In [1]:

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

In [2]:

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

In [3]:

```
df.shape
```

Out[3]:

```
(10841, 13)
```

In [4]:

```
df.head (5)
```

Out[4]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	0	Everyone
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+	Free	0	Everyone
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000,000+	Free	0	Teen
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100,000+	Free	0	Everyone

In [5]:

```
df.info ()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10841 entries, 0 to 10840
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   App             10841 non-null  object
 1   Category        10841 non-null  object
 2   Rating          9367 non-null   float64
 3   Reviews         10841 non-null  object
 4   Size            10841 non-null  object
 5   Installs        10841 non-null  object
 6   Type            10840 non-null  object
```

```
 7   Price           10841 non-null  object
 8   Content Rating  10840 non-null  object
 9   Genres          10841 non-null  object
 10  Last Updated    10841 non-null  object
 11  Current Ver     10833 non-null  object
 12  Android Ver     10838 non-null  object
dtypes: float64(1), object(12)
memory usage: 1.1+ MB
```

In [6]:

```
## Check for null values in the data. Get the number of null values for ea
df.isnull().sum ()
```

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]:

```
## 3. Drop records with nulls in any of the columns.
df = df.dropna()
```

In [8]:

```
#new shape after the drop
df.shape
```

Out[8]:

```
(9360, 13)
```

In [9]:

```
## Drop records with nulls in any of the columns check
df.isnull().sum ()
```

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]:

```
df = df[df.Size != 'Varies with device']
```

In [11]:

```
# 4.1 Size column has sizes in Kb as well as Mb. To analyze, you'll need t
# Extract the numeric value from the column
```

```
# multiply the value by 1,000, if size is mentioned in Mb

def MtoK(b):
    if b[len(b) -1: ] == 'M':
        return(float(b[0: len(b) -1 ])*1000)
    elif b[len(b) -1: ] == 'K' or  b[len(b) -1: ] == 'k':
        return(float(b[0: len(b) -1 ]))
    else:
        return b
```

In [12]:

```
df.Size = df.Size.apply(MtoK)
```

In [13]:

```
df.info ()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7723 entries, 0 to 10840
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   App             7723 non-null   object
 1   Category        7723 non-null   object
 2   Rating          7723 non-null   float64
 3   Reviews         7723 non-null   object
 4   Size            7723 non-null   float64
 5   Installs        7723 non-null   object
 6   Type            7723 non-null   object
 7   Price           7723 non-null   object
 8   Content Rating  7723 non-null   object
 9   Genres          7723 non-null   object
 10  Last Updated    7723 non-null   object
 11  Current Ver     7723 non-null   object
 12  Android Ver     7723 non-null   object
dtypes: float64(2), object(11)
memory usage: 844.7+ KB
```

In [14]:

```
df.Size
```

Out[14]:

```
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: Size, Length: 7723, dtype: float64
```

In [15]:

```
df = df[df.Size != 'Varies with device']
```

In [16]:

```
df.shape
```

Out[16]:

```
(7723, 13)
```

In [17]:
```
# 4.2 Reviews is a numeric field that is loaded as a string field. Convert
#converts to int

df ["Reviews"] = df ['Reviews'].astype ("int64")
```

In [18]:
```
df ["Reviews"].dtype
```

Out[18]:
```
dtype('int64')
```

In [19]:
```
""" 4.3 Installs field is currently stored as string and has values like 1
Treat 1,000,000+ as 1,000,000
remove '+', ',' from the field, convert it to integer"""

def remove_char(val):
    return(int(val.replace(',','').replace('+','')))
```

In [20]:
```
df.Installs = df.Installs.map (remove_char)
```

In [21]:
```
df.Installs
```

Out[21]:
```
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 [22]:
```
# 4.4 Price field is a string and has $ symbol. Remove '$' sign,
# and convert it to numeric.

def remove_symbol(val):
    return(float(val.replace("$", "")))
```

In [23]:
```
df.Price = df.Price.apply(remove_symbol)
```

In [24]:
```
df ["Price"].dtype
```

Out[24]:
```
dtype('float64')
```

In [25]:
```
df.Price
```

Out[25]:
```
0        0.0
1        0.0
2        0.0
3        0.0
4        0.0
```

```
        ...
10833    0.0
10834    0.0
10836    0.0
10837    0.0
10840    0.0
Name: Price, Length: 7723, dtype: float64
```

In [26]:

```
df.shape
```

Out[26]:

```
(7723, 13)
```

In [27]:

```
"""5.1 Sanity checks:
Average rating should be between 1 and 5 as only these values are allowed 
Drop the rows that have a value outside this range.
"""
```

Out[27]:

```
'5.1 Sanity checks:\nAverage rating should be between 1 and 5 as only these
values are allowed on the play store. \nDrop the rows that have a value out
side this range.\n'
```

In [28]:

```
# Way 1 to check
df[(df.Rating <1) | (df.Rating>5)]
```

Out[28]:

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

In [29]:

```
# Way 2 to check
df.loc[df.Rating < 1] & df.loc[df.Rating > 5]
# Result : no rating is out side the range
```

Out[29]:

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

In [30]:

```
#5.2 """Reviews should not be more than installs as only those who install
If there are any such records, drop them."""
```

In [31]:

```
df.loc[df.Reviews > df.Installs]
```

Out[31]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
2454	KBA-EZ Health Guide	MEDICAL	5.0	4	25000.0	1	Free	0.00	Everyone	Medical
5917	Ra Ga Ba	GAME	5.0	2	20000.0	1	Paid	1.49	Everyone	Arcade
6700	Brick Breaker BR	GAME	5.0	7	19000.0	5	Free	0.00	Everyone	Arcade
7402	Trovami se ci riesci	GAME	5.0	11	6100.0	10	Free	0.00	Everyone	Arcade

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
8591	DN Blog	SOCIAL	5.0	20	4200.0	10	Free	0.00	Teen	Social
10697	Mu.F.O.	GAME	5.0	2	16000.0	1	Paid	0.99	Everyone	Arcade

In [32]:

```
df.loc[df.Reviews > df.Installs].describe ()
```

Out[32]:

	Rating	Reviews	Size	Installs	Price
count	6.0	6.000000	6.000000	6.000000	6.000000
mean	5.0	7.666667	15050.000000	4.666667	0.413333
std	0.0	6.947422	8219.914841	4.412105	0.659566
min	5.0	2.000000	4200.000000	1.000000	0.000000
25%	5.0	2.500000	8575.000000	1.000000	0.000000
50%	5.0	5.500000	17500.000000	3.000000	0.000000
75%	5.0	10.000000	19750.000000	8.750000	0.742500
max	5.0	20.000000	25000.000000	10.000000	1.490000

In [33]:

```
df.loc[df.Reviews > df.Installs]
```

Out[33]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
2454	KBA-EZ Health Guide	MEDICAL	5.0	4	25000.0	1	Free	0.00	Everyone	Medical
5917	Ra Ga Ba	GAME	5.0	2	20000.0	1	Paid	1.49	Everyone	Arcade
6700	Brick Breaker BR	GAME	5.0	7	19000.0	5	Free	0.00	Everyone	Arcade
7402	Trovami se ci riesci	GAME	5.0	11	6100.0	10	Free	0.00	Everyone	Arcade
8591	DN Blog	SOCIAL	5.0	20	4200.0	10	Free	0.00	Teen	Social
10697	Mu.F.O.	GAME	5.0	2	16000.0	1	Paid	0.99	Everyone	Arcade

In [39]:

```
df.columns
```

Out[39]:

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

In [40]:

```
df[['Reviews','Installs']]
```

Out[40]:

	Reviews	Installs
0	159	10000
1	967	500000
2	87510	5000000
3	215644	50000000
4	967	100000
...	...	...
10833	44	1000
10834	7	500
10836	38	5000
10837	4	100
10840	398307	10000000

7723 rows × 2 columns

In [41]:

```
# Created a column to easily identify results for the syntax
df['RAI'] = np.where((df['Reviews'] <= df['Installs']), df['Installs'], np
```

In [42]:

```
df['RAI'].shape
```

Out[42]:

```
(7723,)
```

In [43]:

```
df['RAI'].describe()
```

Out[43]:

```
count    7.717000e+03
mean     8.430620e+06
std      5.017636e+07
min      5.000000e+00
25%      1.000000e+04
50%      1.000000e+05
75%      1.000000e+06
max      1.000000e+09
Name: RAI, dtype: float64
```

In [44]:

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

In [46]:

```
# Still have 14 columns and need to drop RAI
df.columns
```

Out[46]:

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

In [47]:

```
# to drop RAI column
df = df.drop(['RAI'], axis = 1)
```

In [48]:

```
df.shape
```

Out[48]:

```
(7717, 13)
```

In [49]:

```
df.columns
```

Out[49]:

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

In [ ]:

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

In [50]:

```
# To determine the rows with Price above $0
df.loc[df.Price > 0]
```

Out[50]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	C
234	TurboScan: scan documents and receipts in PDF	BUSINESS	4.7	11442	6800.0	100000	Paid	4.99	E
235	Tiny Scanner Pro: PDF Doc Scan	BUSINESS	4.8	10295	39000.0	100000	Paid	4.99	E
290	TurboScan: scan documents and receipts in PDF	BUSINESS	4.7	11442	6800.0	100000	Paid	4.99	E
291	Tiny Scanner Pro: PDF Doc Scan	BUSINESS	4.8	10295	39000.0	100000	Paid	4.99	E
477	Calculator	DATING	2.6	57	6200.0	1000	Paid	6.99	E
...	...	...	...	...	...	...	...	...	
10682	Fruit Ninja Classic	GAME	4.3	85468	36000.0	1000000	Paid	0.99	E
10690	FO Bixby	PERSONALIZATION	5.0	5	861.0	100	Paid	0.99	E
10760	Fast Tract Diet	HEALTH_AND_FITNESS	4.4	35	2400.0	1000	Paid	7.99	E
10782	Trine 2: Complete Story	GAME	3.8	252	11000.0	10000	Paid	16.99	
10785	sugar, sugar	FAMILY	4.2	1405	9500.0	10000	Paid	1.20	E

575 rows × 13 columns

In [51]:

```
# check to confirm any free app with Price > 0
df[np.logical_and(df['Type'] == 'Free', df['Price'] > 0)]
```

Out[51]:

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

In [ ]:

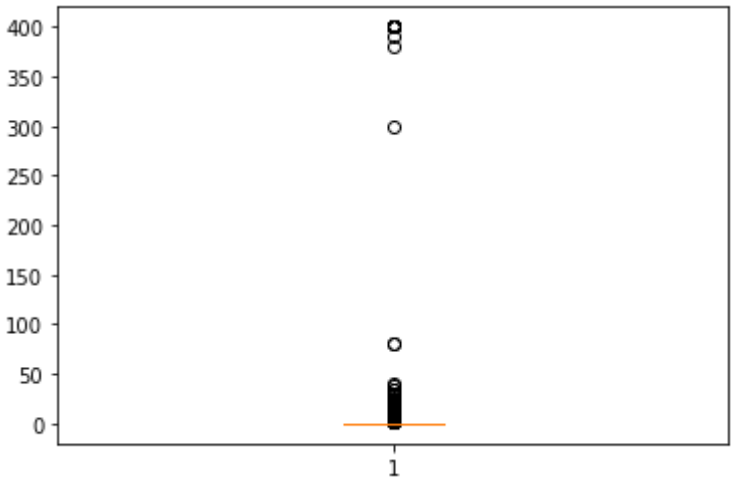
```
#Performing univariate analysis:
#Boxplot for Price
#Are there any outliers? Think about the price of usual apps on Play Store
```

In [53]:

```
plt.boxplot(df['Price'])
# most app prices are less that $100
# some outlier prices above $200
```

Out[53]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1988af23850>,
  <matplotlib.lines.Line2D at 0x1988af23be0>],
 'caps': [<matplotlib.lines.Line2D at 0x1988af23f70>,
  <matplotlib.lines.Line2D at 0x1988af31340>],
 'boxes': [<matplotlib.lines.Line2D at 0x1988af23520>],
 'medians': [<matplotlib.lines.Line2D at 0x1988af316d0>],
 'fliers': [<matplotlib.lines.Line2D at 0x1988af31a60>],
 'means': []}
```



In [54]:

```
df['Price'].describe()
```

Out[54]:

```
count    7717.000000
mean        1.128725
std        17.414784
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max       400.000000
Name: Price, dtype: float64
```

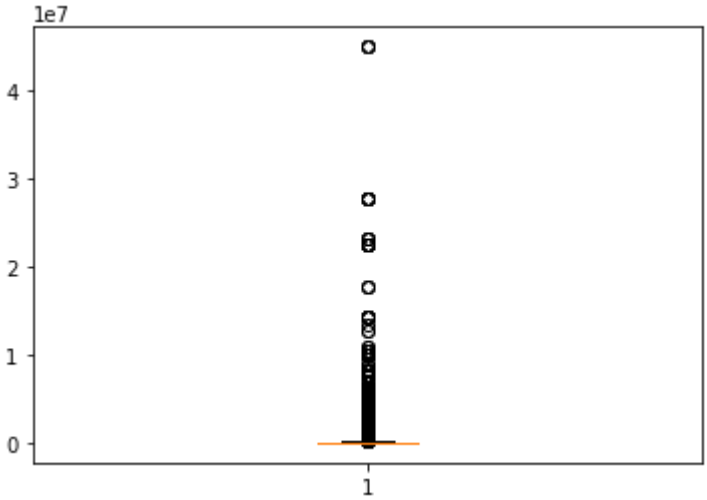
In [55]:

```
#Boxplot for Reviews
#Are there any apps with very high number of reviews? Do the values seem r

plt.boxplot(df['Reviews'])
```

Out[55]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1988af94ca0>,
  <matplotlib.lines.Line2D at 0x1988afa4070>],
```

```
 'caps': [<matplotlib.lines.Line2D at 0x1988afa4400>,
  <matplotlib.lines.Line2D at 0x1988afa4790>],
 'boxes': [<matplotlib.lines.Line2D at 0x1988af94970>],
 'medians': [<matplotlib.lines.Line2D at 0x1988afa4b20>],
 'fliers': [<matplotlib.lines.Line2D at 0x1988afa4ee0>],
 'means': []}
```



In [56]:

```
#Histogram for Rating
#How are the ratings distributed? Is it more toward higher ratings?

df['Reviews'].describe()
```

Out[56]:

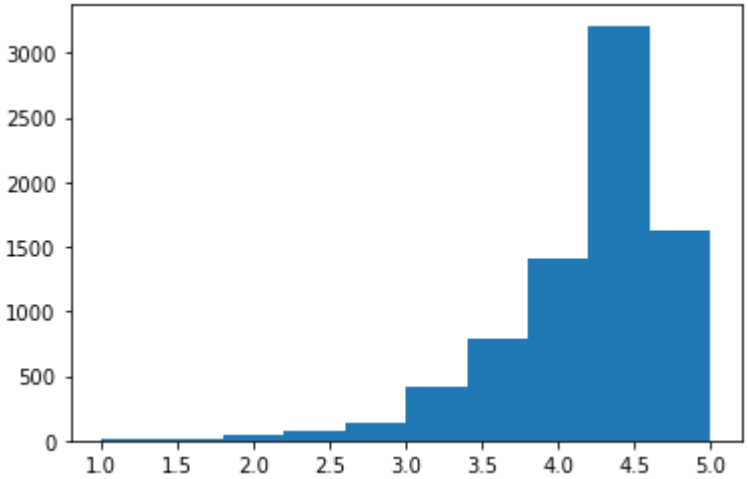
```
count    7.717000e+03
mean     2.951275e+05
std      1.864640e+06
min      1.000000e+00
25%      1.090000e+02
50%      2.351000e+03
75%      3.910900e+04
max      4.489389e+07
Name: Reviews, dtype: float64
```

In [57]:

```
plt.hist(df['Rating'])
```

Out[57]:

```
(array([  17.,   18.,   39.,   72.,  132.,  408.,  781., 1406., 3212.,
        1632.]),
 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>)
```

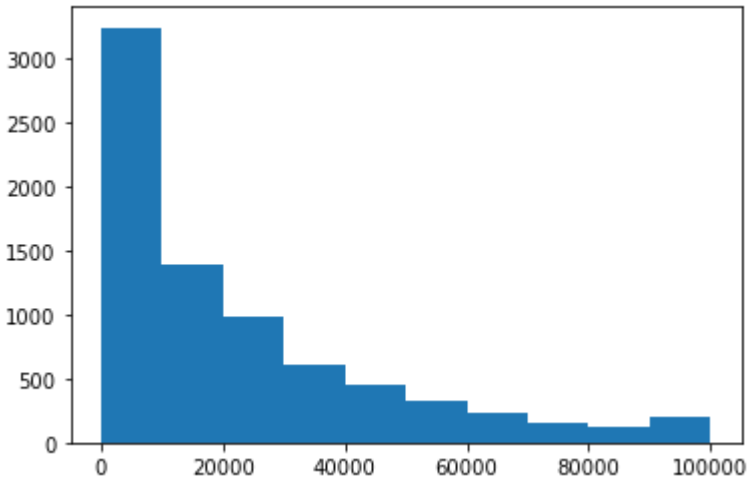


In [58]:

```
plt.hist(df['Size'])
```

Out[58]:

```
(array([3245., 1398.,  991.,  606.,  449.,  325.,  226.,  161.,  117.,
        199.]),
 array([8.500000e+00, 1.000765e+04, 2.000680e+04, 3.000595e+04,
        4.000510e+04, 5.000425e+04, 6.000340e+04, 7.000255e+04,
        8.000170e+04, 9.000085e+04, 1.000000e+05]),
 <BarContainer object of 10 artists>)
```



In [59]:

```
# To determine the rows with Price above $100
# for my DF, I assume any app price greater than $100 is too high and shou
df.loc[df.Price > 100]
```

Out[59]:

	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	Er
4362	💎 I'm rich	LIFESTYLE	3.8	718	26000.0	10000	Paid	399.99	Everyone	
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	Er
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	Er

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	
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	Er
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 [60]:

```
# Second check and same result
df[df.Price>100]
```

Out[60]:

	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	Er
4362	💎 I'm rich	LIFESTYLE	3.8	718	26000.0	10000	Paid	399.99	Everyone	
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	Er
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	Er
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	Er
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 [61]:
```
df[df.Price>100].shape
# 15 rows with outlier prices
```

Out[61]:
```
(15, 13)
```

In [62]:
```
#df shape before the drop
df.shape
```

Out[62]:
```
(7717, 13)
```

In [63]:
```
#df shape after the drop
df[df.Price <=100].shape
```

Out[63]:
```
(7702, 13)
```

In [64]:
```
df = df[df.Price <=100]
```

In [65]:
```
#new df after the dropp
df.shape
```

Out[65]:
```
(7702, 13)
```

In [66]:
```
# df accommodates all the required conditions on Price, Rating Reviews and
df.describe()
```

Out[66]:

	Rating	Reviews	Size	Installs	Price
count	7702.000000	7.702000e+03	7702.000000	7.702000e+03	7702.000000
mean	4.173890	2.957011e+05	23004.020709	8.447011e+06	0.368802
std	0.544481	1.866409e+06	23466.178824	5.022383e+07	2.348127
min	1.000000	1.000000e+00	8.500000	5.000000e+00	0.000000
25%	4.000000	1.090000e+02	5300.000000	1.000000e+04	0.000000
50%	4.300000	2.374500e+03	14000.000000	1.000000e+05	0.000000
75%	4.500000	3.949125e+04	33000.000000	1.000000e+06	0.000000
max	5.000000	4.489389e+07	100000.000000	1.000000e+09	79.990000

In [ ]:
```
# 6.2 Reviews: Very few apps have very high number of reviews.
#These are all star apps that don't help with the analysis and, in fact, w
#Drop records having more than 2 million reviews.
```

In [67]:
```
df['Reviews'].describe()
```

Out[67]:
```
count    7.702000e+03
mean     2.957011e+05
std      1.866409e+06
min      1.000000e+00
25%      1.090000e+02
```

```
50%      2.374500e+03
75%      3.949125e+04
max      4.489389e+07
Name: Reviews, dtype: float64
```

In [68]:

```
# rows with Reviews more that 2million
df.loc[df.Reviews > 2000000]
```

Out[68]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Con Ra
345	Yahoo Mail – Stay Organized	COMMUNICATION	4.3	4187998	16000.0	100000000	Free	0.0	Every
347	imo free video calls and chat	COMMUNICATION	4.3	4785892	11000.0	500000000	Free	0.0	Every
366	UC Browser Mini -Tiny Fast Private & Secure	COMMUNICATION	4.4	3648120	3300.0	100000000	Free	0.0	T
378	UC Browser - Fast Download Private & Secure	COMMUNICATION	4.5	17712922	40000.0	500000000	Free	0.0	T
383	imo free video calls and chat	COMMUNICATION	4.3	4785988	11000.0	500000000	Free	0.0	Every
...	...	...	...	...	...	...	...	...	
9142	Need for Speed™ No Limits	GAME	4.4	3344300	22000.0	50000000	Free	0.0	Every
9166	Modern Combat 5: eSports FPS	GAME	4.3	2903386	58000.0	100000000	Free	0.0	Ma
10186	Farm Heroes Saga	FAMILY	4.4	7615646	71000.0	100000000	Free	0.0	Every
10190	Fallout Shelter	FAMILY	4.6	2721923	25000.0	10000000	Free	0.0	T
10327	Garena Free Fire	GAME	4.5	5534114	53000.0	100000000	Free	0.0	T

219 rows × 13 columns

In [69]:

```
# second check
df[df.Reviews > 2000000]
```

Out[69]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Con Ra

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Con Ra
345	Yahoo Mail – Stay Organized	COMMUNICATION	4.3	4187998	16000.0	100000000	Free	0.0	Every
347	imo free video calls and chat	COMMUNICATION	4.3	4785892	11000.0	500000000	Free	0.0	Every
366	UC Browser Mini -Tiny Fast Private & Secure	COMMUNICATION	4.4	3648120	3300.0	100000000	Free	0.0	T
378	UC Browser - Fast Download Private & Secure	COMMUNICATION	4.5	17712922	40000.0	500000000	Free	0.0	T
383	imo free video calls and chat	COMMUNICATION	4.3	4785988	11000.0	500000000	Free	0.0	Every
...	...	...	...	...	...	...	...	...	
9142	Need for Speed™ No Limits	GAME	4.4	3344300	22000.0	50000000	Free	0.0	Every
9166	Modern Combat 5: eSports FPS	GAME	4.3	2903386	58000.0	100000000	Free	0.0	Ma
10186	Farm Heroes Saga	FAMILY	4.4	7615646	71000.0	100000000	Free	0.0	Every
10190	Fallout Shelter	FAMILY	4.6	2721923	25000.0	10000000	Free	0.0	T
10327	Garena Free Fire	GAME	4.5	5534114	53000.0	100000000	Free	0.0	T

219 rows × 13 columns

In [70]:

```
# 219 rows to be dropped
df[df.Reviews > 2000000].shape
```

Out[70]:

```
(219, 13)
```

In [71]:

```
df.shape
```

Out[71]:

```
(7702, 13)
```

In [72]:

```
# new df
df = df[df.Reviews <= 2000000]
```

In [73]:
```
df.shape
```

Out[73]:
```
(7483, 13)
```

In [74]:
```
df.describe()
```

Out[74]:

	Rating	Reviews	Size	Installs	Price
count	7483.000000	7.483000e+03	7483.000000	7.483000e+03	7483.000000
mean	4.165789	7.260651e+04	22027.284177	3.947465e+06	0.379595
std	0.549946	2.123720e+05	22582.977041	2.781831e+07	2.381384
min	1.000000	1.000000e+00	8.500000	5.000000e+00	0.000000
25%	4.000000	9.900000e+01	5100.000000	1.000000e+04	0.000000
50%	4.300000	2.026000e+03	14000.000000	1.000000e+05	0.000000
75%	4.500000	3.238600e+04	31000.000000	1.000000e+06	0.000000
max	5.000000	1.986068e+06	100000.000000	1.000000e+09	79.990000

In [ ]:
```
# Installs:  There seems to be some outliers in this field too.
#Apps having very high number of installs should be dropped from the analy
#Find out the different percentiles - 10, 25, 50, 70, 90, 95, 99
#Decide a threshold as cutoff for outlier and drop records having values m
```

In [75]:
```
df['Installs'].describe()
```

Out[75]:
```
count    7.483000e+03
mean     3.947465e+06
std      2.781831e+07
min      5.000000e+00
25%      1.000000e+04
50%      1.000000e+05
75%      1.000000e+06
max      1.000000e+09
Name: Installs, dtype: float64
```

In [76]:
```
np.arange(0,1,0.05)
```

Out[76]:
```
array([0.  , 0.05, 0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 ,
       0.55, 0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95])
```

In [77]:
```
# To get the percentiles by 0.05
df['Installs'].quantile(q = np.arange(0,1,0.05))
```

Out[77]:
```
0.00           5.0
0.05         100.0
0.10        1000.0
0.15        1000.0
0.20        5000.0
0.25       10000.0
0.30       10000.0
0.35       10000.0
0.40       50000.0
0.45      100000.0
0.50      100000.0
```

```
0.55      100000.0
0.60      500000.0
0.65     1000000.0
0.70     1000000.0
0.75     1000000.0
0.80     5000000.0
0.85     5000000.0
0.90    10000000.0
0.95    10000000.0
Name: Installs, dtype: float64
```

In [78]:

```
df['Installs'].quantile(0.99)
```

Out[78]:

```
50000000.0
```

In [79]:

```
df.shape
```

Out[79]:

```
(7483, 13)
```

In [80]:

```
# Installations above 10,000,000 are outliers and should be dropped
df[df.Installs > 10000000]
```

Out[80]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Cor Ra
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25000.0	50000000	Free	0.0	
194	OfficeSuite : Free Office + PDF Editor	BUSINESS	4.3	1002861	35000.0	100000000	Free	0.0	Ever
225	Secure Folder	BUSINESS	3.8	14760	8600.0	50000000	Free	0.0	Ever
293	OfficeSuite : Free Office + PDF Editor	BUSINESS	4.3	1002859	35000.0	100000000	Free	0.0	Ever
346	imo beta free calls and text	COMMUNICATION	4.3	659395	11000.0	100000000	Free	0.0	Ever
...	...	...	...	...	...	...	...	...	
10378	BMX Boy	GAME	4.2	839206	12000.0	50000000	Free	0.0	Ever
10408	Shoot Hunter- Gun Killer	GAME	4.3	320334	27000.0	50000000	Free	0.0	
10429	Talking Tom Bubble Shooter	FAMILY	4.4	687136	54000.0	50000000	Free	0.0	Ever
10513	Flight Simulator: Fly Plane 3D	FAMILY	4.0	660613	21000.0	50000000	Free	0.0	Ever

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Cor Ra
10549	Toy Truck Rally 3D	GAME	4.0	301895	25000.0	50000000	Free	0.0	Ever

176 rows × 13 columns

In [81]:

```
# second check
df.loc[df.Installs > 10000000]
```

Out[81]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Cor Ra
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25000.0	50000000	Free	0.0	
194	OfficeSuite : Free Office + PDF Editor	BUSINESS	4.3	1002861	35000.0	100000000	Free	0.0	Ever
225	Secure Folder	BUSINESS	3.8	14760	8600.0	50000000	Free	0.0	Ever
293	OfficeSuite : Free Office + PDF Editor	BUSINESS	4.3	1002859	35000.0	100000000	Free	0.0	Ever
346	imo beta free calls and text	COMMUNICATION	4.3	659395	11000.0	100000000	Free	0.0	Ever
...	...	...	...	...	...	...	...	...	
10378	BMX Boy	GAME	4.2	839206	12000.0	50000000	Free	0.0	Ever
10408	Shoot Hunter-Gun Killer	GAME	4.3	320334	27000.0	50000000	Free	0.0	
10429	Talking Tom Bubble Shooter	FAMILY	4.4	687136	54000.0	50000000	Free	0.0	Ever
10513	Flight Simulator: Fly Plane 3D	FAMILY	4.0	660613	21000.0	50000000	Free	0.0	Ever
10549	Toy Truck Rally 3D	GAME	4.0	301895	25000.0	50000000	Free	0.0	Ever

176 rows × 13 columns

In [82]:

```
#new df without outlier
df[df.Installs <= 10000000].shape
```

Out[82]:

```
(7307, 13)
```

In [83]:

```
df = df[df.Installs <= 10000000]
```

In [84]:

```
df.describe()
```

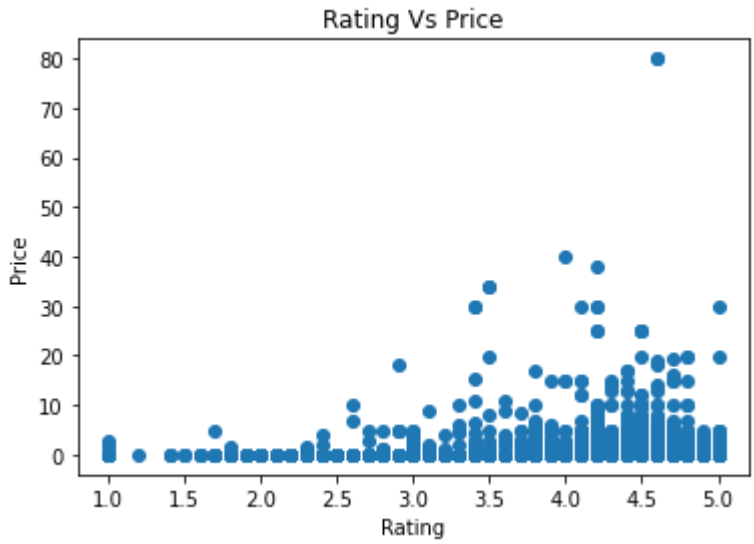
Out[84]:

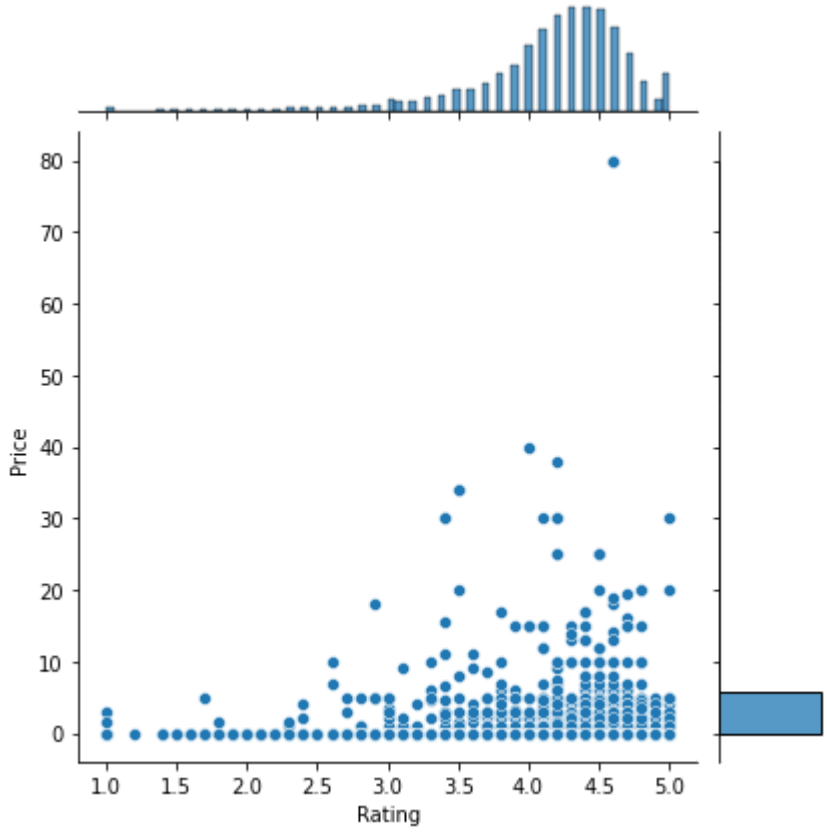
	Rating	Reviews	Size	Installs	Price
count	7307.000000	7.307000e+03	7307.000000	7.307000e+03	7307.000000
mean	4.162899	5.091109e+04	21687.801765	1.716009e+06	0.388738
std	0.555276	1.457407e+05	22460.971012	3.205978e+06	2.409159
min	1.000000	1.000000e+00	8.500000	5.000000e+00	0.000000
25%	4.000000	9.100000e+01	5000.000000	1.000000e+04	0.000000
50%	4.300000	1.749000e+03	14000.000000	1.000000e+05	0.000000
75%	4.500000	2.755850e+04	30000.000000	1.000000e+06	0.000000
max	5.000000	1.736105e+06	100000.000000	1.000000e+07	79.990000

In [85]:

```
# 7.1 Make scatter plot/joinplot for Rating vs. Price
# What pattern do you observe? Does rating increase with price?

#plt.scatter(x,y)

#x --> Rating
#y --> Price

plt.scatter(df['Rating'], df['Price'])
plt.xlabel('Rating')
plt.ylabel('Price')
plt.title('Rating Vs Price')
sns.jointplot(df['Rating'], df['Price'])
print('form the plot below, rating does not increase with price')
```

```
C:\Users\Nitin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Futur
eWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpr
etation.
  warnings.warn(
form the plot below, rating does not increase with price
```

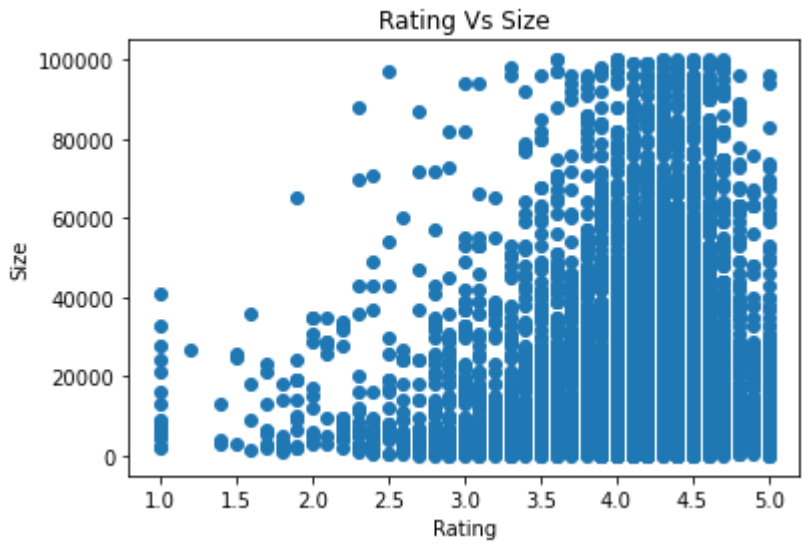


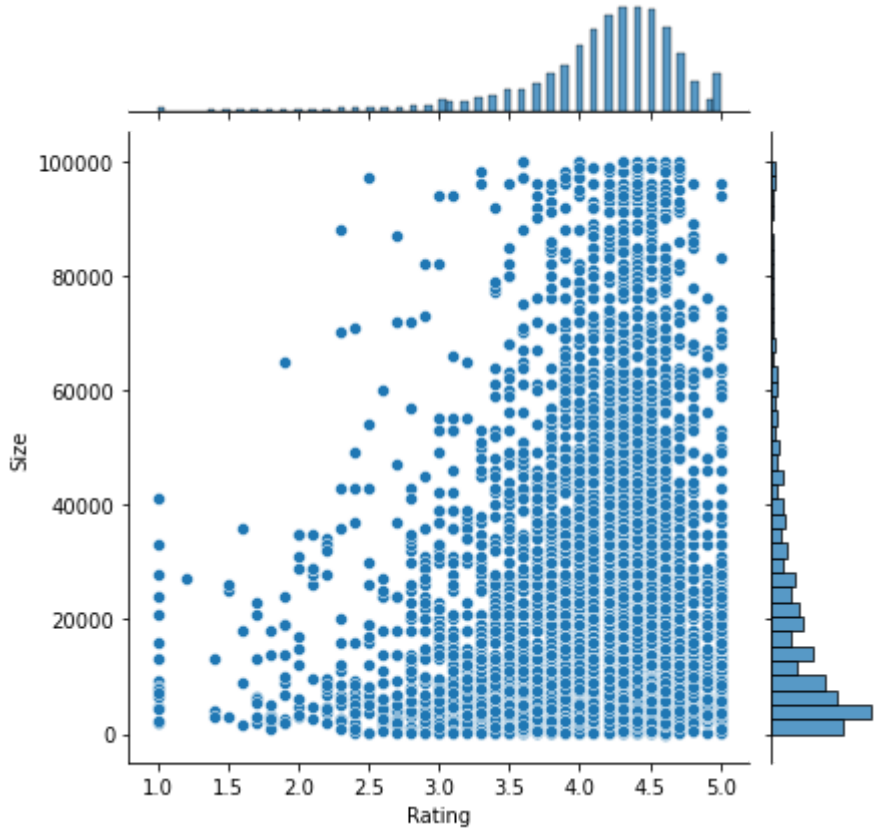


In [87]:

```
#7.2 Make scatter plot/joinplot for Rating vs. Size
# Are heavier apps rated better?

plt.scatter(df['Rating'], df['Size'])
plt.xlabel('Rating')
plt.ylabel('Size')
plt.title('Rating Vs Size')
sns.jointplot(df['Rating'], df['Size'])
print('from the plot, lighter apps have less ratings than the heavier apps
```

```
C:\Users\Nitin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Futur
eWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpr
etation.
  warnings.warn(
from the plot, lighter apps have less ratings than the heavier apps and are
likely to be rated lower
```



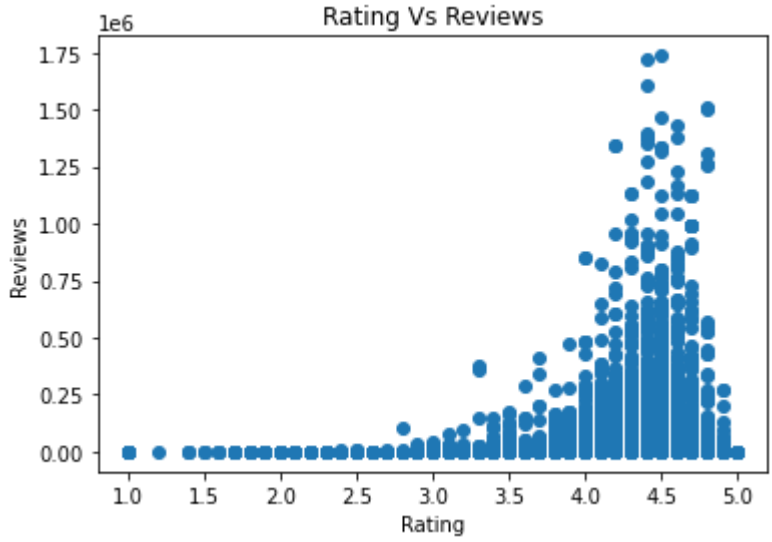


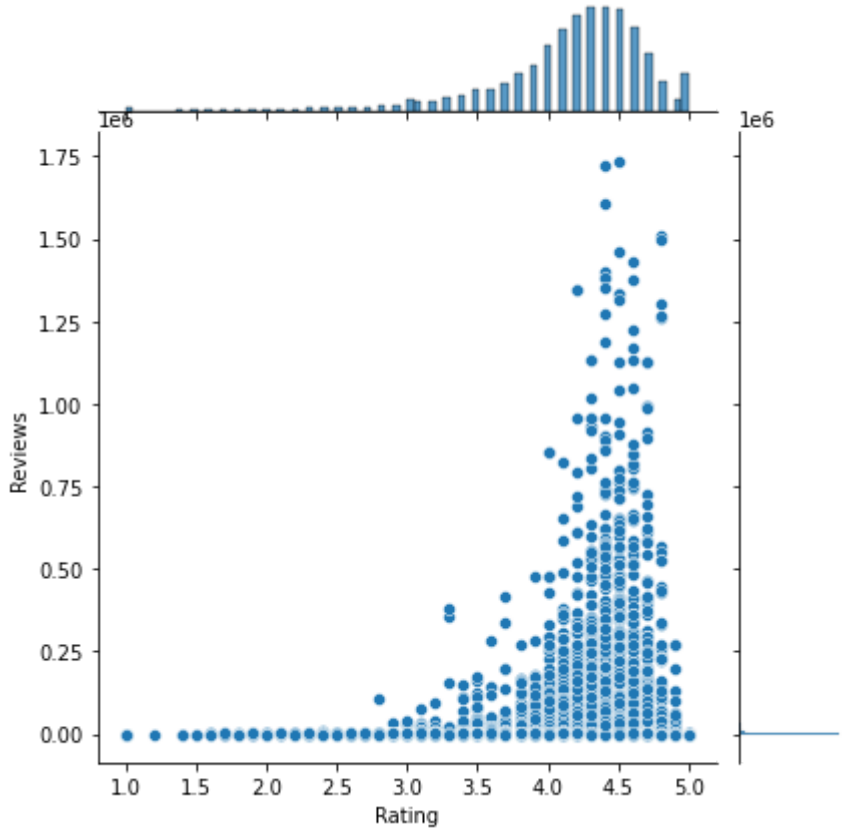
In [89]:

```
# 7.3 Make scatter plot/joinplot for Rating vs. Reviews
# Does more review mean a better rating always?

#plt.scatter(x,y)

#x --> Rating
#y --> Reviews

plt.scatter(df['Rating'], df['Reviews'])
plt.xlabel('Rating')
plt.ylabel('Reviews')
plt.title('Rating Vs Reviews')
sns.jointplot(df['Rating'], df['Reviews'])
print('from the plot, apps with the most reviews are rated highly')
```

```
C:\Users\Nitin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Futur
eWarning: Pass the following variables as keyword args: x, y. From version
```

```
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpr
etation.
  warnings.warn(
from the plot, apps with the most reviews are rated highly
```



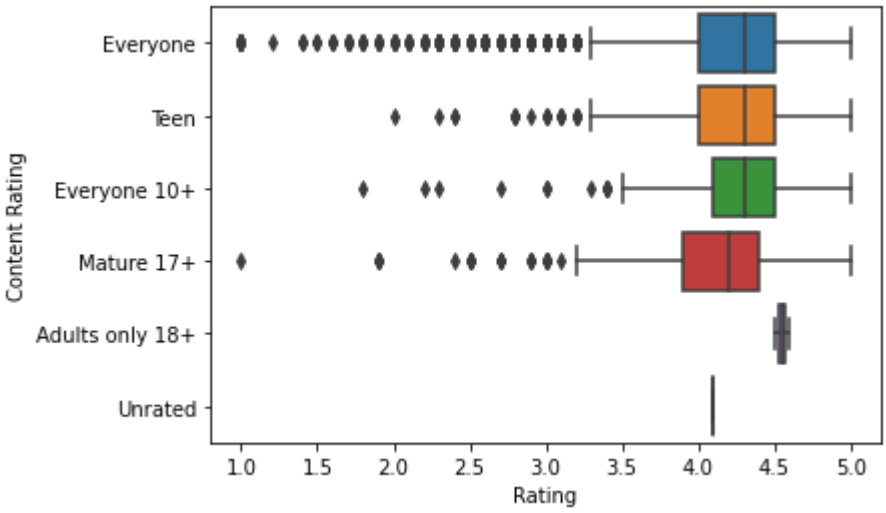


In [90]:

```
# 7.4 Make boxplot for Rating vs. Content Rating
# Is there any difference in the ratings? Are some types liked better?

sns.boxplot(df['Rating'], df['Content Rating'])
print('Apps for Teens  Content Rating are generally rated higher than othe
```

```
C:\Users\Nitin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Futur
eWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpr
etation.
  warnings.warn(
```

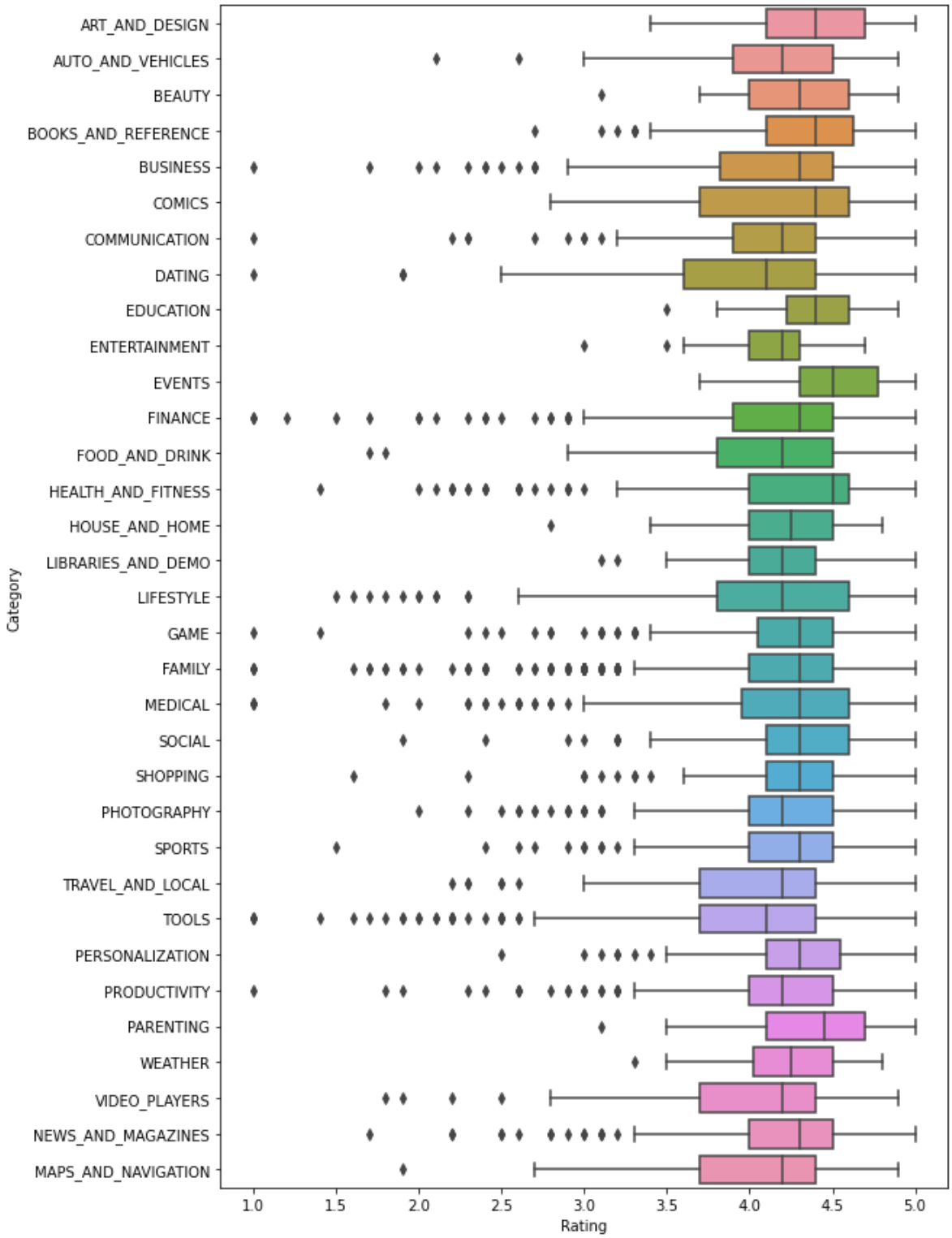
```
Apps for Teens are generally rated higher than others, while the apps for E
veryone show a large variance in rating
```



In [93]:

```
#Make boxplot for Ratings vs. Category
# Which genre has the best ratings?


fig, axis = plt.subplots(figsize=(9, 15))
sns.boxplot(df['Rating'], df['Category'])
print('Apps for parenting and events show the highest ratings')
```

```
C:\Users\Nitin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Futur
eWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterpr
etation.
  warnings.warn(
Apps for parenting and events show the highest ratings
```



In [96]:

```
# 8 For the steps below, create a copy of the dataframe to make all the ed

inp1 = df.copy().reset_index()
```

In [109…

```
# 8.1  Reviews and Install have some values that are still relatively very
# Before building a linear regression model, you need to reduce the skew.
# Apply log transformation (np.log1p) to Reviews and Installs.

inp1['Reviews'] = np.log1p(inp1['Reviews'])
inp1['Installs'] = np.log1p(inp1['Installs'])
inp1['Size'] = np.log1p(inp1['Size'])
```

In [100…

```
# 8.2 Drop columns App, Last Updated, Current Ver, and Android Ver. These

inp1.drop(columns = ['index', 'App', 'Last Updated', 'Current Ver', 'Andro
```

In [101…

```
inp1.columns
```

Out[101…

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

In [102…

```
inp1.head()
```

Out[102…

	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Gen
0	ART_AND_DESIGN	4.1	159	19000.0	10000	Free	0.0	Everyone	Art & Des
1	ART_AND_DESIGN	3.9	967	14000.0	500000	Free	0.0	Everyone	A Design;Prete P
2	ART_AND_DESIGN	4.7	87510	8700.0	5000000	Free	0.0	Everyone	Art & Des
3	ART_AND_DESIGN	4.3	967	2800.0	100000	Free	0.0	Everyone	A Design;Creati
4	ART_AND_DESIGN	4.4	167	5600.0	50000	Free	0.0	Everyone	Art & Des

In [103…

```
inp1.shape
```

Out[103…

```
(7307, 9)
```

In [ ]:

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

In [104…

```
categorical_cols = ['Category', 'Genres', 'Content Rating', 'Type']

inp2 = pd.get_dummies(inp1, columns=categorical_cols, drop_first=True)
```

In [105…

```
inp2.head()
```

Out[105…

	Rating	Reviews	Size	Installs	Price	Category_AUTO_AND_VEHICLES	Category_BEAU
0	4.1	159	19000.0	10000	0.0	0	
1	3.9	967	14000.0	500000	0.0	0	
2	4.7	87510	8700.0	5000000	0.0	0	
3	4.3	967	2800.0	100000	0.0	0	
4	4.4	167	5600.0	50000	0.0	0	

5 rows × 154 columns

In [106…

```
# 9 Train test split  and apply 70-30 split. Name the new dataframes df_t
```

In [113…

```
from sklearn.model_selection import train_test_split
df_train, df_test = train_test_split(inp2, train_size = 0.7, random_state =
```

In [114…

```
df_train.shape, df_test.shape
```

Out[114…

```
((5114, 154), (2193, 154))
```

In [115…

```
# 10. Separate the dataframes into X_train, y_train, X_test, and y_test.
y_train = df_train.Rating
X_train = df_train.drop(['Rating'], axis=1)

y_test = df_test.Rating
X_test = df_test.drop(['Rating'], axis=1)
```

In [116…

```
# 11.1 Model building
# Use linear regression as the technique

from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
```

Out[116…

```
LinearRegression()
```

In [118…

```
# 11.2
# Report the R2 on the train set
from sklearn.metrics import r2_score
y_train_pred= lr.predict(X_train)
r2_score(y_train, y_train_pred)
```

Out[118…

```
0.06861486297278863
```

In [ ]:

```
# 12 Make predictions on test set and report R2.
```

In [119…

```
y_test_pred= lr.predict(X_test)
r2_score(y_test, y_test_pred)
```

Out[119…

```
0.05096091664816793
```