App Rating Prediction

December 20, 2023

1 App Rating Prediction

1.1 Description:

1.1.1 Objective: Make a model to predict the app rating, with other information about the app provided.

1.1.2 Problem Statement:

Google Play Store team is about to launch a new feature wherein, certain apps that are promising, are boosted in visibility. The boost will manifest in multiple ways including higher priority in recommendations sections ("Similar apps", "You might also like", "New and updated games"). These will also get a boost in search results visibility. This feature will help bring more attention to newer apps that have the potential.

1.1.3 Domain: General

Analysis to be done: The problem is to identify the apps that are going to be good for Google to promote. App ratings, which are provided by the customers, is always a great indicator of the goodness of the app. The problem reduces to: predict which apps will have high ratings.

1.1.4 Content: Dataset: Google Play Store data ("googleplaystore.csv")

Fields in the data – App: Application name

Category: Category to which the app belongs

Rating: Overall user rating of the app

Reviews: Number of user reviews for the app

Size: Size of the app

Installs: Number of user downloads/installs for the app

Type: Paid or Free

Price: Price of the app

Content Rating: Age group the app is targeted at - Children / Mature 21+ / Adult

Genres: An app can belong to multiple genres (apart from its main category). For example, a musical family game will belong to Music, Game, Family genres.

Last Updated: Date when the app was last updated on Play Store

Current Ver: Current version of the app available on Play Store

Android Ver: Minimum required Android version

[2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

2 Steps to perform:

2.1 1.Load the data file using pandas.

[3]:
```
data = pd.read_csv("googleplaystore.csv")
```

[4]:
```
data.head()
```

[4]:
```
                              App Category  Rating  Reviews  Size  Installs  \
0            Star Wars  : DIRTY     GAME     4.5    38207   15M  100,000+
1                  ASCCP Mobile  MEDICAL     4.5       63   25M   10,000+
2       Diabetes & Diet Tracker  MEDICAL     4.6      395   19M    1,000+
3  Critical Care Paramedic Review  MEDICAL     4.4       17  1.8M    1,000+
4          InfantRisk Center HCP  MEDICAL     2.6       41   14M    1,000+

   Type  Price Content Rating        Genres      Last Updated Current Ver  \
0  Paid  $9.99           Teen  Role Playing  October 19, 2015       1.0.6
1  Paid  $9.99       Everyone       Medical   October 3, 2016       2.1.1
2  Paid  $9.99       Everyone       Medical     July 16, 2018       6.5.1
3  Paid  $9.99   Everyone 10+       Medical      May 14, 2018       2.0.5
4  Paid  $9.99       Everyone       Medical       May 6, 2015       1.3.4

    Android Ver
0    4.1 and up
1    2.2 and up
2    5.0 and up
3   4.4W and up
4  2.3.3 and up
```

[5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10840 entries, 0 to 10839
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   App             10840 non-null  object
```

```
 1   Category        10840 non-null  object
 2   Rating          9366 non-null   float64
 3   Reviews         10840 non-null  int64
 4   Size            10840 non-null  object
 5   Installs        10840 non-null  object
 6   Type            10839 non-null  object
 7   Price           10840 non-null  object
 8   Content Rating  10840 non-null  object
 9   Genres          10840 non-null  object
 10  Last Updated    10840 non-null  object
 11  Current Ver     10832 non-null  object
 12  Android Ver     10838 non-null  object
dtypes: float64(1), int64(1), object(11)
memory usage: 1.1+ MB
```

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

[6]:
```
data.isnull().sum(axis=0)
```

[6]:
```
App                0
Category           0
Rating          1474
Reviews            0
Size               0
Installs           0
Type               1
Price              0
Content Rating     0
Genres             0
Last Updated       0
Current Ver        8
Android Ver        2
dtype: int64
```

[7]:
```
data.shape
```

[7]:
```
(10840, 13)
```

2.1.2 3.Drop records with nulls in any of the columns.

[8]:
```
data.dropna(how='any', inplace =True)
```

[9]:
```
data.isnull().sum()
```

[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
```

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

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

3.0.2 i. Extract the numeric value from the column

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

[10]:
```
def change_size(Size):
    if 'M' in Size:
        x=Size[:-1]
        x=float(x)*1000
        return x
    elif 'K'==Size[:-1]:
        x=size[:-1]
        x=float(x)
        return x
    else:
        return 0
```

[11]:
```
data["Size"]=data["Size"].map(change_size)
```

[12]:
```
data["Size"]
```

[12]:
```
0        15000.0
1        25000.0
2        19000.0
3         1800.0
4        14000.0
          ...
10833     2600.0
10835    53000.0
```

```
10836     3600.0
10838        0.0
10839    19000.0
Name: Size, Length: 9360, dtype: float64
```

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

[13]:
```
data.Reviews = data.Reviews.astype("int")
```

[14]:
```
data.Reviews.describe()
```

[14]:
```
count    9.360000e+03
mean     5.143767e+05
std      3.145023e+06
min      1.000000e+00
25%      1.867500e+02
50%      5.955000e+03
75%      8.162750e+04
max      7.815831e+07
Name: Reviews, dtype: float64
```

3.0.5 Installs field is currently stored as string and has values like 1,00,000+.

3.0.6 Treat 1,00,000+ as 1,00,000

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

[15]:
```
data.Installs.value_counts()
```

[15]:
```
1,000,000+        1576
10,000,000+       1252
100,000+          1150
10,000+           1009
5,000,000+         752
1,000+             712
500,000+           537
50,000+            466
5,000+             431
100,000,000+       409
100+               309
50,000,000+        289
500+               201
500,000,000+        72
10+                 69
1,000,000,000+      58
50+                 56
```

```
5+                    9
1+                    3
Name: Installs, dtype: int64
```

[16]:
```
def clen_installs(val):
    return int(val.replace(",","").replace("+",""))
```

[17]:
```
data.Installs=data.Installs.map(clen_installs)
```

[18]:
```
data.Installs.describe()
```

[18]:
```
count    9.360000e+03
mean     1.790875e+07
std      9.126637e+07
min      1.000000e+00
25%      1.000000e+04
50%      5.000000e+05
75%      5.000000e+06
max      1.000000e+09
Name: Installs, dtype: float64
```

Price field is a string and has $ symbol. Remove '$' sign, and convert it to numeric.

[19]:
```
data['Price']
```

[19]:
```
0        $9.99
1        $9.99
2        $9.99
3        $9.99
4        $9.99
         …
10833        0
10835        0
10836        0
10838        0
10839        0
Name: Price, Length: 9360, dtype: object
```

[20]:
```
data['Price'] = [value.replace('$', '') for value in data['Price']]
```

[21]:
```
data['Price'].unique()
```

[21]:
```
array(['9.99', '9.00', '8.99', '8.49', '79.99', '7.99', '7.49', '6.99',
       '6.49', '5.99', '5.49', '400.00', '4.99', '4.84', '4.77', '4.60',
       '4.59', '4.49', '4.29', '399.99', '39.99', '389.99', '379.99',
       '37.99', '33.99', '3.99', '3.95', '3.90', '3.88', '3.49', '3.28',
       '3.08', '3.04', '3.02', '299.99', '29.99', '24.99', '2.99', '2.95',
```

```
'2.90', '2.59', '2.56', '2.50', '2.49', '2.00', '19.99', '19.40',
'18.99', '17.99', '16.99', '15.99', '15.46', '14.99', '14.00',
'13.99', '12.99', '11.99', '10.99', '10.00', '1.99', '1.97',
'1.76', '1.75', '1.70', '1.61', '1.59', '1.50', '1.49', '1.29',
'1.20', '1.00', '0.99', '0'], dtype=object)
```

[22]:
```
data.dtypes
```

[22]:
```
App                object
Category           object
Rating            float64
Reviews             int64
Size              float64
Installs            int64
Type               object
Price              object
Content Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
dtype: object
```

[23]:
```
data['Price'] = data['Price'].apply(lambda x: float(x))
```

[24]:
```
data.dtypes
```

[24]:
```
App                object
Category           object
Rating            float64
Reviews             int64
Size              float64
Installs            int64
Type               object
Price             float64
Content Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
dtype: object
```

[25]:
```
data['Price']
```

[25]:
```
0        9.99
1        9.99
2        9.99
```

```
3        9.99
4        9.99
         …
10833    0.00
10835    0.00
10836    0.00
10838    0.00
10839    0.00
Name: Price, Length: 9360, dtype: float64
```

[26]:
```
data.Price.describe()
```

[26]:
```
count    9360.000000
mean        0.961279
std        15.821640
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max       400.000000
Name: Price, dtype: float64
```

[27]:
```
data.head()
```

[27]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
0	Star Wars : DIRTY	GAME	4.5	38207	15000.0	100000	Paid	9.99	Teen	Role Playing	October 19, 2015	1.0.6	4.1 and up
1	ASCCP Mobile	MEDICAL	4.5	63	25000.0	10000	Paid	9.99	Everyone	Medical	October 3, 2016	2.1.1	2.2 and up
2	Diabetes & Diet Tracker	MEDICAL	4.6	395	19000.0	1000	Paid	9.99	Everyone	Medical	July 16, 2018	6.5.1	5.0 and up
3	Critical Care Paramedic Review	MEDICAL	4.4	17	1800.0	1000	Paid	9.99	Everyone 10+	Medical	May 14, 2018	2.0.5	4.4W and up
4	InfantRisk Center HCP	MEDICAL	2.6	41	14000.0	1000	Paid	9.99	Everyone	Medical	May 6, 2015	1.3.4	2.3.3 and up

3.1 Sanity Check

3.1.1 Average rating should be between 1 and 5 as only these values are allowed on the play store.

3.1.2 Drop the rows that have a value outside this range.

[28]:
```
data.Rating.describe()
```

[28]:
```
count    9360.000000
mean        4.191838
std         0.515263
min         1.000000
25%         4.000000
50%         4.300000
75%         4.500000
max         5.000000
Name: Rating, dtype: float64
```

Min value of ratings is 1 and the max values is 5. As the ratings are within range,

we do not drop any record here.

Reviews should not be more than installs as only those who installed can review the app.

if there are any such records,drop them.

[29]:
```
len(data[data.Reviews>data.Installs])
```

[29]:
```
7
```

[30]:
```
data[data.Reviews>data.Installs]
```

[30]:

	App	Category	Rating	Reviews	Size	\
448	Alarmy (Sleep If U Can) - Pro	LIFESTYLE	4.8	10249	0.0	
621	Ra Ga Ba	GAME	5.0	2	20000.0	
798	Mu.F.O.	GAME	5.0	2	16000.0	
3103	KBA-EZ Health Guide	MEDICAL	5.0	4	25000.0	
7030	Brick Breaker BR	GAME	5.0	7	19000.0	
7673	Trovami se ci riesci	GAME	5.0	11	6100.0	
8759	DN Blog	SOCIAL	5.0	20	4200.0	

	Installs	Type	Price	Content Rating	Genres	Last Updated	\
448	10000	Paid	2.49	Everyone	Lifestyle	July 30, 2018	
621	1	Paid	1.49	Everyone	Arcade	February 8, 2017	
798	1	Paid	0.99	Everyone	Arcade	March 3, 2017	
3103	1	Free	0.00	Everyone	Medical	August 2, 2018	

```
7030         5  Free    0.00       Everyone     Arcade     July 23, 2018
7673        10  Free    0.00       Everyone     Arcade    March 11, 2017
8759        10  Free    0.00           Teen     Social     July 23, 2018

             Current Ver         Android Ver
448   Varies with device  Varies with device
621               1.0.4         2.3 and up
798                   1         2.3 and up
3103            1.0.72       4.0.3 and up
7030                  1         4.1 and up
7673                0.1         2.3 and up
8759                  1         4.0 and up
```

[31]:
```
data1=data[data.Reviews<= data.Installs].copy()
```

[32]:
```
data1.shape
```

[32]:
```
(9353, 13)
```

3.1.3 So data1 is my main dataset for usage as it contains the information of only those apps

3.1.4 which do not have suspicious reviews.

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

[33]:
```
len(data[(data.Type =='free')&(data.Price>0)])
```

[33]:
```
0
```

3.1.6 No such 'free' app for which a price is charged

3.1.7 Performing univariate analysis:

3.1.8 Boxplot for price

3.1.9 Are there any outliers? Think about the price of usual apps on play store.

[34]:
```
data1['Price']
```

[34]:
```
0        9.99
1        9.99
2        9.99
3        9.99
4        9.99
        ...
10833    0.00
10835    0.00
10836    0.00
```

```
10838    0.00
10839    0.00
Name: Price, Length: 9353, dtype: float64
```

[35]:
```
data1['Price'].min()
```

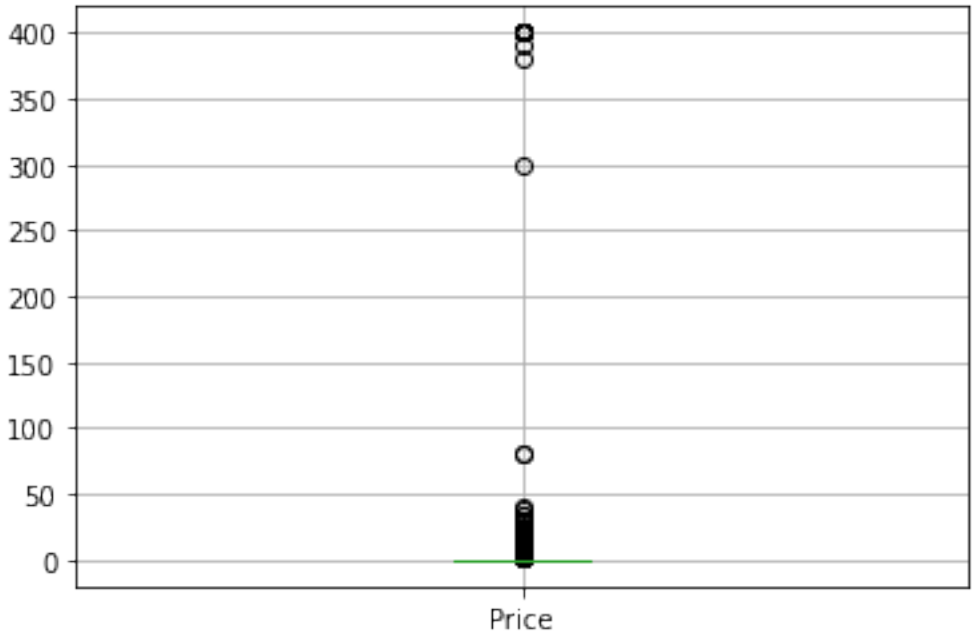
[35]:
```
0.0
```

[36]:
```
data1['Price'].max()
```

[36]:
```
400.0
```

[37]:
```
data1.boxplot(column=["Price"])
```

[37]:
```
<AxesSubplot: >
```



3.1.10 Remove outliers

3.1.11 easy way to remove outliers

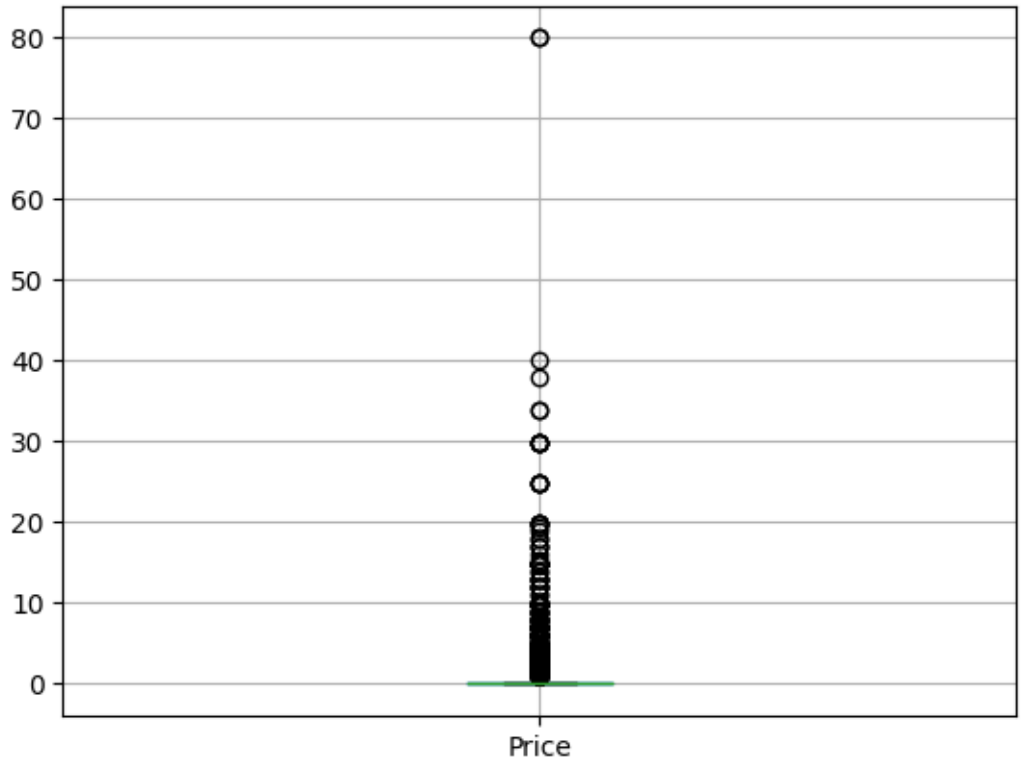
[38]:
```
def Removeoutlier(data1,age):
    Q1 = data[age].quantile(0.25)
    Q3 = data[age].quantile(0.75)
    IQR = Q3 - Q1
    data1 = data1.loc[~((data1[age]<(Q1 - 1.5 * IQR))|(data1[age]>(Q3+1.5 *␣
 ↪IQR))),]
```

```
    return data1
```

[39]:
```
data1 = data1.loc[data1["Price"]<200,]
```

[40]:
```
data1.boxplot(column=["Price"])
```

[40]: <AxesSubplot: >

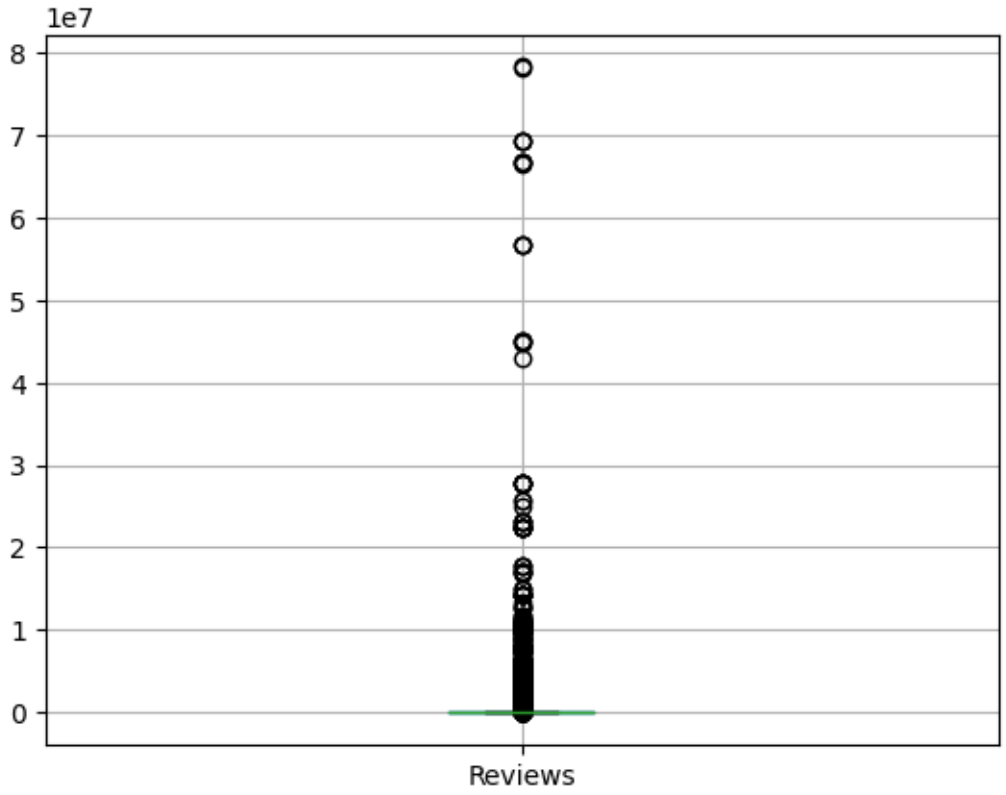


3.1.12 Boxplot for Reviews

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

[41]:
```
data1.boxplot(column=["Reviews"])
```

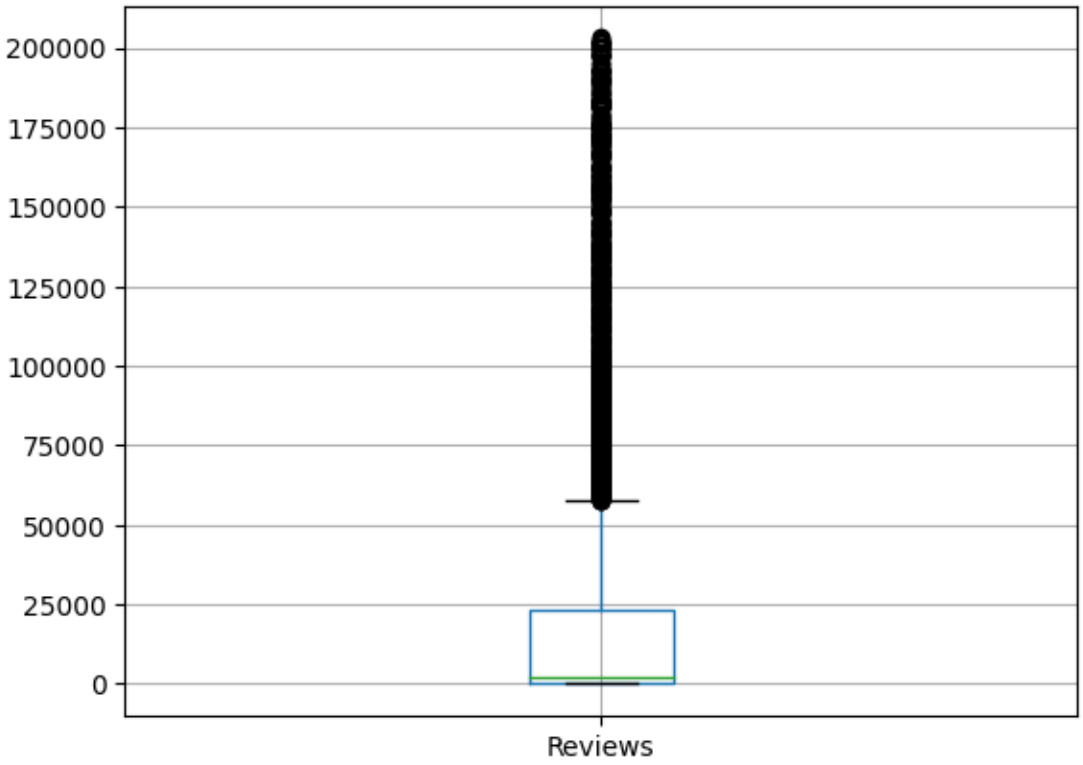
[41]: <AxesSubplot: >



[42]:
```
data1 = Removeoutlier(data1, "Reviews")
```

[43]:
```
data1.boxplot(column=["Reviews"])
```

[43]: <AxesSubplot: >

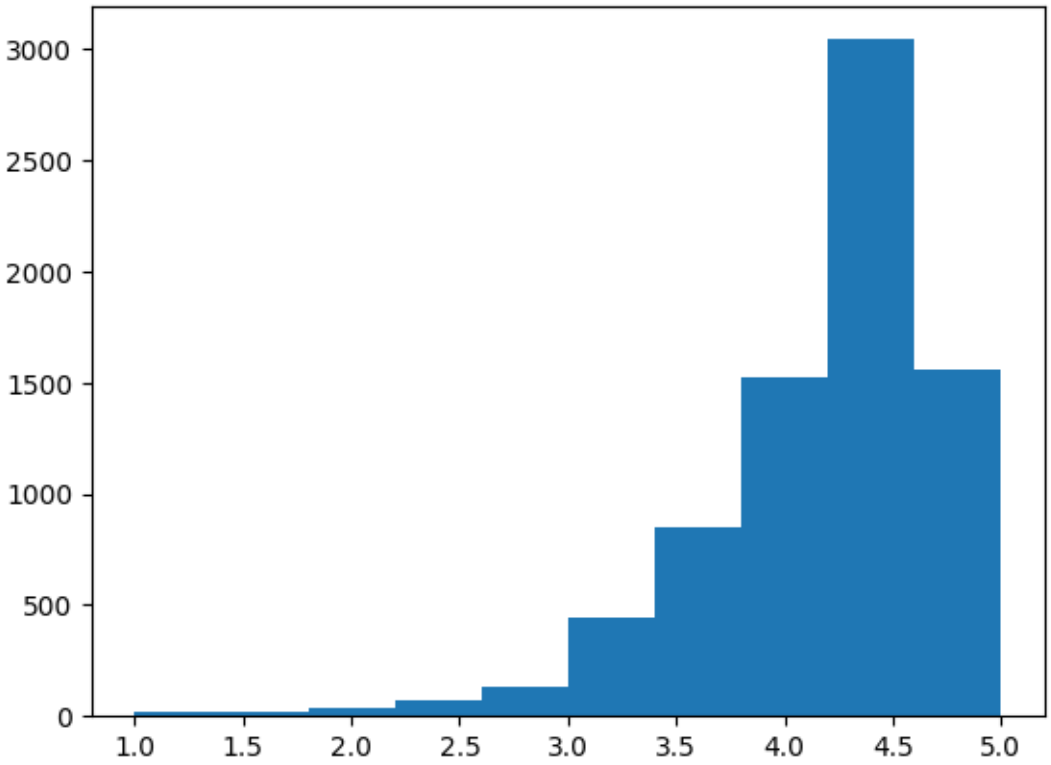


3.1.14 Histogram for Rating

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

[44]:
```
plt.hist(data1["Rating"])
```

[44]:
```
(array([  17.,   18.,   41.,   74.,  136.,  442.,  853., 1525., 3043.,
        1555.]),
 array([1. , 1.4, 1.8, 2.2, 2.6, 3. , 3.4, 3.8, 4.2, 4.6, 5. ]),
 <BarContainer object of 10 artists>)
```

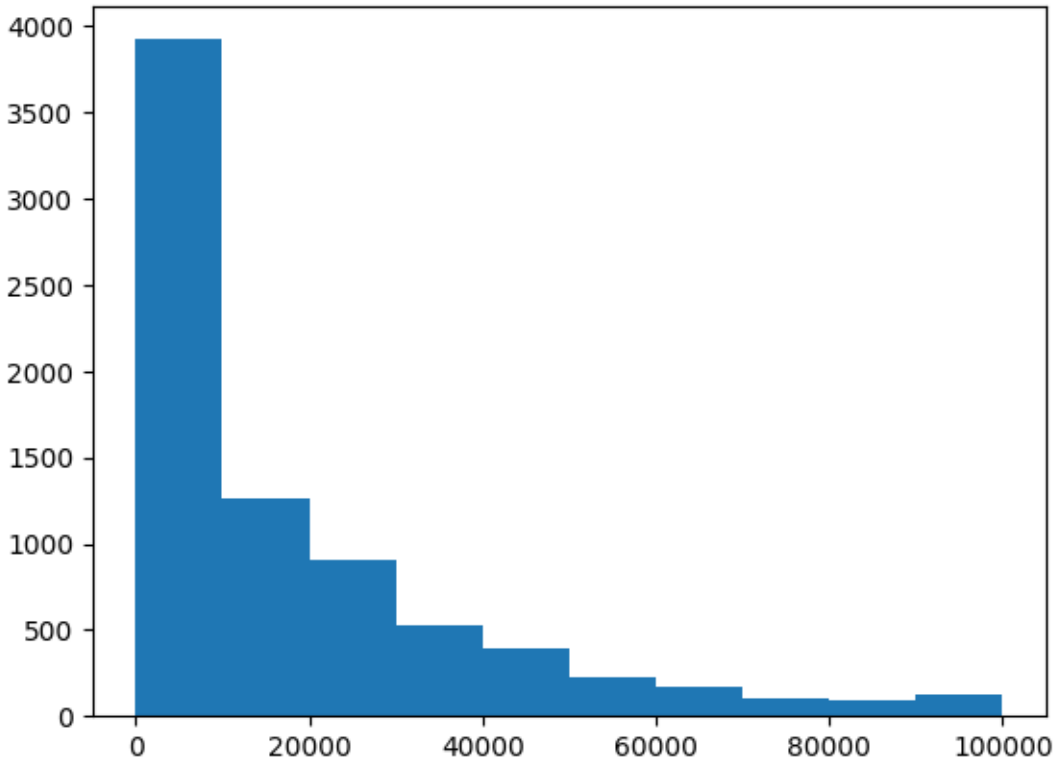


3.1.16 Histogram for size

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

[45]:
```
plt.hist(data1["Size"])
```

[45]:
```
(array([3922., 1259.,  908.,  523.,  388.,  221.,  165.,   99.,   91.,
         128.]),
 array([     0.,  10000.,  20000.,  30000.,  40000.,  50000.,  60000.,
         70000.,  80000.,  90000., 100000.]),
 <BarContainer object of 10 artists>)
```

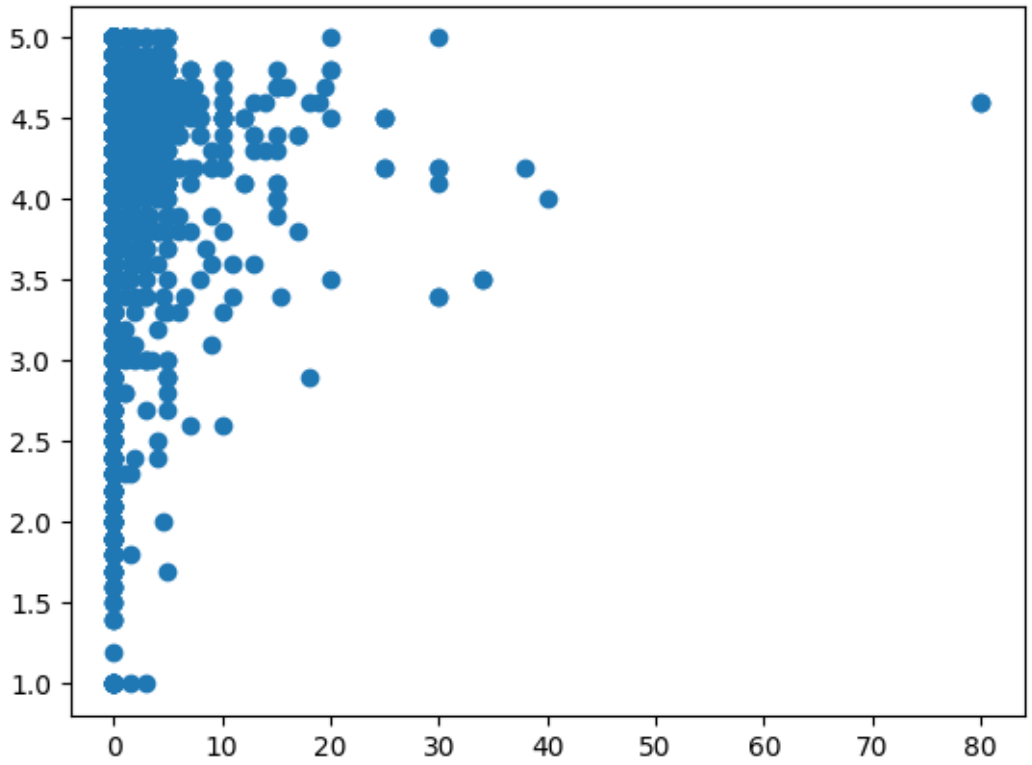


3.1.18 make scatter plot/joinplot for Rating vs. Price

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

[46]:
```
plt.scatter(data1["Price"],data1["Rating"])
```

[46]:
```
<matplotlib.collections.PathCollection at 0x7fa38a7d5ea0>
```



3.1.20 Make scatter plot/joinplot for Rating vd. Size

3.1.21 Are heavier apps rated better?

[47]:
```
data1['Size']
```

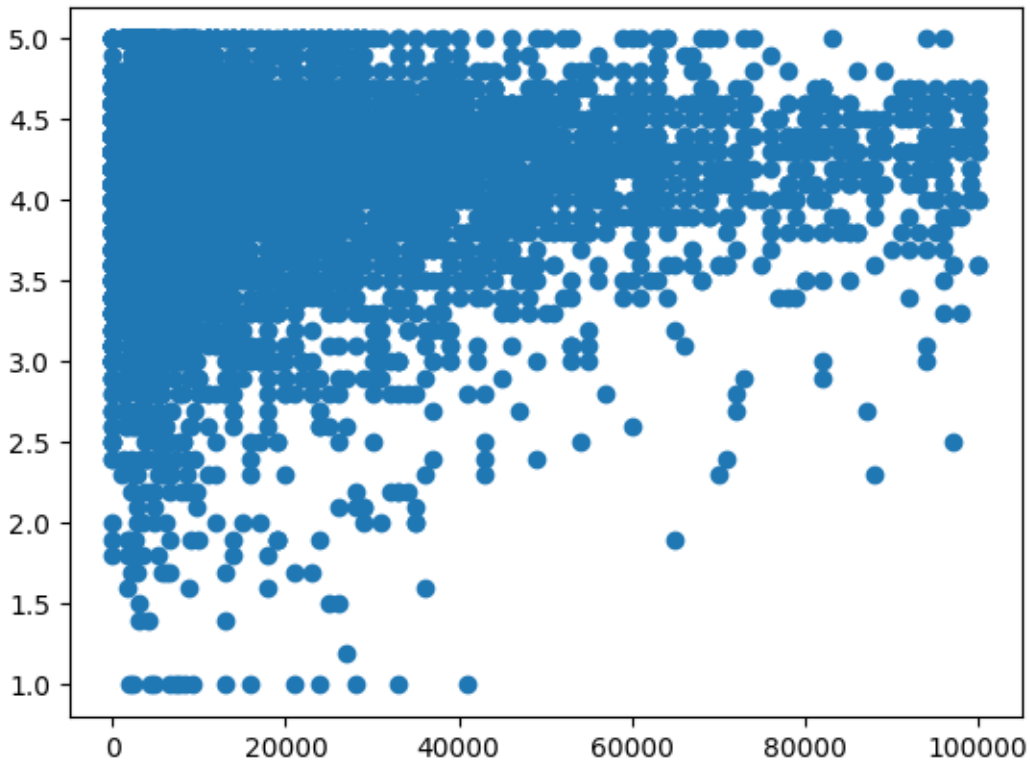
[47]:
```
0        15000.0
1        25000.0
2        19000.0
3         1800.0
4        14000.0
           …
10832        0.0
10833     2600.0
10835    53000.0
10836     3600.0
10838        0.0
Name: Size, Length: 7704, dtype: float64
```

[48]:
```
data1['Rating']
```

[48]:
```
0        4.5
1        4.5
2        4.6
3        4.4
4        2.6
        ...
10832    4.8
10833    4.0
10835    4.5
10836    5.0
10838    4.5
Name: Rating, Length: 7704, dtype: float64
```

[49]:
```
plt.scatter(data1["Size"],data1["Rating"])
```

[49]:
```
<matplotlib.collections.PathCollection at 0x7fa38aa370d0>
```

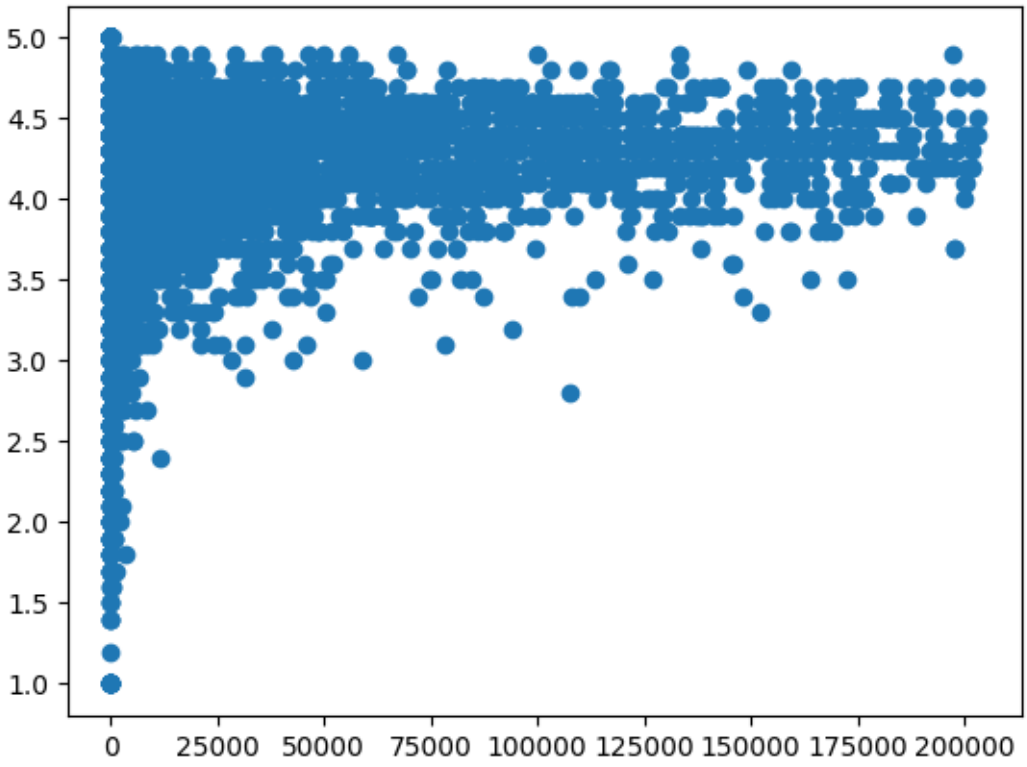


3.1.22 Make scatter plot/joinplot for Rating vs.Reviews

3.1.23 Does more review mean a better rating always?

[50]:
```
plt.scatter(data1["Reviews"],data1["Rating"])
```

[50]:
```
<matplotlib.collections.PathCollection at 0x7fa38a79b910>
```

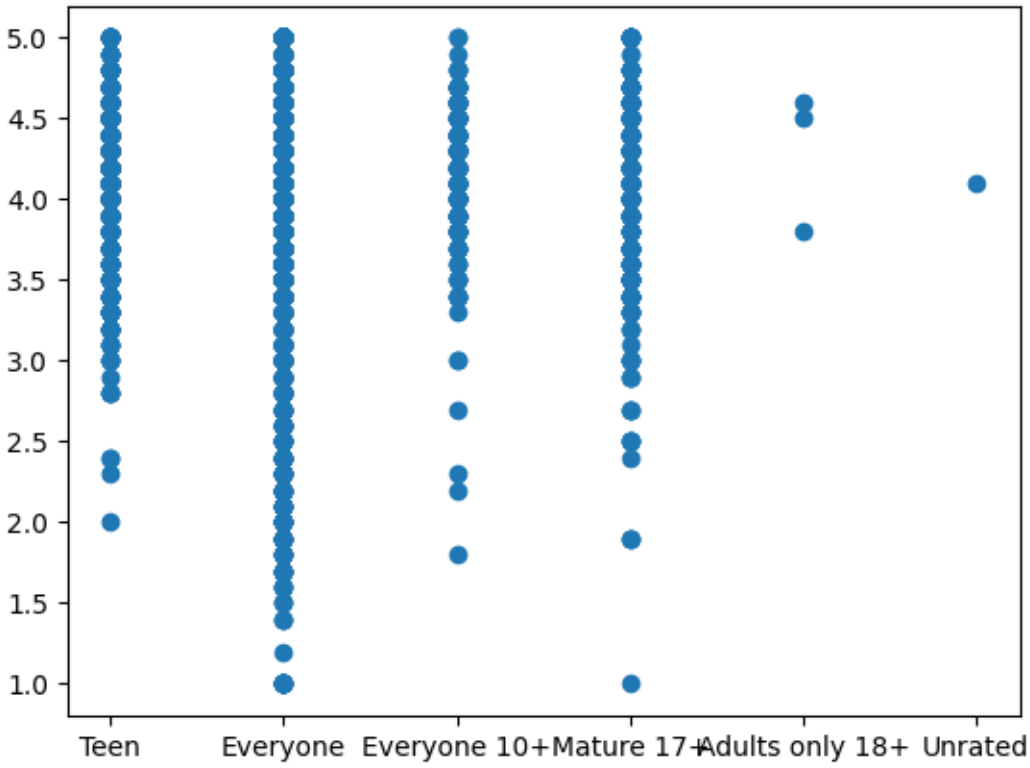


3.1.24 Make boxplot for Rating vs.Content Rating

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

[51]:
```
plt.scatter(data1["Content Rating"],data1["Rating"])
```

[51]:
```
<matplotlib.collections.PathCollection at 0x7fa38a5724a0>
```



3.1.26 Make boxplot for Ratings vs. Category

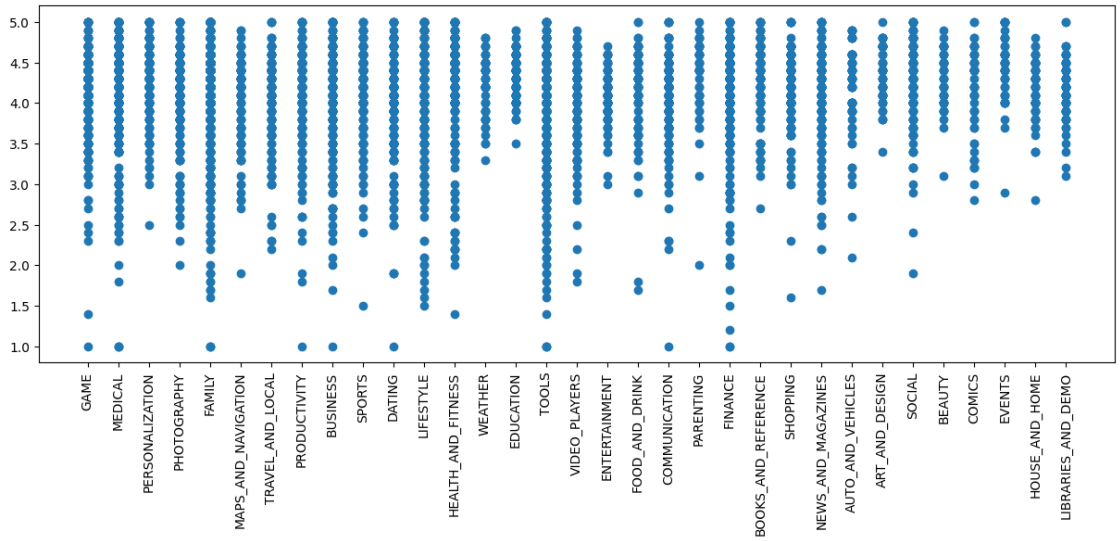
3.1.27 Which genre has the best ratings?

[52]:
```
plt.figure(figsize=(15,5))
plt.scatter(data1["Category"],data1["Rating"])
plt.xticks(rotation=90)
```

[52]:
```
([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,
 29,
 30,
 31,
 32],
[Text(0, 0, 'GAME'),
 Text(1, 0, 'MEDICAL'),
 Text(2, 0, 'PERSONALIZATION'),
 Text(3, 0, 'PHOTOGRAPHY'),
 Text(4, 0, 'FAMILY'),
 Text(5, 0, 'MAPS_AND_NAVIGATION'),
 Text(6, 0, 'TRAVEL_AND_LOCAL'),
 Text(7, 0, 'PRODUCTIVITY'),
 Text(8, 0, 'BUSINESS'),
 Text(9, 0, 'SPORTS'),
 Text(10, 0, 'DATING'),
 Text(11, 0, 'LIFESTYLE'),
 Text(12, 0, 'HEALTH_AND_FITNESS'),
 Text(13, 0, 'WEATHER'),
 Text(14, 0, 'EDUCATION'),
 Text(15, 0, 'TOOLS'),
 Text(16, 0, 'VIDEO_PLAYERS'),
 Text(17, 0, 'ENTERTAINMENT'),
 Text(18, 0, 'FOOD_AND_DRINK'),
 Text(19, 0, 'COMMUNICATION'),
 Text(20, 0, 'PARENTING'),
 Text(21, 0, 'FINANCE'),
 Text(22, 0, 'BOOKS_AND_REFERENCE'),
 Text(23, 0, 'SHOPPING'),
 Text(24, 0, 'NEWS_AND_MAGAZINES'),
 Text(25, 0, 'AUTO_AND_VEHICLES'),
 Text(26, 0, 'ART_AND_DESIGN'),
 Text(27, 0, 'SOCIAL'),
```

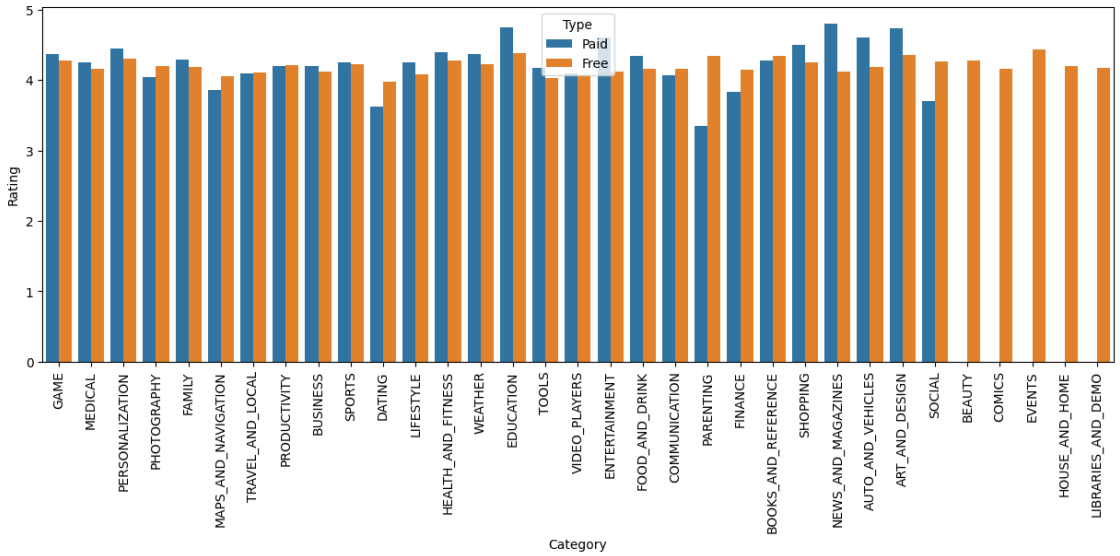
```
Text(28, 0, 'BEAUTY'),
Text(29, 0, 'COMICS'),
Text(30, 0, 'EVENTS'),
Text(31, 0, 'HOUSE_AND_HOME'),
Text(32, 0, 'LIBRARIES_AND_DEMO')])
```



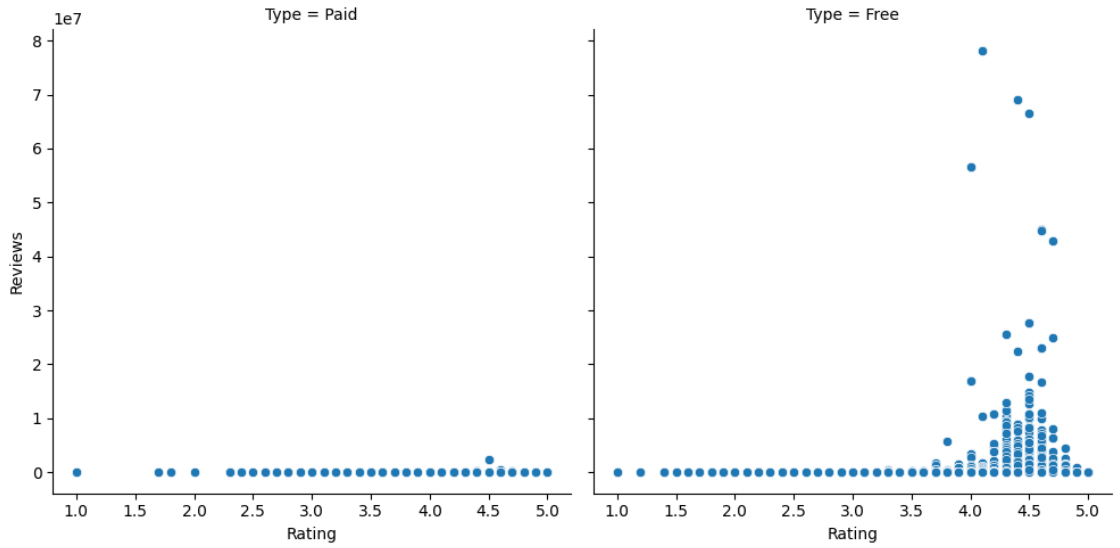
[53]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

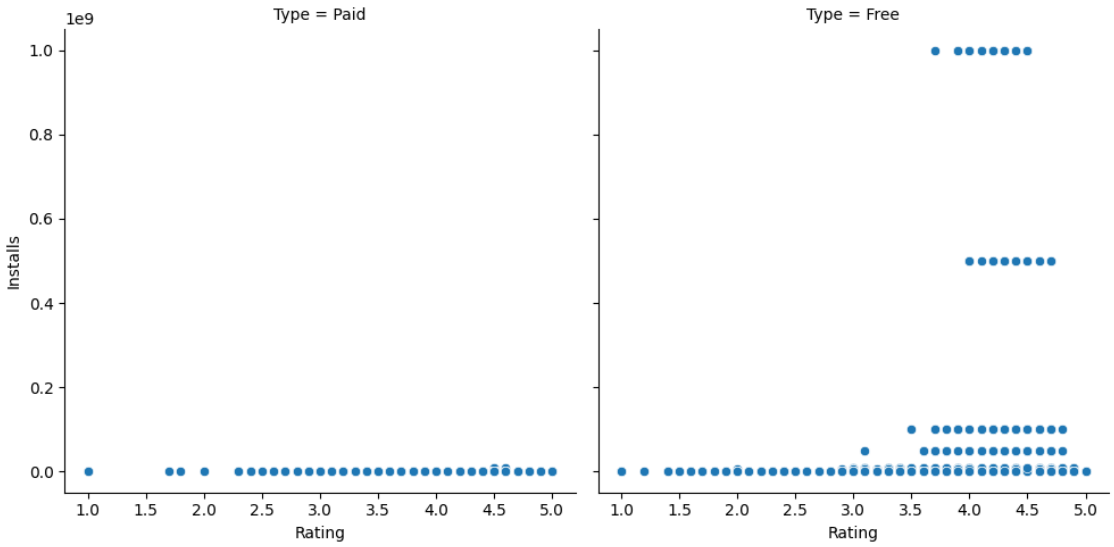
[54]:
```
plt.figure(figsize=(15,5))
sns.barplot(data = data,␣
  ↪x="Category",y="Rating",dodge=True,ci=None,estimator=np.mean,hue="Type")
plt.xticks(rotation=90)
plt.show()
```

```
/tmp/ipykernel_82/3607608562.py:2: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(data = data,
x="Category",y="Rating",dodge=True,ci=None,estimator=np.mean,hue="Type")
```



[72]:
```
sns.relplot(data=data,x="Rating",y="Reviews",col="Type",kind="scatter")
sns.relplot(data=data,x="Rating",y="Installs",col="Type",kind="scatter")
plt.show()
```





[58]:
```
inp1 = data1.copy()
```

3.1.28 log transforamtion since they are skewed

[59]:
```
import numpy as np
inp1["Reviews"]=np.log(inp1["Reviews"])
inp1["Installs"]=np.log(inp1["Installs"])
```

3.1.29 Delete unnecessary columns

[60]:
```
inp1.drop(["App","Last Updated","Current Ver","Android␣
 ↪Ver"],axis=1,inplace=True)
inp1.head()
```

[60]:

	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
0	GAME	4.5	10.550774	15000.0	11.512925	Paid	9.99	Teen	Role Playing
1	MEDICAL	4.5	4.143135	25000.0	9.210340	Paid	9.99	Everyone	Medical
2	MEDICAL	4.6	5.978886	19000.0	6.907755	Paid	9.99	Everyone	Medical
3	MEDICAL	4.4	2.833213	1800.0	6.907755	Paid	9.99	Everyone 10+	Medical
4	MEDICAL	2.6	3.713572	14000.0	6.907755	Paid	9.99	Everyone	Medical

3.1.30 create dummy variables

[61]:
```
inp2 = pd.get_dummies(data=inp1,columns=["Category","Content␣
↪Rating","Genres","Type"],drop_first=True)
```

3.1.31 segregating ind and depnd variables

[62]:
```
X = inp2.drop("Rating",axis=1)
y = inp2.loc[:,"Rating"]
from sklearn.model_selection import train_test_split
```

3.1.32 Spliting

[63]:
```
train_X,test_X,train_y,test_y = train_test_split(X,y,test_size = 0.3,␣
↪random_state =123)
```

3.1.33 using Linear regression

[64]:
```
from sklearn.linear_model import LinearRegression
line_reg = LinearRegression()
```

[65]:
```
line_reg.fit(train_X, train_y)
```

[65]:
```
LinearRegression()
```

3.1.34 R sq value on train data is 0.144

[66]:
```
print(f'r_sqr value:{line_reg.score(train_X,train_y)}')
```

```
r_sqr value:0.1419864546097953
```

3.1.35 predict

[67]:
```
y_pred =line_reg.predict(test_X)
```

[68]:
```
d=pd.DataFrame()
d["test_y"] = test_y
d["y_pred"] = y_pred
###map with sklearn
#map with formula
d["mp"]=abs((d["test_y"]-d["y_pred"])/d["test_y"])
(d.mp.mean())*100
```

[68]:
```
10.416915885814571
```

3.1.36 R sq value on test data is 0.106

[69]:
```
print(f'r_sqr value: {line_reg.score(train_X,train_y)}')
```

```
r_sqr value: 0.1419864546097953
```

3.1.37 Conclusion

The R sq is pretty low, but the MAP is quite high