In [1]:

```
#1 call
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statistics as stc
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

In [2]:

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

In [4]:

```
print("Count of null values in data")
df.isnull().sum()
```

Out[4]:

```
Count of null values in data
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 [5]:

```
df.dropna(inplace=True)
print("Check for null values after removing nulls")
df.isnull().sum()
```

Out[5]:

```
Check for null values after removing nulls
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 [6]:

```
df=df[-df['Size'].str.contains('Var')]
```

In [7]:

```
 df.loc[:,'SizeNum'] =df.Size.str.rstrip('Mk+')
df.SizeNum=pd.to_numeric(df['SizeNum'])
df.SizeNum.dtype
```

Out[7]:

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

In [8]:

```
df['SizeNum']=np.where(df.Size.str.contains('M'),df.SizeNum*1000, df.SizeNum)
```

In [9]:

```
# Size no more needed, replace it with SizeNum and drop SizeNum
df.Size=df.SizeNum
df.drop('SizeNum',axis=1,inplace=True)
#df
```

In [10]:

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

In [11]:

```
 df.Reviews.dtype
```

Out[11]:

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

In [12]:

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

```
C:\Users\MYPC\AppData\Local\Temp/ipykernel_4732/519759075.py:1: FutureWarnin
g: The default value of regex will change from True to False in a future vers
ion. In addition, single character regular expressions will *not* be treated
as literal strings when regex=True.
  df['Installs']=df.Installs.str.replace("+","")
```

In [13]:

```
 df.Installs=df.Installs.str.replace(",","")
df.Installs=pd.to_numeric(df.Installs)
df.Installs.dtype
```

Out[13]:

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

In [14]:

```
 df.Price=df.Price.str.replace("$","")
df.Price=pd.to_numeric(df.Price)
df.Price.dtype
```

```
C:\Users\MYPC\AppData\Local\Temp/ipykernel_4732/3806789969.py:1: FutureWarnin
g: The default value of regex will change from True to False in a future vers
ion. In addition, single character regular expressions will *not* be treated
as literal strings when regex=True.
  df.Price=df.Price.str.replace("$","")
```

Out[14]:
```
dtype('float64')
```

In [15]:
```
 df=df[(df.Rating>=1) & (df.Rating<=5) ]
```

In [16]:
```
 len(df.index)
```

Out[16]:
```
7723
```

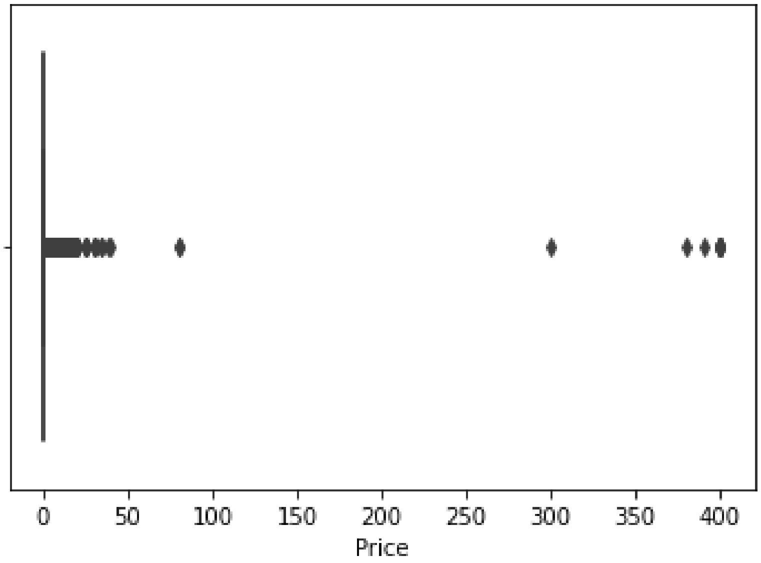
In [17]:
```
 df.drop(df.index[df.Reviews>df.Installs],axis=0,inplace=True)
len(df.index)
```

Out[17]:
```
7717
```

In [19]:
```
 index_free_and_price_gt_0=df.index[((df.Type=='Free')&(df.Price>0))]
if len(index_free_and_price_gt_0)>0:
    print("Dropping following indices:",index_free_and_price_gt_0)
    df.drop(index_free_and_price_gt_0,axis=0,inplace=True)
else:
    print("There is no Free Apps with price >0")
```

```
There is no Free Apps with price >0
```

In [20]:
```
 ax = sns.boxplot(x='Price', data=df)
```



In [22]:
```
 price_std=stc.stdev(df.Price)
```

```
price_std
```

Out[22]:

```
17.414783874309933
```

In [23]:

```
 price_mean=stc.mean(df.Price)
price_mean
```

Out[23]:

```
1.128724893093171
```

In [24]:

```
price_outlier_uplimit=price_mean+3*price_std
price_outlier_uplimit
```

Out[24]:

```
53.37307651602297
```

In [25]:

```
 #price_outlier_downlimit=price_mean-3*price_std
#price_outlier_downlimit
#df[df.Price>price_outlier_uplimit]
print("# of upper outliers is ",len(df[(df.Price>price_outlier_uplimit) ]))
```

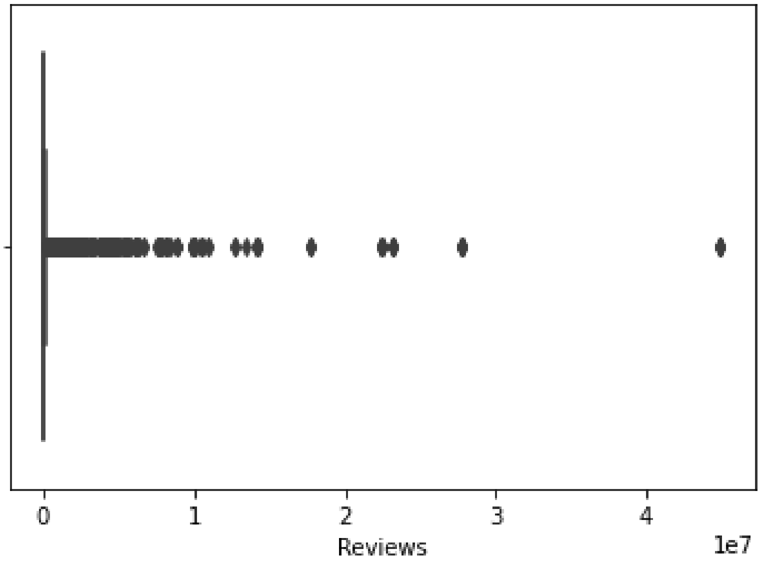
```
# of upper outliers is  17
```

In [26]:

```
 #df[df.Price<price_outlier_downlimit]
#print("# of lower outliers is ",len(df[df.Price<price_outlier_downlimit]))
sns.boxplot(x='Reviews',data=df)
```

Out[26]:

```
<AxesSubplot:xlabel='Reviews'>
```



In [27]:

```
 rev_std=stc.stdev(df.Reviews)
rev_std
```

```
1864639.6094670836
```

Out[27]:

In [29]:

```
rev_mean=stc.mean(df.Reviews)
rev_mean
```

Out[29]:

```
295127.5482700531
```

In [30]:

```
rev_outlier_uplimit=rev_mean+3*rev_std
rev_outlier_uplimit
```

Out[30]:

```
5889046.376671304
```

In [31]:

```
 rev_outlier_downlimit=rev_mean-3*rev_std
rev_outlier_downlimit
```

Out[31]:

```
-5298791.280131198
```

In [32]:

```
#df[df.Reviews>rev_outlier_uplimit]
print("# of upper outliers is ",len(df[(df.Reviews>rev_outlier_uplimit) ]))
```

```
# of upper outliers is  89
```

In [33]:

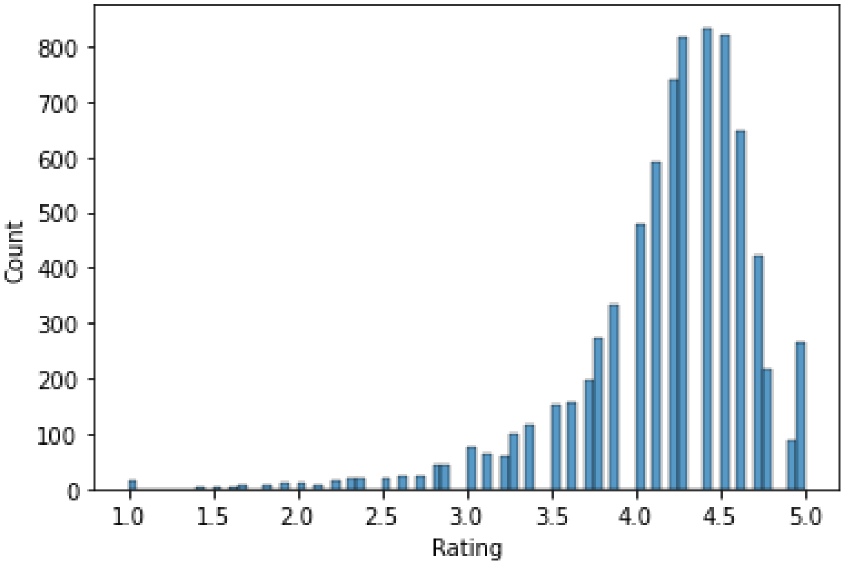
```
# Since reviews cannot be less than 1, no need to check lower outliers
# remove outliers
#df.drop(df.index[(df.Reviews>rev_outlier_uplimit) ],inplace=True)
#len(df.index)
```

In [34]:

```
 #sns.boxplot(x='Rating',data=df)
sns.histplot(x='Rating',data=df)
 #rating_std=stc.stdev(df.Rating)
#rating_std
#rating_mean=stc.mean(df.Rating)
#rating_mean
#rating_outlier_uplimit=rating_mean+3*rating_std
#rating_outlier_uplimit
 #rating_outlier_downlimit=rating_mean-3*rating_std
#rating_outlier_downlimit
# Since max possible value of rating (5) is less than upper limit, no need to
#df[df.Rating<rating_outlier_downlimit]
#print("# of lower outliers is ",len(df[(df.Rating<rating_outlier_downlimit)
#df.drop(df.index[(df.Rating<rating_outlier_downlimit)],inplace=True)
#len(df.index)
```

Out[34]:

```
<AxesSubplot:xlabel='Rating', ylabel='Count'>
```

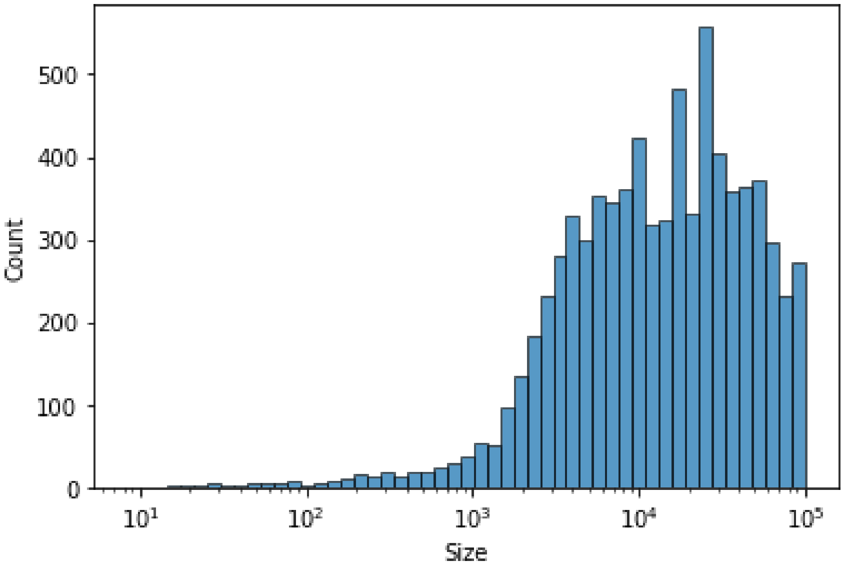


In [35]:

```
# use log scale to make histogram more representable
sns.histplot(x='Size',data=df,log_scale=True)
```

Out[35]:

```
<AxesSubplot:xlabel='Size', ylabel='Count'>
```



In [36]:

```
df[df.Price>=200]
```

Out[36]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating
4197	most expensive app (H)	FAMILY	4.3	6	1500.0	100	Paid	399.99	Everyone
4362	💎 I'm rich	LIFESTYLE	3.8	718	26000.0	10000	Paid	399.99	Everyone

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating
4367	I'm Rich - Trump Edition	LIFESTYLE	3.6	275	7300.0	10000	Paid	400.00	Everyone
5351	I am rich	LIFESTYLE	3.8	3547	1800.0	100000	Paid	399.99	Everyone
5354	I am Rich Plus	FAMILY	4.0	856	8700.0	10000	Paid	399.99	Everyone
5355	I am rich VIP	LIFESTYLE	3.8	411	2600.0	10000	Paid	299.99	Everyone
5356	I Am Rich Premium	FINANCE	4.1	1867	4700.0	50000	Paid	399.99	Everyone
5357	I am extremely Rich	LIFESTYLE	2.9	41	2900.0	1000	Paid	379.99	Everyone
5358	I am Rich!	FINANCE	3.8	93	22000.0	1000	Paid	399.99	Everyone
5359	I am rich(premium)	FINANCE	3.5	472	965.0	5000	Paid	399.99	Everyone
5362	I Am Rich Pro	FAMILY	4.4	201	2700.0	5000	Paid	399.99	Everyone
5364	I am rich (Most expensive app)	FINANCE	4.1	129	2700.0	1000	Paid	399.99	Teen
5366	I Am Rich	FAMILY	3.6	217	4900.0	10000	Paid	389.99	Everyone
5369	I am Rich	FINANCE	4.3	180	3800.0	5000	Paid	399.99	Everyone
5373	I AM RICH PRO PLUS	FINANCE	4.0	36	41000.0	1000	Paid	399.99	Everyone

In [37]:

```
print("# of Apps with price >= 200 = ",len(df[(df.Price>=200) ]))
```

```
# of Apps with price >= 200 =  15
```

In [38]:

```
df.drop(df.index[(df.Price>=200)], inplace=True)
len(df.index)
```

Out[38]:

```
7702
```

In [39]:

```
df.drop(df.index[(df.Reviews>=2000000)], inplace=True)
len(df.index)
```

Out[39]:

```
7483
```

In [40]:

```
 install_10_perc=np.percentile(df.Installs, 10)
install_10_perc
```

Out[40]:

```
1000.0
```

In [41]:

```
 install_25_perc=np.percentile(df.Installs, 25)
install_25_perc
```

Out[41]:

```
10000.0
```

In [42]:

```
install_50_perc=np.percentile(df.Installs, 50)
install_50_perc
```

Out[42]:

```
100000.0
```

In [43]:

```
install_70_perc=np.percentile(df.Installs, 70)
install_70_perc
```

Out[43]:

```
1000000.0
```

In [44]:

```
 install_90_perc=np.percentile(df.Installs,90)
install_90_perc
```

Out[44]:

```
10000000.0
```

In [45]:

```
 install_95_perc=np.percentile(df.Installs,95)
install_95_perc
```

Out[45]:

```
10000000.0
```

In [46]:

```
 install_99_perc=np.percentile(df.Installs,99)
install_99_perc
```

Out[46]:

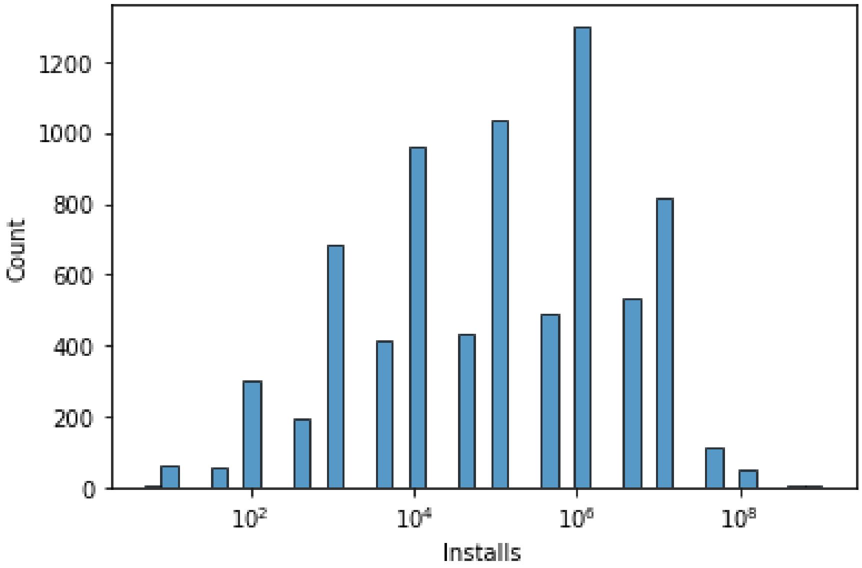
```
50000000.0
```

In [47]:

```
sns.histplot(data=df,x='Installs',log_scale=True)
```

Out[47]:

```
<AxesSubplot:xlabel='Installs', ylabel='Count'>
```



In [48]:

```
print("As result, ",len(df[df.Installs >= install_99_perc])," will be droppe
```

```
As result,  176  will be dropped
```

In [49]:

```
 df.drop(df.index[df.Installs >= install_99_perc],inplace=True)
len(df.index)
```

Out[49]:

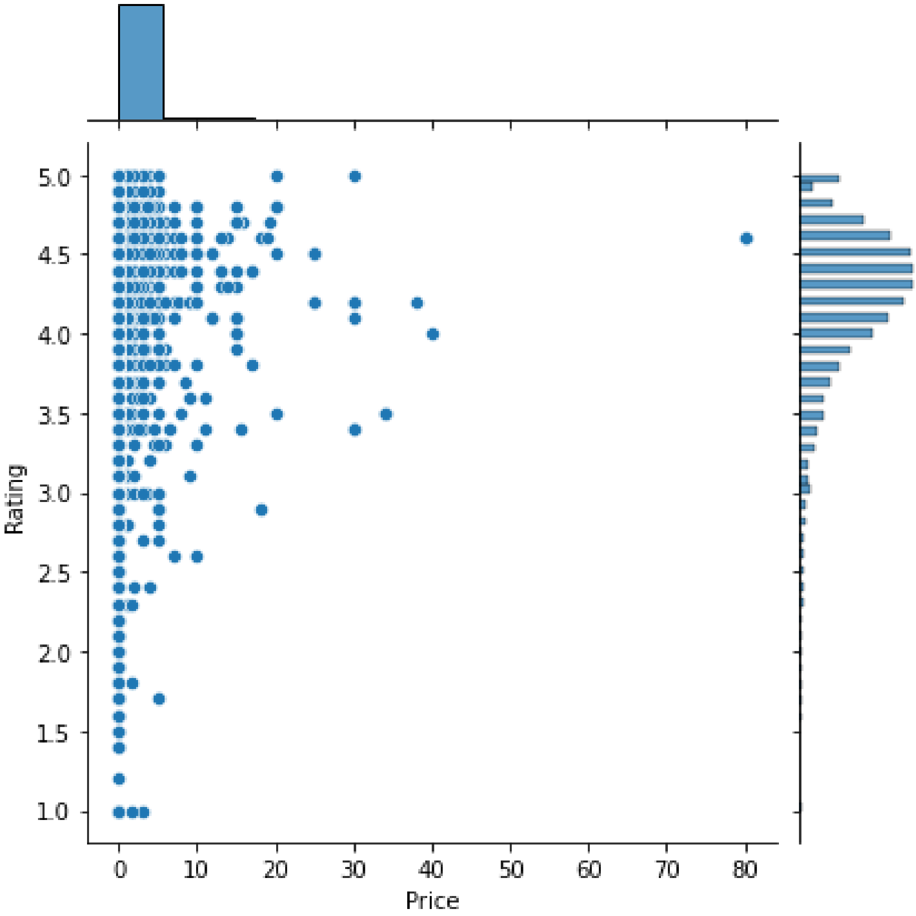
```
7307
```

In [50]:

```
sns.jointplot(data=df,y='Rating',x='Price')
```

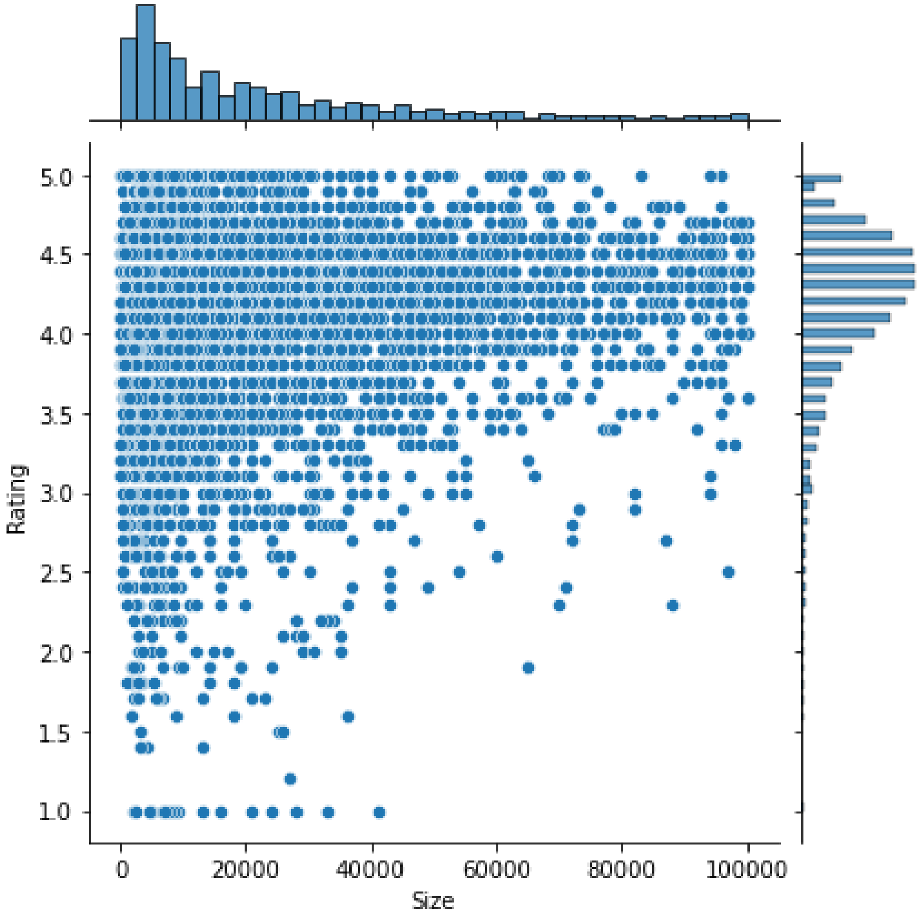
Out[50]:

```
<seaborn.axisgrid.JointGrid at 0x1b87a920ca0>
```



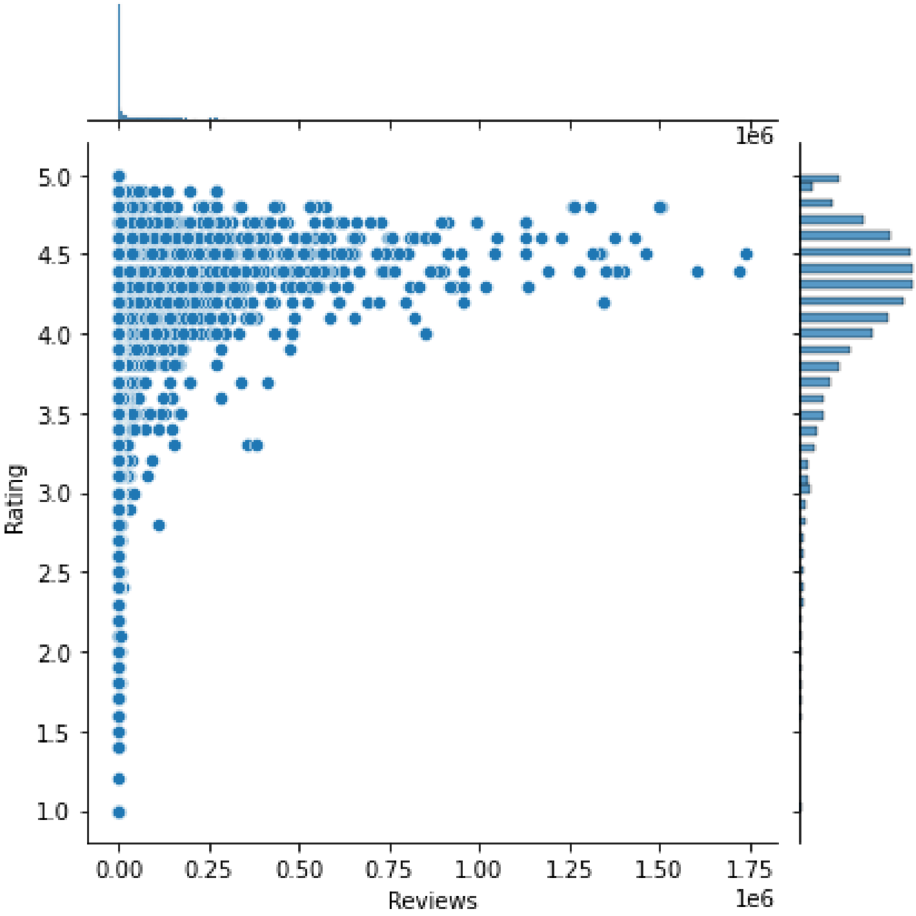
In [51]:
```
sns.jointplot(data=df,y='Rating',x='Size')
```

Out[51]:
```
<seaborn.axisgrid.JointGrid at 0x1b87aad5640>
```



In [52]:
```
sns.jointplot(data=df,y='Rating',x='Reviews')
```

Out[52]:
```
<seaborn.axisgrid.JointGrid at 0x1b87c197160>
```



In [53]:

```
df['Content Rating'].unique()
```

Out[53]:

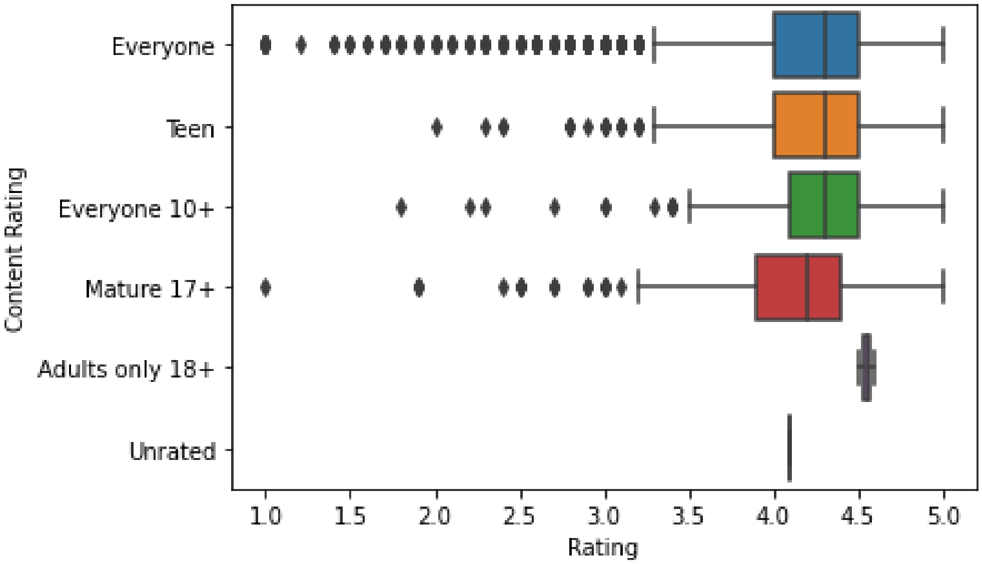
```
array(['Everyone', 'Teen', 'Everyone 10+', 'Mature 17+',
       'Adults only 18+', 'Unrated'], dtype=object)
```

In [54]:

```
sns.boxplot(data=df,x='Rating',y='Content Rating')
```

Out[54]:

```
<AxesSubplot:xlabel='Rating', ylabel='Content Rating'>
```

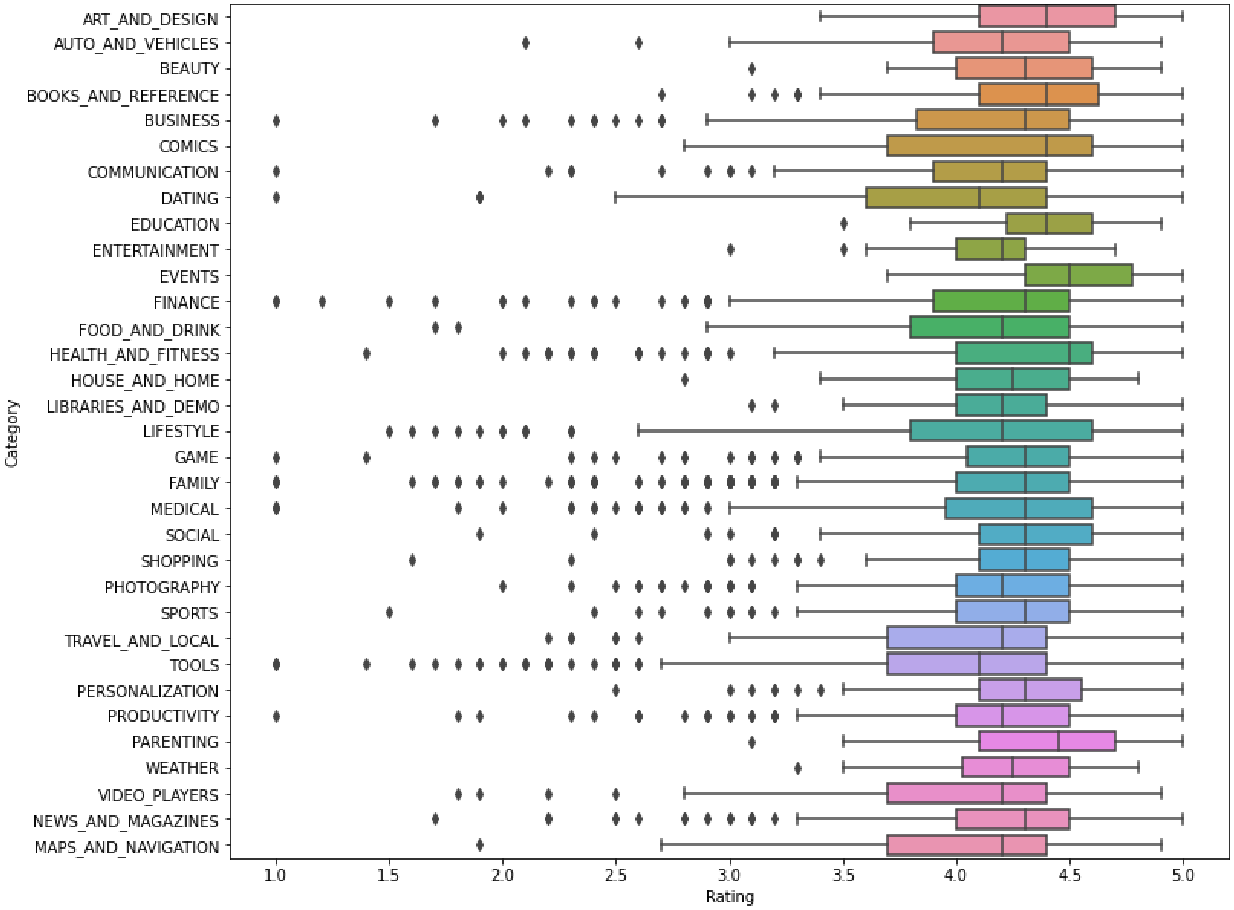


In [55]:

```
 a4_dims = (11.7, 10.27)
fig, ax = plt.subplots(figsize=a4_dims)
sns.boxplot(data=df,x='Rating',y='Category',ax=ax)
```

Out[55]:

```
<AxesSubplot:xlabel='Rating', ylabel='Category'>
```



In [56]:

```
 #8.1
inp1=df.copy()
inp1.Reviews=inp1.Reviews.apply(np.log1p)
```

In [57]:

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

In [60]:

```
#8.2
inp1.drop(columns=['App','Last Updated','Current Ver','Android Ver'],inplace=
```

In [61]:

```
 inp1.shape
```

Out[61]:

```
(7307, 9)
```

In [62]:

```
#8.3
```

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

In [63]:

```
inp2.shape
```

Out[63]:

```
(7307, 158)
```

In [64]:

```
 data = inp2.drop(columns='Rating')
data.shape
```

Out[64]:

```
(7307, 157)
```

In [65]:

```
 target = pd.DataFrame(inp2.Rating)
target.shape
```

Out[65]:

```
(7307, 1)
```

In [68]:

```
x_train, x_test, y_train, y_test = train_test_split(data, target, test_size=0
print("x_train shape is ", x_train.shape)
print("y_train shape is ", y_train.shape)
print("x_test shape is ", x_test.shape)
print("y_test shape is ", y_test.shape)
```

```
x_train shape is  (5114, 157)
y_train shape is  (5114, 1)
x_test shape is  (2193, 157)
y_test shape is  (2193, 1)
```

In [69]:

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

Out[69]:

```
LinearRegression()
```

In [70]:

```
 train_pred=model.predict(x_train)
```

In [71]:

```
print("R2 value of the model(by train) is ", r2_score(y_train, train_pred))
```

```
R2 value of the model(by train) is  0.15264772134593874
```

In [72]:

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

In [73]:

```
 print("R2 value of the model(by test) is ", r2_score(y_test, test_pred))
```

```
R2 value of the model(by test) is  0.14262263030973144
```