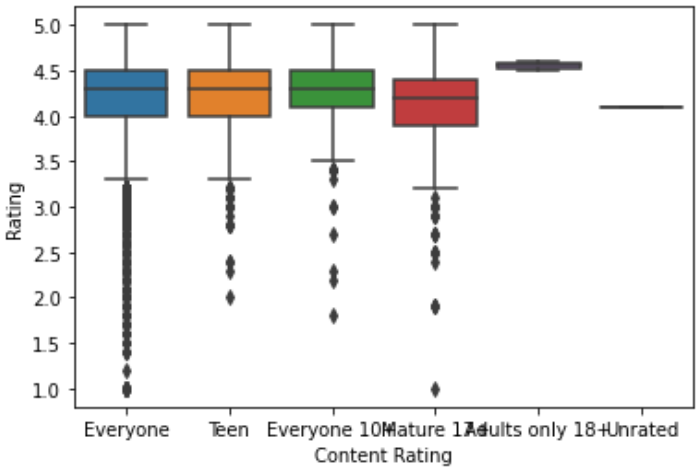


In [ ]:

```
sns.boxplot(x="Content Rating",y="Rating",data=df1)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f508a274190>
```

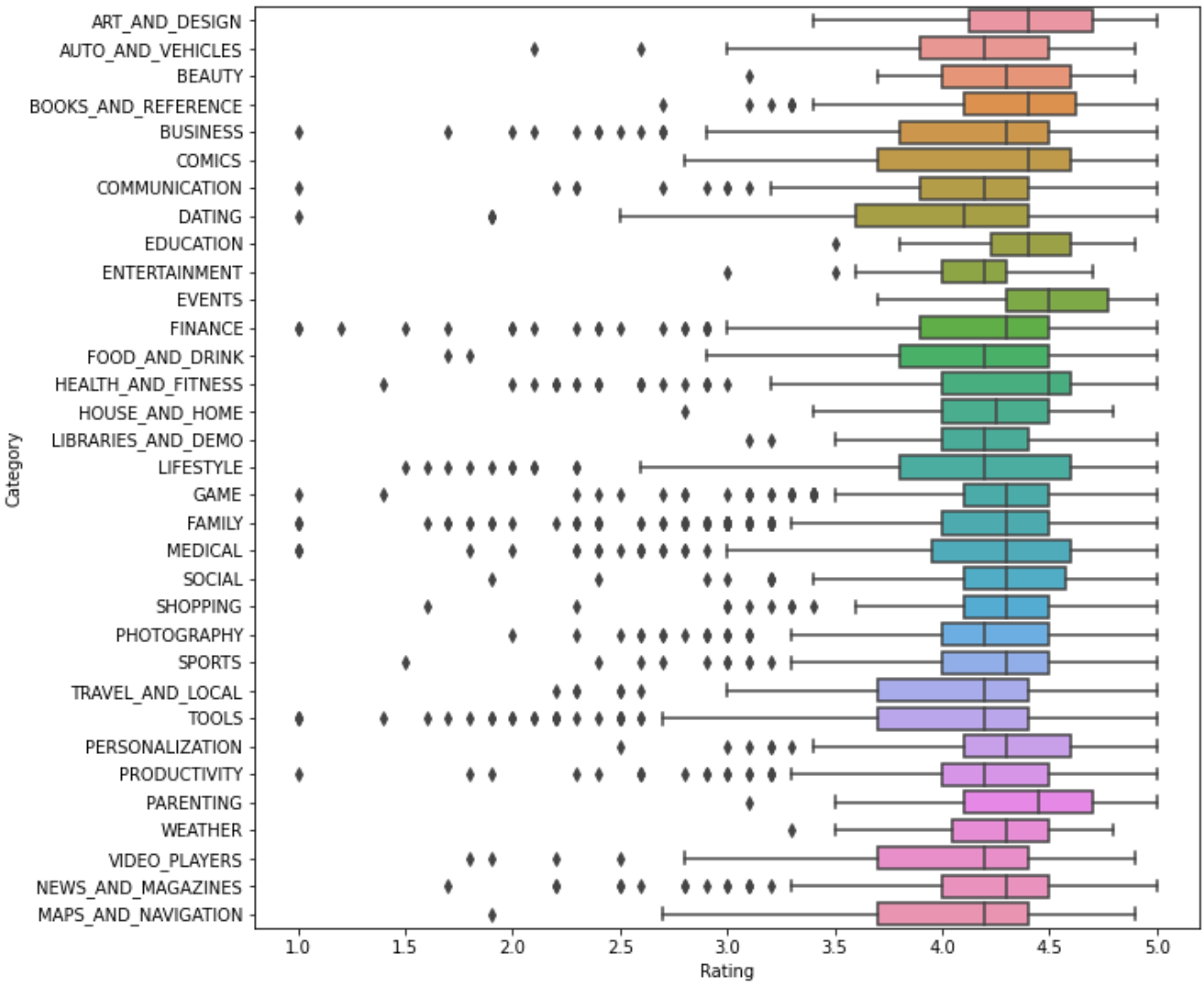


In [ ]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
sns.boxplot(x="Rating",y="Category",data=df1)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5089e26ac0>
```



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 [ ]:

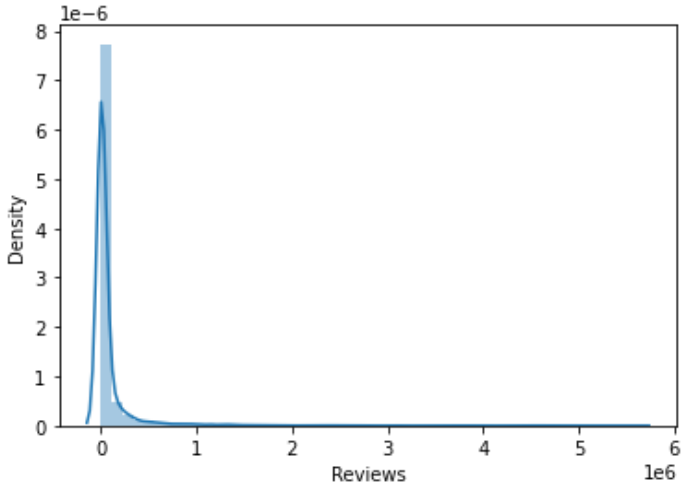
```
inp1 = df1.copy()
```

In [ ]:

```
sns.distplot(inp1["Reviews"])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5089ba6bb0>
```



In [ ]:

```
#plt.hist(inp1[["Reviews"]])
#plt.show()
```

In [ ]:

```
inp1.Reviews=inp1.Reviews.apply(np.log1p)
```

In [ ]:

```
inp1.Installs=inp1.Installs.apply(np.log1p)
```

In [ ]:

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

In [ ]:

```
inp1.shape
```

Out[ ]:

```
(7476, 9)
```

In [ ]:

```
inp1.columns
```

Out[ ]:

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

In [ ]:

In [ ]:

```
inp1["Type"].unique()
```

Out[ ]:

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

In [ ]:

```
inp2 = pd.get_dummies(inp1)
```

In [ ]:

```
inp2.shape
```

Out[ ]:

```
(7476, 158)
```

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

1. **Separate the dataframes into X_train, y_train, X_test, and y_test.**

11 . Model building

Use linear regression as the technique

Report the R2 on the train set

1. **Make predictions on test set and report R2.**

In [ ]:

```
inp2.head(2)
```

Out[ ]:

	Rating	Reviews	Size	Installs	Price	Category_ART_AND_DESIGN	Category_AUTO_AND_VEHICLES	Category_BEAUT
0	4.1	5.075174	19000.0	9.210440	0.0	1	0	
1	3.9	6.875232	14000.0	13.122365	0.0	1	0	

2 rows × 158 columns

In [ ]:

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

In [ ]:

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

In [ ]:

```
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 [ ]:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

In [ ]:

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

Out[ ]:

```
LinearRegression()
```

In [ ]:

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

In [ ]:

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

In [ ]:

```
r2_score(y_test,y_pred)
```

Out[ ]:

```
0.1304405462811623
```

In [ ]:

```
import matplotlib.pyplot as plt
x = [1,2,3,4]
y =x
plt.plot(x,y,linewidth=5,alpha=0.2)
plt.show()
```