In [113…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

#importing all neccessary libraries
```

In [2]:
```
df= pd.read_csv(r"C:\Users\rahul\Desktop\Study Materials\Data Analytics\Python\Project\g
#1)Loaded the data file using pandas
```

In [3]:
```
df.describe(include='all')
```

Out[3]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated
count	10841	10841	9367.000000	10841	10841	10841	10840	10841	10840	10841	10841
unique	9660	34	NaN	6002	462	22	3	93	6	120	1378
top	ROBLOX	FAMILY	NaN	0	Varies with device	1,000,000+	Free	0	Everyone	Tools	August 3, 2018
freq	9	1972	NaN	596	1695	1579	10039	10040	8714	842	326
mean	NaN	NaN	4.193338	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	NaN	NaN	0.537431	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	NaN	NaN	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	NaN	NaN	4.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	NaN	NaN	4.300000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	NaN	NaN	4.500000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	NaN	NaN	19.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [4]:
```
df.drop_duplicates(inplace=True)
#removed all the duplicates as part of data cleaning to increase accuracy
```

In [5]:
```
df.isna().sum()
#2)Checking for null values in the data and displaying null values for each column.
```

Out[5]:
```
App                0
Category           0
Rating          1465
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 [6]:
```
df.dropna(subset = ['Content Rating','Type','Android Ver','Current Ver'], axis = 0, inpl
#3)Droping records with nulls from columns - 'Content Rating','Type','Android Ver','Curr
df["Rating"].fillna(df["Rating"].mode()[0],axis=0,inplace = True)
#Replaced nulls from Rating column with "mode of rating" which is 4.4
```

In [7]:
```
df["Size-Numeric"] = df["Size"].str.extract("(\d+\.?\d*)").astype("float")
df.loc[df["Size"].str.contains("M"),"Size-Numeric"] *=1000
# 4.1 Extracted the numeric value from the column using expression pattern, extract meth
```

In [8]:
```
df.drop("Size",axis =1,inplace = True)
#dropping the Size column
```

In [9]:
```
df.rename(columns={"Size-Numeric":"Size"}, inplace = True)
#renaming the new column to original
```

In [10]:
```
df.isnull().sum()
#the Size column generated 1525 null values which were orignally as string "varies with
```

Out[10]:
```
App                0
Category           0
Rating             0
Reviews            0
Installs           0
Type               0
Price              0
Content Rating     0
Genres             0
Last Updated       0
Current Ver        0
Android Ver        0
Size            1525
dtype: int64
```

In [11]:
```
df["Size"].fillna(df["Size"].mode()[0], axis =0, inplace = True)
#Replace null with mode of strings which is 11000.0 kb
```

In [12]:
```
df["Reviews"] = df["Reviews"].astype("int")
#4.2) Reviews is a numeric field that is loaded as a string field, Converted it to inter
```

In [13]:
```
df["Installs"] = df["Installs"].str.replace("+","",regex = True).str.replace(",","",rege
#4.3) Removed symbols from Installs column
```

In [14]:
```
df["Installs"] = df["Installs"].astype("int")
#4.3) Installs column converted to interger
```

In [15]:
```
df["Price"] = df["Price"].str.replace("$","",regex = True)
# 4.4) Price field is a string and has $ symbol. Removed '$' sign
```

In [16]:
```
df["Price"] = df["Price"].astype("float")
# 4.4) Converted Price field to float.
```

In [17]:
```
(df['Rating'] >=1).value_counts()
```

Out[17]:
```
True    10346
Name: Rating, dtype: int64
```

In [18]:
```
(df['Rating'] <= 5 ).value_counts()
#all 10346 values in the rating field are within the specified range of >=1 and <=5
```

Out[18]:
```
True    10346
Name: Rating, dtype: int64
```

In [19]:
```
df['Rating'].mean()
#5.1) avergae rating is 4.2 with the range and all the value falls within the range, not
```

Out[19]:
```
4.217881306785419
```

In [20]:
```
(df['Reviews'] > df['Installs']).sum()
#there are 11 records where review is greater than installs
```

Out[20]:
```
11
```

In [21]:
```
df.drop(df[df['Reviews']>df['Installs']].index,axis =0, inplace=True)
#5.2) All the 11 records where number of reviews greater that number of installs are dro
```

In [22]:
```
df['Type'].describe()
```

Out[22]:
```
count     10335
unique        2
top        Free
freq       9579
Name: Type, dtype: object
```

In [23]:
```
df["Type"].value_counts()
```

Out[23]:
```
Free    9579
Paid     756
Name: Type, dtype: int64
```

In [24]:
```
((df["Type"] == "Free") & (df['Price']>0)).value_counts()
#5.3) All 9579 free app are priced 0, so there is nothing to drop
```

Out[24]:
```
False    10335
dtype: int64
```

In [25]:
```
def turkey_IQR(col):
    Q1 = np.percentile(col, 25)
    Q3 = np.percentile(col, 75)
    IQR = Q3 - Q1
    print("Q1 =",Q1)
    print("Q3 =",Q3)
    print("IQR =",IQR)
    upperf = Q3+1.5*IQR
    lowerf = Q1-1.5*IQR
    print("Lower Fence =",lowerf)
    print("Upper Fence =",upperf)
    upper = np.where(col>upperf)
    lower = np.where(col<lowerf)
    print("Upper Outliers :",upper)
    print("Lower Outliers :",lower)

#function to calculate turkey's fence, IQR etc
```

In [26]:
```
df["Price"].describe()
```

Out[26]:
```
count    10335.000000
mean         1.031891
std         16.295895
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max        400.000000
Name: Price, dtype: float64
```

In [27]:
```
(df["Price"] > 0).value_counts()
#7.8% values are outliers
```

Out[27]:
```
False    9579
True      756
Name: Price, dtype: int64
```

In [28]:
```
df["Price"].unique()
```

Out[28]:
```
array([  0.  ,   4.99,   3.99,   6.99,   1.49,   2.99,   7.99,   5.99,
         3.49,   1.99,   9.99,   7.49,   0.99,   9.  ,   5.49,  10.  ,
        24.99,  11.99,  79.99,  16.99,  14.99,   1.  ,  29.99,  12.99,
         2.49,  10.99,   1.5 ,  19.99,  15.99,  33.99,  74.99,  39.99,
         3.95,   4.49,   1.7 ,   8.99,   2.  ,   3.88,  25.99, 399.99,
        17.99, 400.  ,   3.02,   1.76,   4.84,   4.77,   1.61,   2.5 ,
         1.59,   6.49,   1.29,   5.  ,  13.99, 299.99, 379.99,  37.99,
        18.99, 389.99,  19.9 ,   8.49,   1.75,  14.  ,   4.85,  46.99,
       109.99, 154.99,   3.08,   2.59,   4.8 ,   1.96,  19.4 ,   3.9 ,
         4.59,  15.46,   3.04,   4.29,   2.6 ,   3.28,   4.6 ,  28.99,
         2.95,   2.9 ,   1.97, 200.  ,  89.99,   2.56,  30.99,   3.61,
       394.99,   1.26,   1.2 ,   1.04])
```

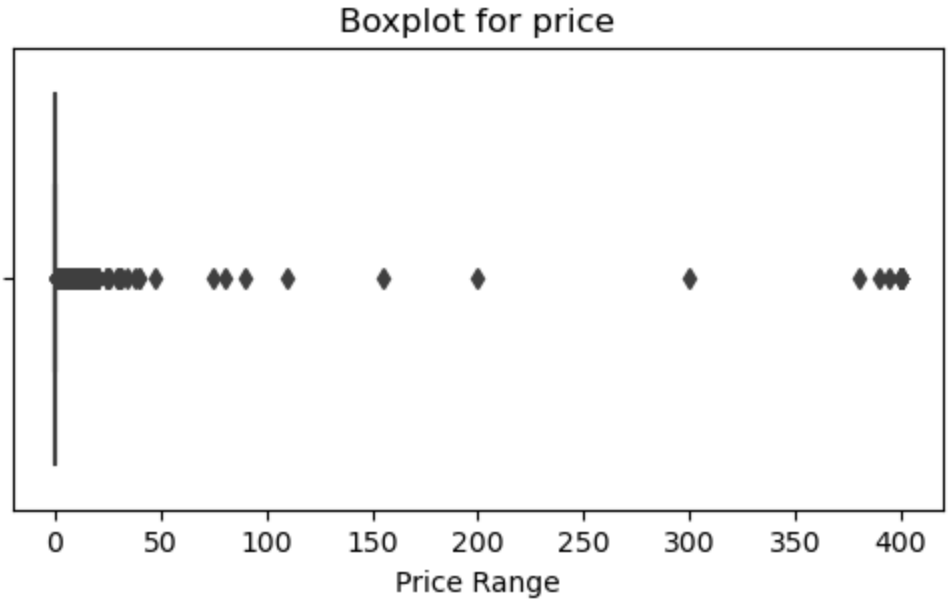
In [29]:
```
turkey_IQR(df["Price"])
```

```
Q1 = 0.0
Q3 = 0.0
IQR = 0.0
Lower Fence = 0.0
Upper Fence = 0.0
Upper Outliers : (array([  232,   233,   389,   428,   429,   430,   431,   432,   433,
         506,   740,   741,   742,   743,   846,   852,  1072,  1073,
        1168,  1176,  1182,  1613,  1614,  1615,  1616,  1617,  1618,
        1619,  1620,  1621,  1830,  1852,  1853,  1854,  1912,  1913,
        1914,  1915,  1916,  1930,  1932,  1933,  1934,  1935,  1936,
        1937,  1938,  1939,  1940,  1941,  1950,  1951,  1952,  1953,
        1981,  1982,  1983,  1984,  1985,  1986,  1987,  1988,  1989,
        1990,  1991,  1992,  1993,  1994,  1995,  1996,  1997,  1998,
        1999,  2000,  2001,  2002,  2003,  2004,  2005,  2006,  2007,
        2008,  2009,  2010,  2011,  2012,  2013,  2014,  2015,  2016,
        2017,  2018,  2019,  2020,  2021,  2022,  2023,  2024,  2025,
        2026,  2027,  2028,  2029,  2030,  2031,  2032,  2033,  2034,
        2035,  2036,  2037,  2038,  2039,  2040,  2041,  2042,  2120,
        2135,  2139,  2140,  2141,  2142,  2529,  2545,  2551,  2662,
        2983,  2985,  2988,  3019,  3020,  3136,  3229,  3509,  3511,
        3516,  3517,  3519,  3527,  3535,  3543,  3547,  3553,  3584,
        3596,  3606,  3634,  3654,  3672,  3677,  3678,  3679,  3682,
        3684,  3685,  3701,  3703,  3706,  3710,  3711,  3714,  3717,
        3718,  3720,  3722,  3723,  3726,  3735,  3737,  3742,  3748,
        3761,  3763,  3803,  3808,  3825,  3836,  3842,  3844,  3859,
        3890,  3901,  3905,  3910,  3923,  3934,  3938,  3940,  3943,
        3946,  3948,  3952,  3954,  3956,  3959,  3960,  3962,  3966,
        3970,  3972,  3974,  3976,  3980,  3992,  3993,  3994,  3997,
        3998,  3999,  4000,  4001,  4002,  4003,  4006,  4007,  4009,
        4012,  4018,  4028,  4029,  4030,  4031,  4048,  4061,  4074,
        4082,  4093,  4095,  4098,  4100,  4110,  4116,  4124,  4145,
        4151,  4152,  4155,  4156,  4158,  4159,  4160,  4176,  4232,
        4235,  4238,  4246,  4249,  4254,  4259,  4281,  4293,  4297,
        4302,  4307,  4311,  4314,  4317,  4319,  4326,  4327,  4351,
        4372,  4380,  4382,  4383,  4386,  4406,  4420,  4439,  4476,
        4484,  4489,  4490,  4493,  4496,  4500,  4501,  4502,  4509,
        4511,  4515,  4518,  4521,  4526,  4543,  4574,  4578,  4589,
        4621,  4632,  4664,  4682,  4718,  4752,  4754,  4758,  4761,
        4774,  4783,  4789,  4794,  4797,  4800,  4802,  4806,  4844,
        4878,  4888,  4891,  4892,  4893,  4894,  4895,  4896,  4897,
        4898,  4899,  4901,  4903,  4904,  4906,  4908,  4910,  4948,
        4949,  4951,  5002,  5003,  5012,  5013,  5016,  5017,  5019,
        5023,  5026,  5027,  5037,  5093,  5114,  5122,  5163,  5167,
        5177,  5181,  5184,  5196,  5237,  5248,  5275,  5287,  5292,
```

```
       5303,  5308,  5313,  5340,  5367,  5369,  5381,  5382,  5444,
       5446,  5447,  5449,  5458,  5461,  5476,  5479,  5489,  5491,
       5492,  5494,  5499,  5506,  5510,  5511,  5515,  5526,  5588,
       5620,  5634,  5647,  5652,  5664,  5672,  5711,  5712,  5713,
       5730,  5733,  5734,  5737,  5749,  5784,  5809,  5824,  5843,
       5872,  5877,  5892,  5897,  5920,  5944,  5956,  5959,  5960,
       5963,  5971,  5977,  5988,  5990,  6022,  6027,  6064,  6068,
       6077,  6079,  6083,  6085,  6089,  6090,  6095,  6097,  6100,
       6120,  6141,  6147,  6175,  6198,  6203,  6210,  6215,  6244,
       6254,  6275,  6282,  6288,  6318,  6329,  6330,  6358,  6407,
       6416,  6418,  6439,  6445,  6449,  6456,  6463,  6468,  6494,
       6505,  6555,  6615,  6618,  6625,  6663,  6666,  6683,  6701,
       6711,  6714,  6716,  6722,  6729,  6733,  6743,  6751,  6755,
       6762,  6773,  6846,  6850,  6852,  6853,  6856,  6859,  6863,
       6864,  6872,  6873,  6874,  6876,  6877,  6881,  6892,  6894,
       6901,  6907,  6932,  6980,  6981,  6992,  7019,  7053,  7070,
       7093,  7095,  7099,  7100,  7101,  7125,  7129,  7132,  7133,
       7134,  7135,  7142,  7143,  7172,  7179,  7251,  7258,  7287,
       7288,  7305,  7311,  7323,  7335,  7345,  7375,  7399,  7405,
       7406,  7409,  7410,  7411,  7444,  7474,  7490,  7505,  7509,
       7511,  7515,  7516,  7518,  7524,  7525,  7526,  7529,  7530,
       7531,  7533,  7563,  7586,  7589,  7597,  7649,  7651,  7652,
       7653,  7657,  7661,  7662,  7663,  7673,  7676,  7683,  7688,
       7703,  7723,  7724,  7731,  7737,  7748,  7754,  7763,  7794,
       7798,  7799,  7833,  7840,  7858,  7860,  7862,  7871,  7874,
       7876,  7880,  7883,  7888,  7961,  7994,  8001,  8023,  8038,
       8067,  8077,  8083,  8086,  8102,  8111,  8115,  8127,  8129,
       8220,  8222,  8223,  8224,  8225,  8227,  8234,  8237,  8256,
       8278,  8279,  8280,  8281,  8286,  8288,  8293,  8294,  8296,
       8301,  8302,  8305,  8324,  8326,  8342,  8346,  8354,  8361,
       8380,  8388,  8391,  8413,  8428,  8451,  8494,  8511,  8525,
       8527,  8529,  8531,  8540,  8546,  8555,  8557,  8558,  8559,
       8561,  8564,  8565,  8566,  8583,  8601,  8604,  8650,  8653,
       8664,  8669,  8705,  8710,  8719,  8726,  8793,  8803,  8824,
       8833,  8836,  8856,  8908,  8964,  8969,  8977,  8979,  8989,
       9040,  9049,  9065,  9068,  9073,  9085,  9091,  9112,  9126,
       9146,  9156,  9168,  9171,  9177,  9178,  9189,  9196,  9199,
       9203,  9209,  9214,  9216,  9218,  9227,  9229,  9284,  9327,
       9347,  9368,  9375,  9404,  9409,  9416,  9419,  9431,  9433,
       9440,  9504,  9505,  9534,  9535,  9536,  9538,  9540,  9542,
       9545,  9546,  9548,  9549,  9550,  9551,  9558,  9564,  9570,
       9571,  9620,  9768,  9939,  9943,  9945,  9947,  9950,  9954,
       9956,  9957, 10013, 10027, 10036, 10066, 10079, 10082, 10090,
      10141, 10146, 10147, 10157, 10158, 10160, 10164, 10165, 10170,
      10171, 10175, 10178, 10186, 10230, 10255, 10276, 10279, 10292],
     dtype=int64),)
Lower Outliers : (array([], dtype=int64),)
```

In [30]:

```
plt.figure(figsize=(6,3))
sns.boxplot(x=df["Price"])
plt.title("Boxplot for price")
plt.xlabel("Price Range")
plt.show()
#5) boxplot of price
```



Univariate analysis of Price: 1) Price range: 0 to 400. 2) Majority of dataset: '0'. 3) Tukey's fences calculation: Q1, Q3, IQR, lower, and upper fences all '0' due to dataset majority. 4) Standard method for identifying outliers: Values below threshold are potential outliers. 5) Understanding further requirements is necessary to identify outliers. Extremely high price could be considered an outlier with a defined threshold. 6) Potential outliers: Sorted unique high prices - 400.0, 399.99, 379.99, 299.99, 200.0, 154.99, 109.99, 89.99, 79.99, 74.99, 46.99, 33.99, 30.99, 29.99, 28.99, 25.99. 7) Outliers account for approximately 7.8% of the data.

In [31]:
```
df["Reviews"].describe()
```

Out[31]:
```
count    1.033500e+04
mean     4.067653e+05
std      2.699582e+06
min      0.000000e+00
25%      3.300000e+01
50%      1.697000e+03
75%      4.677100e+04
max      7.815831e+07
Name: Reviews, dtype: float64
```

In [32]:
```
df["Reviews"].nunique()
```

Out[32]:
```
5998
```

In [33]:
```
np.sort(df["Reviews"].unique())
```

Out[33]:
```
array([       0,        1,        2, ..., 69119316, 78128208, 78158306])
```

In [34]:
```
turkey_IQR(df["Reviews"])
```

```
Q1 = 33.0
Q3 = 46771.0
IQR = 46738.0
Lower Fence = -70074.0
Upper Fence = 116878.0
Upper Outliers : (array([    3,    17,    18, ..., 10278, 10303, 10334], dtype=int64),)
Lower Outliers : (array([], dtype=int64),)
```

In [35]:
```
(df["Reviews"] > 116878.0).value_counts()
```

Out[35]:
```
False    8469
True     1866
Name: Reviews, dtype: int64
```

In [36]:
```
df[df["Reviews"] > 116878.0]["Reviews"].value_counts()
```

Out[36]:
```
484981     2
134203     2
1125438    2
148945     2
182103     2
          ..
412725     1
382120     1
315441     1
3781770    1
398307     1
Name: Reviews, Length: 1858, dtype: int64
```

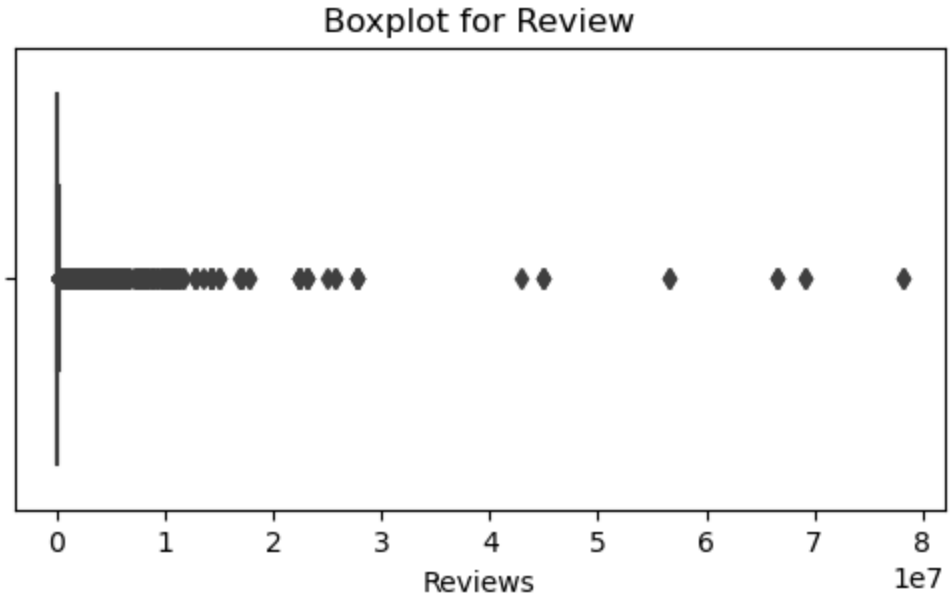
In [37]:
```
df[df["Reviews"] > 116878.0]
#data of apps with rating higher than 116878. The values seems reasonable because they a
```

Out[37]:

	App	Category	Rating	Reviews	Installs	Type	Price	Content Rating	Genres	Last Updated	C
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	50000000	Free	0.0	Teen	Art & Design	June 8, 2018	
18	FlipaClip - Cartoon animation	ART_AND_DESIGN	4.3	194216	5000000	Free	0.0	Everyone	Art & Design	August 3, 2018	
19	ibis Paint X	ART_AND_DESIGN	4.6	224399	10000000	Free	0.0	Everyone	Art & Design	July 30, 2018	
42	Textgram - write on photos	ART_AND_DESIGN	4.4	295221	10000000	Free	0.0	Everyone	Art & Design	July 30, 2018	
45	Canva: Poster, banner, card maker & graphic de...	ART_AND_DESIGN	4.7	174531	10000000	Free	0.0	Everyone	Art & Design	July 31, 2018	
...	...	...	...	...	...	...	...	...	...	...	...
10740	PhotoFunia	PHOTOGRAPHY	4.3	316378	10000000	Free	0.0	Everyone	Photography	June 3, 2017	
10781	Modern Strike Online	GAME	4.3	834117	10000000	Free	0.0	Teen	Action	July 30, 2018	
10784	Big Hunter	GAME	4.3	245455	10000000	Free	0.0	Everyone 10+	Action	May 31, 2018	
10809	Castle Clash: RPG War and Strategy FR	FAMILY	4.7	376223	1000000	Free	0.0	Everyone	Strategy	July 18, 2018	
10840	iHoroscope - 2018 Daily Horoscope & Astrology	LIFESTYLE	4.5	398307	10000000	Free	0.0	Everyone	Lifestyle	July 25, 2018	

1866 rows × 13 columns

In [38]:
```
plt.figure(figsize=(6, 3))
sns.boxplot(x=df["Reviews"])
plt.title("Boxplot for Review")
plt.show()
#box plot of Reviews
```



Univariate analysis of Review: 1) Review has 5998 unique values, ranging from 0 to 78158306. 2) Out of the total apps, 1866 (18.04%) have extremely high review values compared to the majority. This is determined based on IQR values and an upper fence value of 116878. The three highest ratings are 69119316, 78128208, and 78158306 respectively. Currently, 18.04% of the data are considered outliers. 3) The high rating values appear reasonable as they are lower than the number of installs. 4) The data might require normalization.

In [39]:
```
df["Rating"].describe()
```

Out[39]:
```
count    10335.000000
mean         4.217300
std          0.489608
min          1.000000
25%          4.100000
50%          4.400000
75%          4.500000
max          5.000000
Name: Rating, dtype: float64
```

In [40]:
```
turkey_IQR(df["Rating"])
```

```
Q1 = 4.1
Q3 = 4.5
IQR = 0.40000000000000036
Lower Fence = 3.499999999999999
Upper Fence = 5.1000000000000005
Upper Outliers : (array([], dtype=int64),)
Lower Outliers : (array([   86,   158,   175,   208,   277,   278,   291,   429,   444,
         453,   462,   465,   466,   470,   472,   474,   479,   480,
         485,   494,   496,   507,   513,   544,   552,   558,   578,
         596,   604,   611,   751,   758,   774,   802,   907,   908,
        1031,  1035,  1058,  1062,  1105,  1279,  1298,  1328,  1348,
        1360,  1370,  1375,  1390,  1393,  1565,  1567,  1854,  1989,
        2000,  2004,  2006,  2035,  2040,  2057,  2067,  2069,  2077,
        2091,  2095,  2100,  2127,  2130,  2132,  2133,  2145,  2202,
        2206,  2232,  2236,  2386,  2472,  2636,  2646,  2647,  2658,
        2732,  2734,  2738,  2740,  2747,  2766,  2828,  2831,  2843,
        2849,  2852,  2854,  2860,  3038,  3061,  3269,  3280,  3312,
        3319,  3443,  3490,  3507,  3513,  3514,  3515,  3522,  3529,
        3556,  3560,  3611,  3615,  3627,  3672,  3680,  3706,  3709,
        3710,  3713,  3716,  3726,  3732,  3739,  3754,  3755,  3770,
```

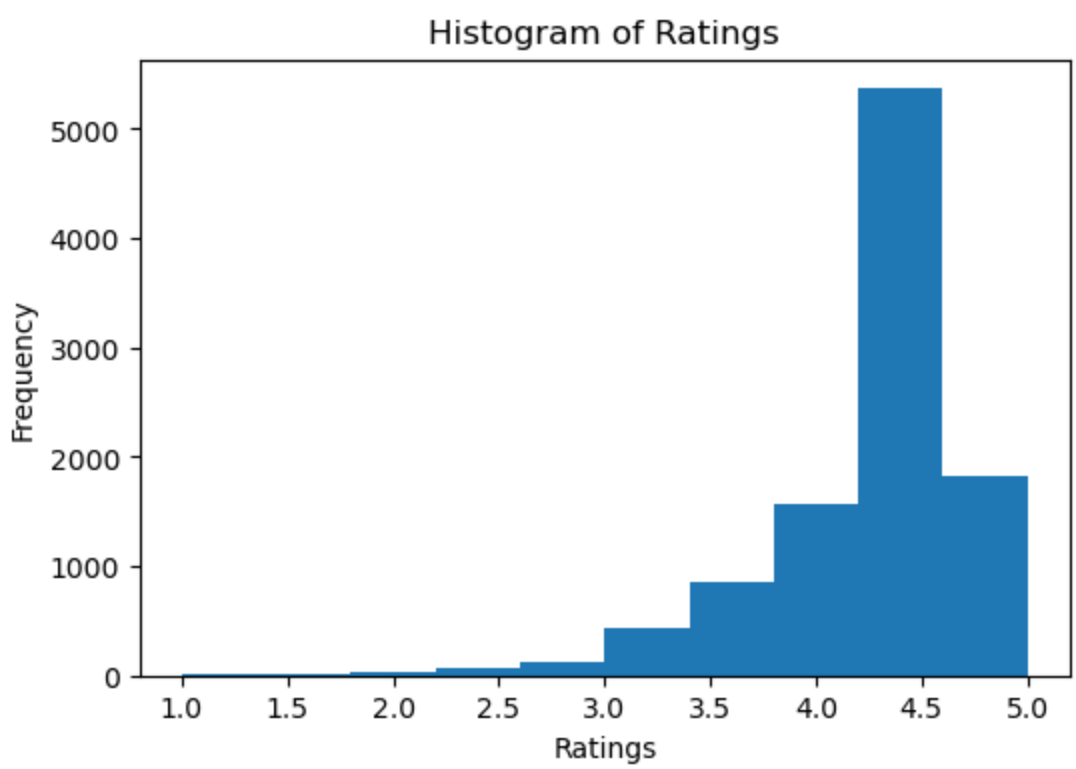
3774, 3794, 3796, 3801, 3874, 3904, 3916, 3918, 3929,
3932, 3935, 4011, 4032, 4033, 4037, 4038, 4049, 4051,
4052, 4062, 4087, 4091, 4134, 4136, 4138, 4147, 4153,
4177, 4180, 4181, 4185, 4186, 4189, 4190, 4192, 4196,
4203, 4207, 4227, 4231, 4259, 4264, 4266, 4290, 4301,
4325, 4328, 4366, 4369, 4405, 4438, 4440, 4441, 4442,
4443, 4444, 4446, 4447, 4448, 4450, 4451, 4457, 4460,
4463, 4464, 4467, 4470, 4471, 4472, 4473, 4508, 4511,
4519, 4533, 4577, 4579, 4583, 4586, 4592, 4635, 4640,
4642, 4671, 4675, 4686, 4688, 4710, 4712, 4716, 4718,
4729, 4734, 4737, 4749, 4752, 4753, 4770, 4773, 4776,
4828, 4829, 4850, 4894, 4963, 4980, 4984, 4995, 5016,
5025, 5051, 5086, 5091, 5092, 5101, 5115, 5116, 5140,
5150, 5172, 5186, 5194, 5196, 5198, 5233, 5241, 5242,
5245, 5246, 5247, 5248, 5252, 5258, 5272, 5287, 5297,
5305, 5306, 5339, 5341, 5344, 5393, 5409, 5410, 5413,
5440, 5443, 5448, 5449, 5451, 5475, 5483, 5496, 5507,
5511, 5520, 5522, 5527, 5540, 5541, 5548, 5553, 5571,
5573, 5620, 5623, 5631, 5644, 5653, 5655, 5659, 5665,
5667, 5669, 5673, 5688, 5693, 5700, 5728, 5733, 5753,
5755, 5779, 5780, 5781, 5785, 5795, 5840, 5851, 5858,
5890, 5910, 5935, 5939, 5942, 5982, 5992, 5994, 6021,
6031, 6035, 6041, 6043, 6062, 6111, 6112, 6113, 6116,
6125, 6130, 6142, 6143, 6145, 6148, 6163, 6168, 6184,
6186, 6200, 6208, 6238, 6240, 6279, 6287, 6289, 6291,
6295, 6312, 6314, 6327, 6333, 6335, 6358, 6405, 6412,
6415, 6418, 6420, 6424, 6430, 6443, 6468, 6478, 6481,
6490, 6529, 6536, 6549, 6563, 6566, 6568, 6580, 6588,
6593, 6600, 6603, 6606, 6607, 6608, 6615, 6621, 6622,
6636, 6644, 6662, 6680, 6693, 6705, 6718, 6756, 6760,
6765, 6768, 6771, 6772, 6784, 6804, 6869, 6885, 6900,
6922, 6926, 6928, 6931, 6940, 6942, 6946, 6971, 6979,
6998, 7026, 7027, 7033, 7039, 7062, 7083, 7129, 7147,
7149, 7150, 7155, 7158, 7167, 7172, 7183, 7191, 7236,
7259, 7271, 7292, 7302, 7309, 7311, 7316, 7319, 7332,
7342, 7351, 7365, 7376, 7399, 7406, 7420, 7424, 7426,
7438, 7440, 7442, 7478, 7487, 7489, 7490, 7501, 7502,
7503, 7528, 7535, 7537, 7543, 7545, 7547, 7553, 7556,
7557, 7558, 7560, 7571, 7604, 7641, 7643, 7647, 7648,
7653, 7654, 7674, 7712, 7723, 7728, 7733, 7741, 7748,
7770, 7796, 7797, 7828, 7877, 7879, 7880, 7882, 7885,
7889, 7897, 7899, 7914, 7934, 7942, 7981, 7986, 7988,
7990, 7993, 7997, 8020, 8055, 8061, 8063, 8064, 8065,
8067, 8069, 8073, 8076, 8081, 8083, 8096, 8097, 8101,
8103, 8105, 8120, 8126, 8143, 8217, 8231, 8235, 8238,
8240, 8245, 8268, 8291, 8303, 8308, 8309, 8312, 8313,
8320, 8321, 8327, 8331, 8333, 8339, 8356, 8358, 8360,
8376, 8382, 8393, 8417, 8428, 8433, 8434, 8437, 8439,
8443, 8445, 8446, 8447, 8449, 8452, 8453, 8468, 8471,
8476, 8477, 8480, 8483, 8492, 8512, 8523, 8570, 8571,
8573, 8654, 8667, 8668, 8675, 8676, 8681, 8703, 8710,
8712, 8729, 8735, 8743, 8766, 8769, 8772, 8775, 8776,
8778, 8782, 8784, 8785, 8786, 8787, 8791, 8816, 8826,
8831, 8833, 8835, 8843, 8873, 8893, 8905, 8906, 8914,
8921, 8924, 8925, 8929, 8932, 8934, 8942, 8948, 9012,
9041, 9042, 9046, 9090, 9148, 9193, 9210, 9264, 9275,
9280, 9282, 9299, 9308, 9316, 9366, 9368, 9389, 9394,
9407, 9425, 9432, 9445, 9446, 9447, 9449, 9450, 9457,
9458, 9472, 9500, 9519, 9523, 9540, 9542, 9545, 9546,
9548, 9550, 9574, 9575, 9580, 9613, 9626, 9627, 9628,
9629, 9631, 9634, 9635, 9640, 9643, 9647, 9649, 9650,
9655, 9657, 9662, 9673, 9702, 9733, 9748, 9765, 9811,
9820, 9822, 9823, 9839, 9857, 9894, 9897, 9921, 9925,
9934, 9951, 9955, 9961, 9967, 9970, 9972, 9974, 10058,
10074, 10076, 10087, 10088, 10106, 10111, 10120, 10126, 10135,

```
       10144, 10161, 10173, 10198, 10210, 10238, 10252, 10260, 10261,
       10313, 10322], dtype=int64),)
```

In [41]:
```
df["Rating"].value_counts()
```

Out[41]:
```
4.4    2487
4.3    1016
4.5     976
4.2     887
4.6     768
4.1     656
4.0     538
4.7     484
3.9     372
3.8     293
5.0     265
3.7     231
4.8     227
3.6     169
3.5     157
3.4     127
3.3     101
4.9      87
3.0      82
3.1      69
3.2      63
2.9      45
2.8      40
2.6      24
2.7      23
2.5      20
2.3      20
2.4      19
1.0      16
2.2      14
1.9      12
2.0      12
1.7       8
1.8       8
2.1       8
1.6       4
1.4       3
1.5       3
1.2       1
Name: Rating, dtype: int64
```

In [42]:
```
plt.figure(figsize=(6,4))
plt.hist(df["Rating"])
plt.xlabel("Ratings")
plt.ylabel("Frequency")
plt.title("Histogram of Ratings")
plt.show()
#5) Histogram of Ratings
```



Univariate analysis: Histogram of Rating: 1) The majority of the ratings are higher, with the highest count observed at 4.4 (2487 counts). Other significant counts include 4.3 (1016), 4.5 (976), 4.2 (887), and 4.6 (768). 2) The bin ranging from 4.2 to 4.6 has the highest count according to the histogram, indicating it as the majority range. 3) There are fewer counts towards the left side of the histogram.

In [43]:
```
df["Size"].describe()
#size of app is in kb
```

Out[43]:
```
count     10335.000000
mean      19787.720610
std       21137.961154
min           8.500000
25%        5700.000000
50%       11000.000000
75%       26000.000000
max      100000.000000
Name: Size, dtype: float64
```

In [44]:
```
np.sort(df["Size"].unique())
```

Out[44]:
```
array([8.50e+00, 1.40e+01, 1.70e+01, 1.80e+01, 2.00e+01, 2.30e+01,
       2.40e+01, 2.50e+01, 2.60e+01, 2.70e+01, 2.80e+01, 2.90e+01,
       3.30e+01, 3.40e+01, 3.90e+01, 4.10e+01, 4.40e+01, 4.50e+01,
       4.80e+01, 5.00e+01, 5.10e+01, 5.40e+01, 5.50e+01, 5.80e+01,
       6.10e+01, 6.70e+01, 7.00e+01, 7.20e+01, 7.30e+01, 7.40e+01,
       7.80e+01, 7.90e+01, 8.10e+01, 8.20e+01, 8.90e+01, 9.10e+01,
       9.30e+01, 9.70e+01, 1.03e+02, 1.08e+02, 1.16e+02, 1.18e+02,
       1.21e+02, 1.22e+02, 1.41e+02, 1.43e+02, 1.44e+02, 1.53e+02,
       1.54e+02, 1.57e+02, 1.60e+02, 1.61e+02, 1.64e+02, 1.69e+02,
       1.70e+02, 1.72e+02, 1.73e+02, 1.75e+02, 1.76e+02, 1.86e+02,
       1.90e+02, 1.91e+02, 1.92e+02, 1.93e+02, 1.96e+02, 2.00e+02,
       2.01e+02, 2.03e+02, 2.06e+02, 2.08e+02, 2.09e+02, 2.10e+02,
       2.19e+02, 2.20e+02, 2.21e+02, 2.26e+02, 2.28e+02, 2.32e+02,
       2.34e+02, 2.39e+02, 2.40e+02, 2.41e+02, 2.43e+02, 2.45e+02,
       2.46e+02, 2.51e+02, 2.53e+02, 2.57e+02, 2.59e+02, 2.66e+02,
       2.69e+02, 2.70e+02, 2.80e+02, 2.83e+02, 2.88e+02, 2.92e+02,
       2.93e+02, 3.06e+02, 3.08e+02, 3.09e+02, 3.13e+02, 3.14e+02,
       3.17e+02, 3.18e+02, 3.19e+02, 3.22e+02, 3.23e+02, 3.29e+02,
       3.34e+02, 3.35e+02, 3.50e+02, 3.51e+02, 3.53e+02, 3.64e+02,
       3.71e+02, 3.73e+02, 3.75e+02, 3.76e+02, 3.78e+02, 3.83e+02,
       3.87e+02, 4.00e+02, 4.04e+02, 4.11e+02, 4.12e+02, 4.14e+02,
```

```
       4.17e+02, 4.20e+02, 4.21e+02, 4.29e+02, 4.30e+02, 4.37e+02,
       4.42e+02, 4.44e+02, 4.54e+02, 4.58e+02, 4.59e+02, 4.60e+02,
       4.67e+02, 4.70e+02, 4.73e+02, 4.75e+02, 4.78e+02, 4.85e+02,
       4.96e+02, 4.98e+02, 4.99e+02, 5.00e+02, 5.06e+02, 5.11e+02,
       5.14e+02, 5.16e+02, 5.18e+02, 5.23e+02, 5.25e+02, 5.26e+02,
       5.40e+02, 5.44e+02, 5.45e+02, 5.49e+02, 5.51e+02, 5.52e+02,
       5.54e+02, 5.56e+02, 5.62e+02, 5.69e+02, 5.82e+02, 5.85e+02,
       5.92e+02, 5.97e+02, 5.98e+02, 6.00e+02, 6.01e+02, 6.08e+02,
       6.09e+02, 6.13e+02, 6.19e+02, 6.24e+02, 6.26e+02, 6.29e+02,
       6.36e+02, 6.42e+02, 6.43e+02, 6.47e+02, 6.55e+02, 6.56e+02,
       6.63e+02, 6.76e+02, 6.83e+02, 6.88e+02, 6.91e+02, 6.95e+02,
       6.96e+02, 7.04e+02, 7.05e+02, 7.13e+02, 7.14e+02, 7.16e+02,
       7.17e+02, 7.20e+02, 7.21e+02, 7.28e+02, 7.30e+02, 7.43e+02,
       7.46e+02, 7.49e+02, 7.54e+02, 7.56e+02, 7.72e+02, 7.75e+02,
       7.78e+02, 7.79e+02, 7.80e+02, 7.82e+02, 7.84e+02, 7.85e+02,
       7.87e+02, 8.01e+02, 8.09e+02, 8.11e+02, 8.12e+02, 8.16e+02,
       8.18e+02, 8.37e+02, 8.40e+02, 8.42e+02, 8.47e+02, 8.53e+02,
       8.57e+02, 8.60e+02, 8.61e+02, 8.62e+02, 8.65e+02, 8.72e+02,
       8.74e+02, 8.79e+02, 8.81e+02, 8.85e+02, 8.87e+02, 8.92e+02,
       8.98e+02, 8.99e+02, 9.02e+02, 9.03e+02, 9.04e+02, 9.13e+02,
       9.14e+02, 9.16e+02, 9.20e+02, 9.21e+02, 9.24e+02, 9.30e+02,
       9.39e+02, 9.40e+02, 9.42e+02, 9.48e+02, 9.51e+02, 9.53e+02,
       9.54e+02, 9.57e+02, 9.61e+02, 9.63e+02, 9.65e+02, 9.70e+02,
       9.75e+02, 9.76e+02, 9.80e+02, 9.81e+02, 9.82e+02, 9.86e+02,
       9.92e+02, 9.94e+02, 1.00e+03, 1.02e+03, 1.10e+03, 1.20e+03,
       1.30e+03, 1.40e+03, 1.50e+03, 1.60e+03, 1.70e+03, 1.80e+03,
       1.90e+03, 2.00e+03, 2.10e+03, 2.20e+03, 2.30e+03, 2.40e+03,
       2.50e+03, 2.60e+03, 2.70e+03, 2.80e+03, 2.90e+03, 3.00e+03,
       3.10e+03, 3.20e+03, 3.30e+03, 3.40e+03, 3.50e+03, 3.60e+03,
       3.70e+03, 3.80e+03, 3.90e+03, 4.00e+03, 4.10e+03, 4.20e+03,
       4.30e+03, 4.40e+03, 4.50e+03, 4.60e+03, 4.70e+03, 4.80e+03,
       4.90e+03, 5.00e+03, 5.10e+03, 5.20e+03, 5.30e+03, 5.40e+03,
       5.50e+03, 5.60e+03, 5.70e+03, 5.80e+03, 5.90e+03, 6.00e+03,
       6.10e+03, 6.20e+03, 6.30e+03, 6.40e+03, 6.50e+03, 6.60e+03,
       6.70e+03, 6.80e+03, 6.90e+03, 7.00e+03, 7.10e+03, 7.20e+03,
       7.30e+03, 7.40e+03, 7.50e+03, 7.60e+03, 7.70e+03, 7.80e+03,
       7.90e+03, 8.00e+03, 8.10e+03, 8.20e+03, 8.30e+03, 8.40e+03,
       8.50e+03, 8.60e+03, 8.70e+03, 8.80e+03, 8.90e+03, 9.00e+03,
       9.10e+03, 9.20e+03, 9.30e+03, 9.40e+03, 9.50e+03, 9.60e+03,
       9.70e+03, 9.80e+03, 9.90e+03, 1.00e+04, 1.10e+04, 1.20e+04,
       1.30e+04, 1.40e+04, 1.50e+04, 1.60e+04, 1.70e+04, 1.80e+04,
       1.90e+04, 2.00e+04, 2.10e+04, 2.20e+04, 2.30e+04, 2.40e+04,
       2.50e+04, 2.60e+04, 2.70e+04, 2.80e+04, 2.90e+04, 3.00e+04,
       3.10e+04, 3.20e+04, 3.30e+04, 3.40e+04, 3.50e+04, 3.60e+04,
       3.70e+04, 3.80e+04, 3.90e+04, 4.00e+04, 4.10e+04, 4.20e+04,
       4.30e+04, 4.40e+04, 4.50e+04, 4.60e+04, 4.70e+04, 4.80e+04,
       4.90e+04, 5.00e+04, 5.10e+04, 5.20e+04, 5.30e+04, 5.40e+04,
       5.50e+04, 5.60e+04, 5.70e+04, 5.80e+04, 5.90e+04, 6.00e+04,
       6.10e+04, 6.20e+04, 6.30e+04, 6.40e+04, 6.50e+04, 6.60e+04,
       6.70e+04, 6.80e+04, 6.90e+04, 7.00e+04, 7.10e+04, 7.20e+04,
       7.30e+04, 7.40e+04, 7.50e+04, 7.60e+04, 7.70e+04, 7.80e+04,
       7.90e+04, 8.00e+04, 8.10e+04, 8.20e+04, 8.30e+04, 8.40e+04,
       8.50e+04, 8.60e+04, 8.70e+04, 8.80e+04, 8.90e+04, 9.00e+04,
       9.10e+04, 9.20e+04, 9.30e+04, 9.40e+04, 9.50e+04, 9.60e+04,
       9.70e+04, 9.80e+04, 9.90e+04, 1.00e+05])
```

In [45]:
```
(df["Size"].value_counts())
```

Out[45]:
```
11000.0    1711
13000.0     186
12000.0     186
14000.0     181
15000.0     174
           ...
430.0         1
429.0         1
```

```
200.0         1
460.0         1
619.0         1
Name: Size, Length: 454, dtype: int64
```

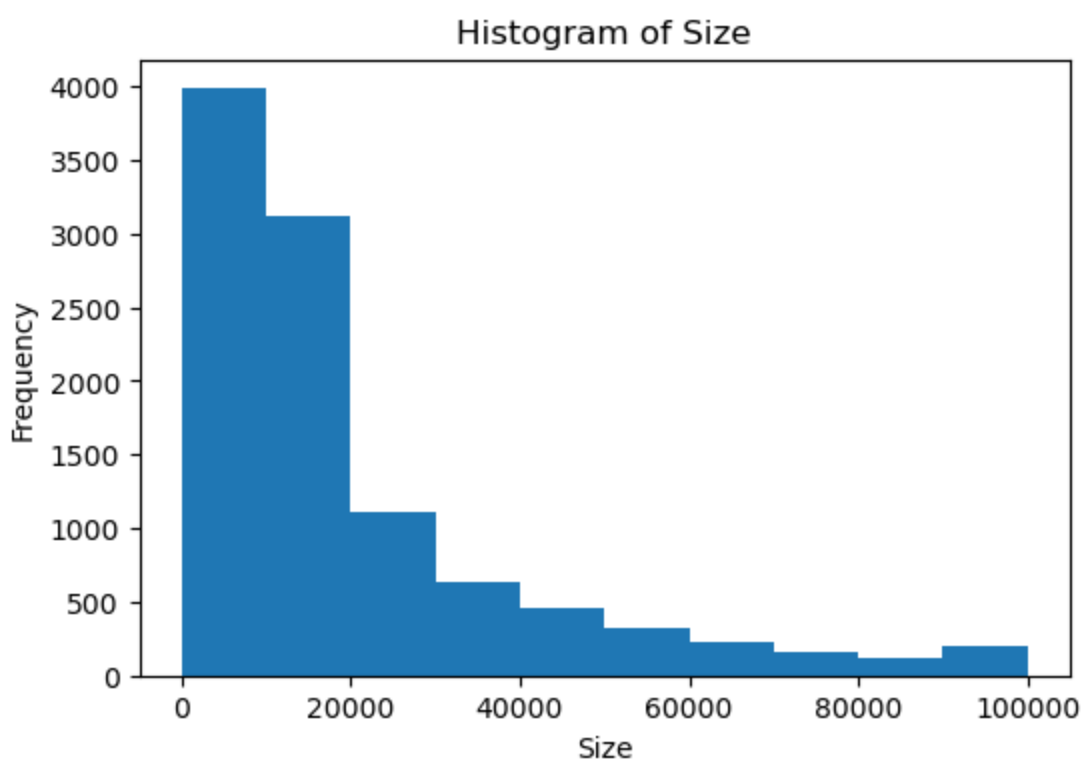
In [46]:
```
turkey_IQR(df["Size"])
```

```
Q1 = 5700.0
Q3 = 26000.0
IQR = 20300.0
Lower Fence = -24750.0
Upper Fence = 56450.0
Upper Outliers : (array([   50,   121,   168,   345,   346,   504,   533,   548,   643,
         660,   695,   703,   725,   735,   740,   779,   782,   790,
         839,   853,   879,   904,   906,   923,   928,   931,   984,
        1012,  1014,  1031,  1034,  1077,  1130,  1138,  1140,  1141,
        1143,  1150,  1154,  1155,  1160,  1162,  1186,  1215,  1226,
        1243,  1298,  1322,  1331,  1364,  1365,  1398,  1444,  1445,
        1446,  1451,  1452,  1457,  1458,  1459,  1461,  1468,  1470,
        1471,  1478,  1479,  1480,  1483,  1484,  1486,  1487,  1488,
        1491,  1492,  1495,  1496,  1497,  1499,  1500,  1503,  1504,
        1506,  1510,  1513,  1515,  1517,  1520,  1521,  1525,  1527,
        1528,  1533,  1534,  1536,  1538,  1541,  1544,  1548,  1549,
        1560,  1561,  1563,  1564,  1567,  1572,  1574,  1576,  1578,
        1579,  1580,  1582,  1583,  1584,  1587,  1591,  1593,  1594,
        1598,  1602,  1603,  1604,  1605,  1606,  1607,  1609,  1610,
        1611,  1612,  1613,  1616,  1618,  1623,  1624,  1625,  1631,
        1633,  1634,  1635,  1637,  1639,  1641,  1642,  1644,  1645,
        1648,  1650,  1651,  1653,  1654,  1659,  1660,  1661,  1662,
        1667,  1670,  1671,  1673,  1674,  1678,  1679,  1682,  1683,
        1686,  1687,  1690,  1694,  1699,  1700,  1701,  1702,  1703,
        1704,  1705,  1708,  1709,  1710,  1711,  1716,  1722,  1725,
        1731,  1732,  1733,  1738,  1739,  1747,  1749,  1750,  1753,
        1756,  1758,  1760,  1766,  1768,  1775,  1779,  1784,  1806,
        1813,  1826,  1829,  1830,  1832,  1835,  1836,  1841,  1842,
        1844,  1845,  1848,  1851,  1852,  1854,  1855,  1857,  1861,
        1863,  1864,  1866,  1868,  1869,  1870,  1874,  1883,  1884,
        1889,  1890,  1891,  1895,  1897,  1901,  1908,  1910,  1911,
        1927,  1936,  1938,  1945,  1946,  1948,  1952,  1954,  1958,
        1960,  1965,  1971,  1972,  1975,  1977,  1979,  2020,  2024,
        2039,  2041,  2060,  2073,  2089,  2091,  2114,  2119,  2124,
        2141,  2152,  2174,  2207,  2222,  2270,  2291,  2305,  2307,
        2313,  2324,  2386,  2401,  2450,  2454,  2528,  2591,  2651,
        2655,  2661,  2664,  2676,  2684,  2709,  2712,  2742,  2754,
        2760,  2761,  2771,  2774,  2855,  3051,  3104,  3140,  3151,
        3153,  3157,  3170,  3192,  3263,  3285,  3394,  3395,  3416,
        3426,  3430,  3435,  3436,  3448,  3449,  3454,  3460,  3462,
        3464,  3468,  3477,  3483,  3487,  3488,  3489,  3494,  3497,
        3499,  3504,  3515,  3523,  3525,  3527,  3536,  3537,  3541,
        3544,  3553,  3567,  3589,  3590,  3592,  3594,  3597,  3604,
        3605,  3608,  3609,  3622,  3624,  3635,  3638,  3655,  3686,
        3693,  3713,  3729,  3733,  3736,  3756,  3764,  3773,  3778,
        3787,  3792,  3804,  3811,  3832,  3834,  3835,  3840,  3848,
        3857,  3861,  3864,  3870,  3875,  3877,  3882,  3892,  3906,
        3920,  3924,  3926,  3939,  3946,  3947,  3950,  3951,  3958,
        3967,  4018,  4034,  4038,  4055,  4067,  4073,  4075,  4076,
        4095,  4098,  4099,  4120,  4123,  4126,  4133,  4169,  4215,
        4216,  4217,  4228,  4230,  4321,  4333,  4336,  4337,  4339,
        4344,  4345,  4354,  4355,  4356,  4357,  4358,  4361,  4363,
        4364,  4365,  4368,  4370,  4371,  4372,  4373,  4375,  4376,
        4380,  4394,  4397,  4398,  4399,  4402,  4408,  4424,  4439,
        4530,  4553,  4616,  4654,  4663,  4671,  4676,  4695,  4726,
        4732,  4760,  4764,  4765,  4783,  4798,  4804,  4814,  4847,
        4876,  4879,  4887,  4912,  4922,  4926,  4934,  4935,  4945,
        4954,  4956,  4959,  4964,  4965,  4966,  4967,  4968,  4974,
        4979,  5016,  5026,  5027,  5054,  5067,  5068,  5072,  5076,
```

```
       5077,  5078,  5089,  5092,  5093,  5095,  5098,  5099,  5107,
       5108,  5125,  5131,  5132,  5135,  5146,  5152,  5158,  5171,
       5173,  5177,  5178,  5185,  5222,  5227,  5238,  5264,  5282,
       5300,  5302,  5318,  5356,  5358,  5395,  5397,  5399,  5400,
       5464,  5482,  5484,  5491,  5517,  5518,  5611,  5619,  5629,
       5636,  5638,  5645,  5711,  5713,  5738,  5825,  5833,  5841,
       5865,  5915,  5937,  5940,  5970,  6077,  6078,  6081,  6082,
       6096,  6199,  6221,  6223,  6229,  6234,  6241,  6242,  6247,
       6248,  6267,  6270,  6274,  6332,  6345,  6394,  6397,  6402,
       6411,  6496,  6506,  6539,  6560,  6605,  6667,  6685,  6747,
       6840,  6853,  6856,  6881,  6890,  6920,  6923,  6930,  6936,
       6943,  6944,  6951,  6952,  6958,  6964,  6991,  7030,  7040,
       7106,  7110,  7116,  7117,  7119,  7123,  7126,  7131,  7136,
       7138,  7143,  7144,  7145,  7229,  7241,  7314,  7324,  7338,
       7359,  7367,  7377,  7412,  7444,  7449,  7532,  7546,  7557,
       7569,  7597,  7619,  7684,  7705,  7707,  7725,  7757,  7762,
       7765,  7767,  7768,  7771,  7773,  7780,  7784,  7785,  7808,
       7812,  7826,  7848,  7911,  7918,  7921,  7923,  7938,  7941,
       7945,  7949,  7963,  7965,  7972,  7974,  7981,  7986,  7990,
       7993,  8065,  8085,  8121,  8135,  8136,  8144,  8171,  8218,
       8222,  8247,  8249,  8250,  8261,  8264,  8265,  8280,  8287,
       8292,  8297,  8298,  8301,  8302,  8316,  8336,  8348,  8361,
       8364,  8387,  8418,  8437,  8445,  8465,  8490,  8540,  8567,
       8584,  8641,  8643,  8645,  8648,  8651,  8655,  8657,  8658,
       8664,  8665,  8667,  8669,  8670,  8694,  8709,  8835,  8847,
       8850,  8852,  8857,  8858,  8866,  8881,  8890,  8902,  8903,
       8904,  8908,  8939,  8940,  8953,  8962,  8968,  8975,  8987,
       9019,  9032,  9035,  9037,  9044,  9050,  9063,  9068,  9082,
       9083,  9100,  9104,  9106,  9111,  9113,  9115,  9167,  9168,
       9170,  9171,  9176,  9177,  9181,  9182,  9185,  9186,  9187,
       9196,  9231,  9234,  9235,  9236,  9237,  9249,  9251,  9257,
       9261,  9263,  9272,  9273,  9336,  9349,  9352,  9353,  9354,
       9360,  9389,  9431,  9445,  9454,  9473,  9482,  9483,  9487,
       9504,  9508,  9510,  9512,  9515,  9520,  9526,  9530,  9531,
       9558,  9562,  9598,  9618,  9654,  9664,  9672,  9677,  9684,
       9690,  9744,  9757,  9765,  9769,  9773,  9797,  9842,  9859,
       9881,  9884,  9888,  9891,  9893,  9966, 10003, 10022, 10084,
      10125, 10169, 10225, 10241, 10252, 10273, 10277, 10278, 10287,
      10297, 10318], dtype=int64),)
Lower Outliers : (array([], dtype=int64),)
```

In [47]:

```
plt.figure(figsize=(6,4))
plt.hist(df["Size"])
plt.xlabel("Size")
plt.ylabel("Frequency")
plt.title("Histogram of Size")
plt.show()
#5) Histogram of Size
```



Univariate analysis: Histogram of Rating: 1) The size field consists of 454 unique values, ranging from 8.5 to 100,000 kb. 2) The histogram reveals that there is a higher number of apps with smaller sizes, and fewer apps towards the right side of the graph. 3) The majority of the apps fall within the bin range of 10,000 to 20,000.All the columns, including Price, Reviews, Rating, and Size, exhibit outliers.

In [48]:
```
(df["Price"] >= 200).value_counts()
```

Out[48]:
```
False    10317
True        18
Name: Price, dtype: int64
```

In [49]:
```
df[df["Price"] >=200]
#6.1) details of apps with high price, price greater than 200.
```

Out[49]:

	App	Category	Rating	Reviews	Installs	Type	Price	Content Rating	Genres	Last Updated	Curren Ve
4197	most expensive app (H)	FAMILY	4.3	6	100	Paid	399.99	Everyone	Entertainment	July 16, 2018	1
4362	💎 I'm rich	LIFESTYLE	3.8	718	10000	Paid	399.99	Everyone	Lifestyle	March 11, 2018	1.0
4367	I'm Rich - Trump Edition	LIFESTYLE	3.6	275	10000	Paid	400.00	Everyone	Lifestyle	May 3, 2018	1.0
5351	I am rich	LIFESTYLE	3.8	3547	100000	Paid	399.99	Everyone	Lifestyle	January 12, 2018	2
5354	I am Rich Plus	FAMILY	4.0	856	10000	Paid	399.99	Everyone	Entertainment	May 19, 2018	3
5355	I am rich VIP	LIFESTYLE	3.8	411	10000	Paid	299.99	Everyone	Lifestyle	July 21, 2018	1.1
5356	I Am Rich Premium	FINANCE	4.1	1867	50000	Paid	399.99	Everyone	Finance	November 12, 2017	1
5357	I am extremely Rich	LIFESTYLE	2.9	41	1000	Paid	379.99	Everyone	Lifestyle	July 1, 2018	1

5358	I am Rich!	FINANCE	3.8	93	1000	Paid	399.99	Everyone	Finance	December 11, 2017	1
5359	I am rich(premium)	FINANCE	3.5	472	5000	Paid	399.99	Everyone	Finance	May 1, 2017	3
5362	I Am Rich Pro	FAMILY	4.4	201	5000	Paid	399.99	Everyone	Entertainment	May 30, 2017	1.5
5364	I am rich (Most expensive app)	FINANCE	4.1	129	1000	Paid	399.99	Teen	Finance	December 6, 2017	
5366	I Am Rich	FAMILY	3.6	217	10000	Paid	389.99	Everyone	Entertainment	June 22, 2018	1
5369	I am Rich	FINANCE	4.3	180	5000	Paid	399.99	Everyone	Finance	March 22, 2018	1
5373	I AM RICH PRO PLUS	FINANCE	4.0	36	1000	Paid	399.99	Everyone	Finance	June 25, 2018	1.0
9719	EP Cook Book	MEDICAL	4.4	0	0	Paid	200.00	Everyone	Medical	July 26, 2015	1
9917	Eu Sou Rico	FINANCE	4.4	0	0	Paid	394.99	Everyone	Finance	July 11, 2018	1
9934	I'm Rich/Eu sou Rico/أنا غني/我很有錢	LIFESTYLE	4.4	0	0	Paid	399.99	Everyone	Lifestyle	December 1, 2017	MONE

6.1.1)App with price above 200: 1) Apps with prices above 200 raise suspicion as they all share the same name, "I'm Rich," which suggests they may be scam apps intended to deceive and defraud customers. 2) The app "EP Cook Book" is priced at 200 but has no downloads or ratings. This app seems inappropriate to include as it is excessively expensive for a cookbook, and its categorization as "Medical" is also incorrect. 3) It is recommended to remove these app data, as promoting such applications goes against ethical considerations.

In [50]:
```
df.drop(df[df["Price"] >=200].index,axis =0,inplace = True)
#6.1.2)All 18 apps with price greater than 200 is dropped.
```

In [51]:
```
(df["Reviews"] >2000000).value_counts()
#6.2) there are 408 apps that has reviews greater than 2million - contributes to 4% of t
```

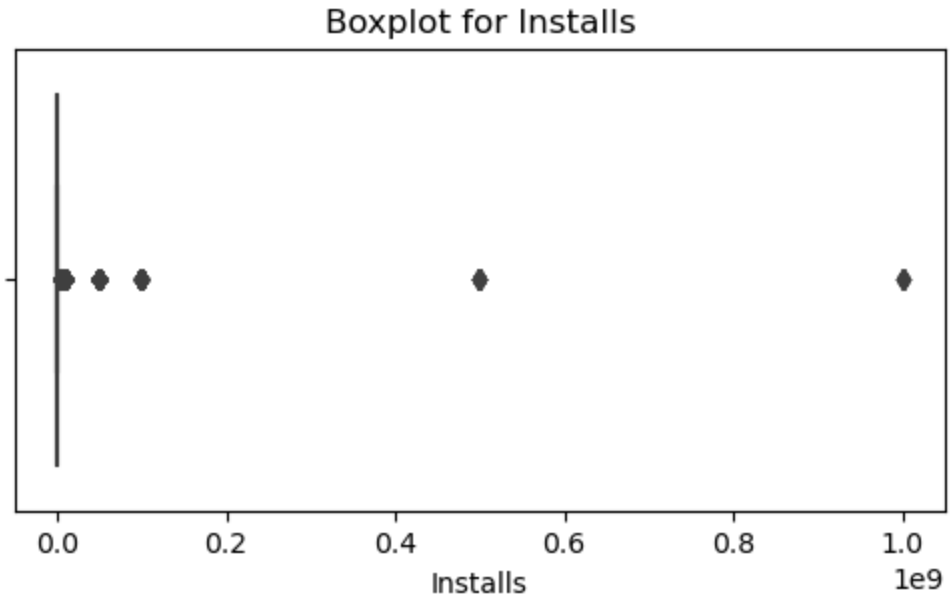
Out[51]:
```
False    9909
True      408
Name: Reviews, dtype: int64
```

In [52]:
```
df.drop(df[df["Reviews"] >2000000].index,axis =0,inplace = True)
#6.2) Dropping 4% of the high review apps to avoid skewing
```

In [53]:
```
turkey_IQR(df["Installs"])
```

```
Q1 = 1000.0
Q3 = 1000000.0
IQR = 999000.0
Lower Fence = -1497500.0
Upper Fence = 2498500.0
Upper Outliers : (array([   2,    3,   12, ..., 9852, 9894, 9908], dtype=int64),)
Lower Outliers : (array([], dtype=int64),)
```

In [54]:
```
plt.figure(figsize=(6, 3))
sns.boxplot(x=df["Installs"])
plt.title("Boxplot for Installs")
plt.show()
#box plot of Reviews
```



In [55]:
```
#6.3.1) Finding out the different percentiles - 10, 25, 50, 70, 90, 95, 99.
print("10th Percentile =", np.percentile(df["Installs"], 10))
print("25th Percentile =", np.percentile(df["Installs"], 25))
print("50th Percentile =", np.percentile(df["Installs"], 50))
print("70th Percentile =", np.percentile(df["Installs"], 70))
print("90th Percentile =", np.percentile(df["Installs"], 90))
print("95th Percentile =", np.percentile(df["Installs"], 95))
print("99th Percentile =", np.percentile(df["Installs"], 99))
```

```
10th Percentile = 100.0
25th Percentile = 1000.0
50th Percentile = 100000.0
70th Percentile = 1000000.0
90th Percentile = 10000000.0
95th Percentile = 10000000.0
99th Percentile = 100000000.0
```

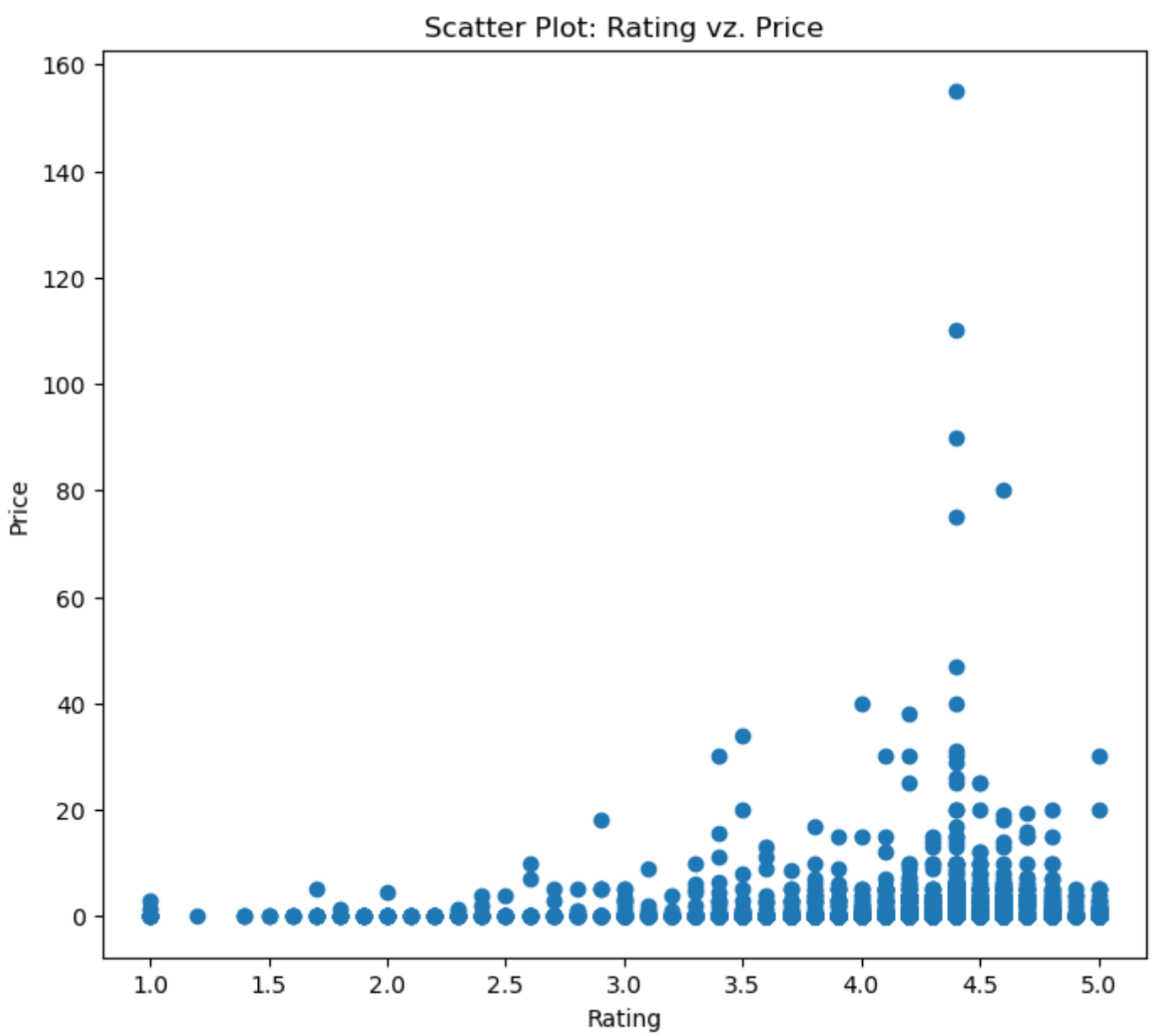
6.3.2) Finding the cuttoff threshold for installs. In addition to the above: IQR = 999000.0 Upper Fence = 2498500.0 The cutoff threshold for installs is set at the upper fence value of 2,498,500.0 using Tukey's fence method.

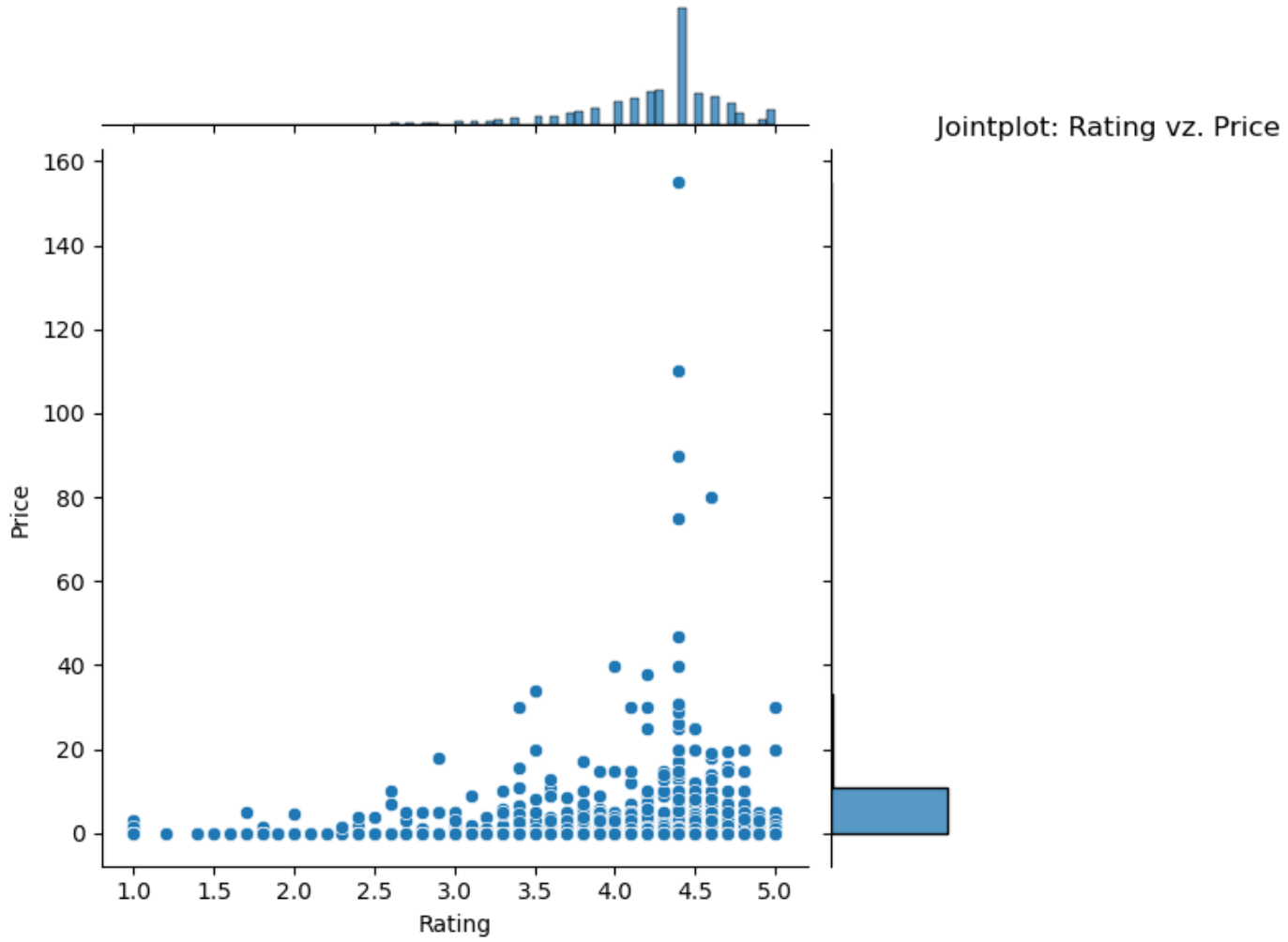
In [56]:
```
(df["Installs"] > 2498500.0).value_counts()
#2188 records will be dropped.
```

Out[56]:
```
False    7751
True     2158
Name: Installs, dtype: int64
```

In [57]:
```
df.drop(df[df["Installs"] > 2498500.0].index, inplace=True)
#6.3.2dropping values by consider upper fence value as threshold. 27.84% of the current
```

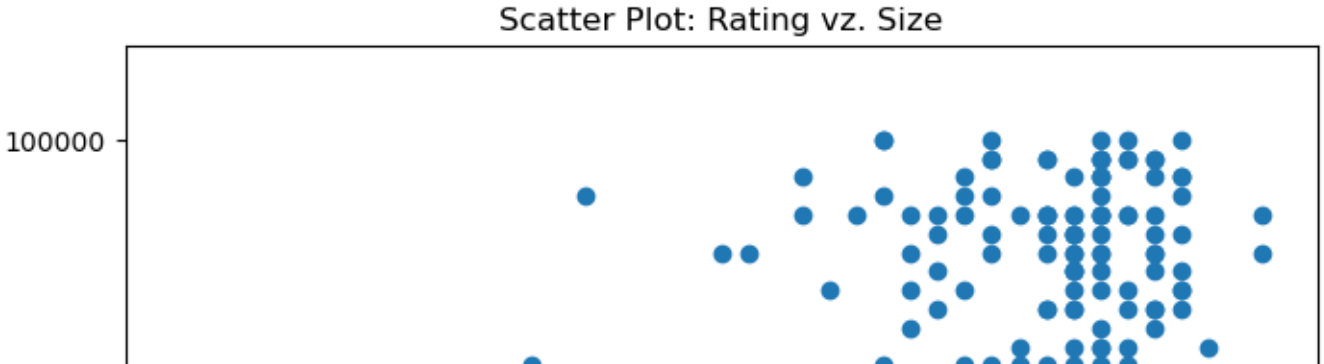
In [58]:
```
#Q)7.1.1 scatter plot/joinplot for Rating vs. Price
plt.figure(figsize=(8,7))
plt.scatter(x=df["Rating"],y=df["Price"])
plt.xlabel("Rating")
plt.ylabel("Price")
plt.title("Scatter Plot: Rating vz. Price")
plt.show()



sns.jointplot(x=df["Rating"], y=df["Price"])
plt.title("                                                Jointplot: Rating vz. Price
plt.show()
```

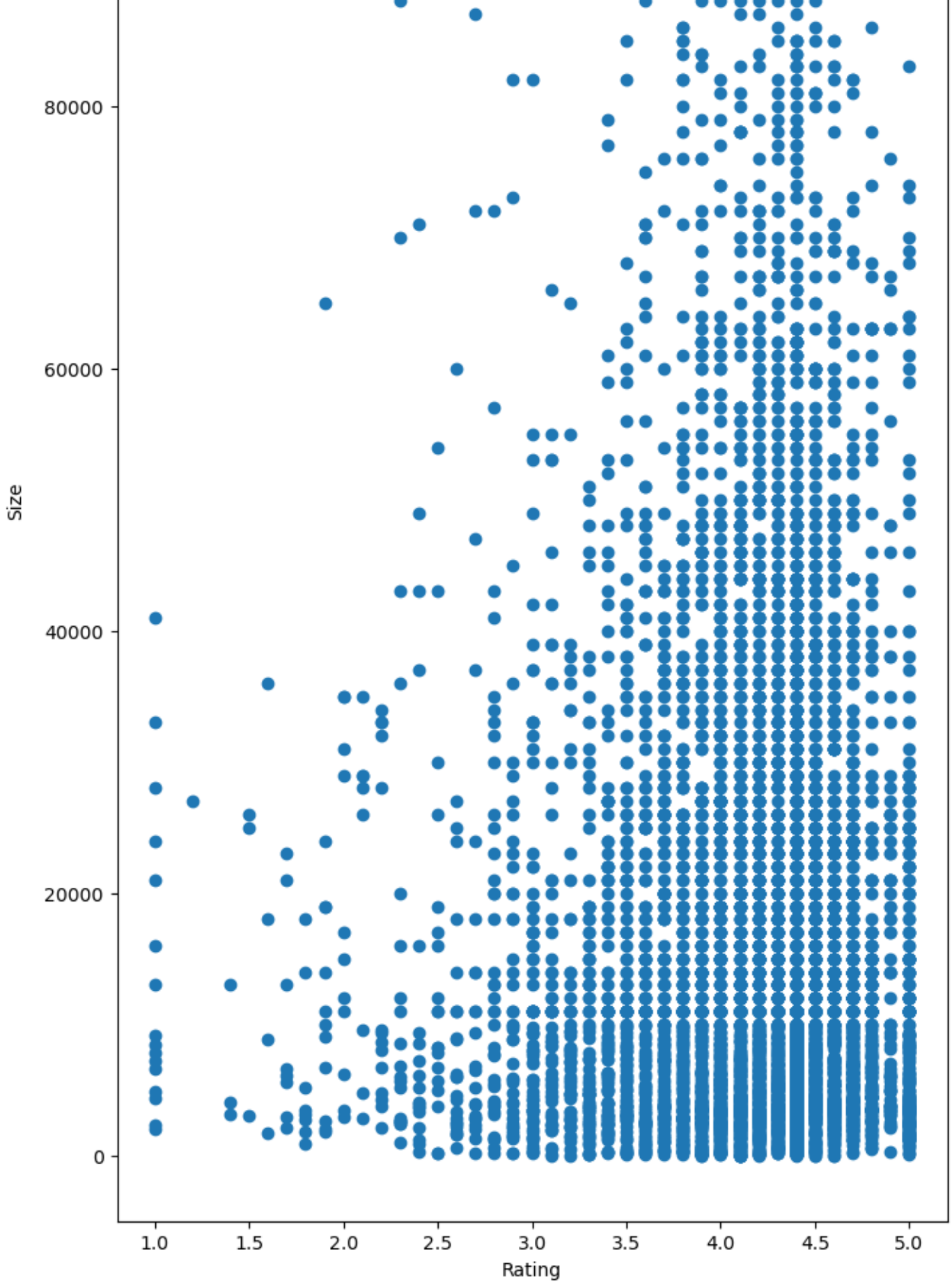

Scatter Plot: Rating vz. Price
Price
Rating
160
140
120
100
80
60
40
20
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

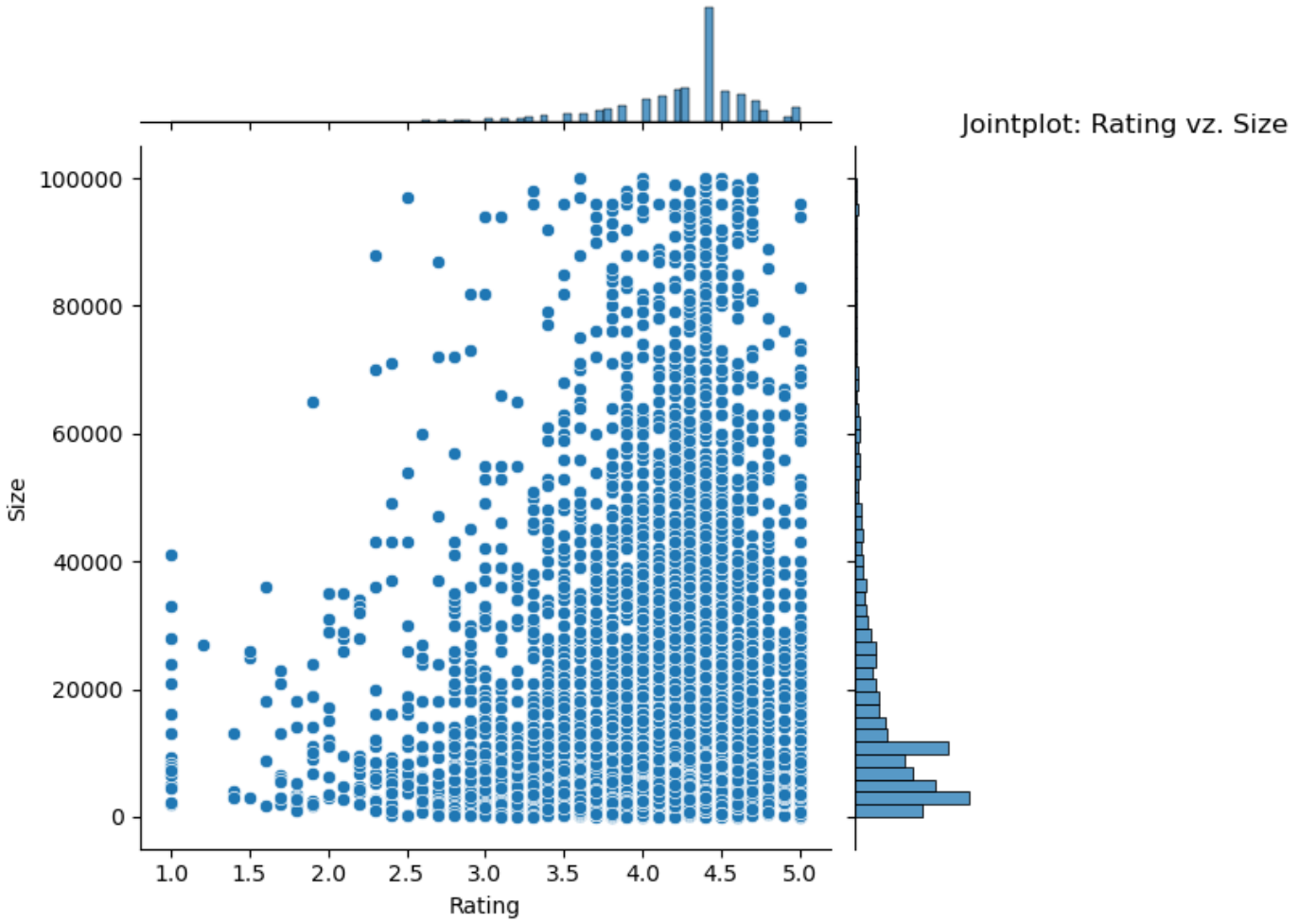


Q)7.1.1 What pattern do you observe? Does rating increase with price? Scatterplot/joinplot observation of Rating vz Price: From the scatterplot and the bin plot from jointplot we can see that, 1) Apps with ratings around 4.3-4.4 tend to have the highest prices, ranging from approximately 0 to 158. Additionally, this rating bin (4.3-4.4) contains the largest number of apps. 2) The majority of apps are priced between 0 and 10, while the remaining apps are dispersed between 10 and 40. There are only a few apps (around 6) priced above 40. Conclusion: Although there is a trend suggesting that higher-rated apps may have higher prices in some cases, it's important to note that when comparing the price ranges of 4.4 and 5.0 ratings, the apps with a rating of 4.4 tend to have higher prices. Therefore, it is not necessary to conclude that apps with higher ratings are always more expensive. Thus, the observation indicates that rating does not necessarily increase with price.

In [59]:
```
#Q)7.2.1 scatter plot/joinplot for Rating vs. Size
plt.figure(figsize=(8,14))
plt.scatter(x=df["Rating"],y=df["Size"])
plt.xlabel("Rating")
plt.ylabel("Size")
plt.title("Scatter Plot: Rating vz. Size")
plt.show()
#------------------------jointplot--------------------------------
sns.jointplot(x=df["Rating"], y=df["Size"])
plt.title("                                                Jointplot: Rating vz. Size"
plt.show()
```

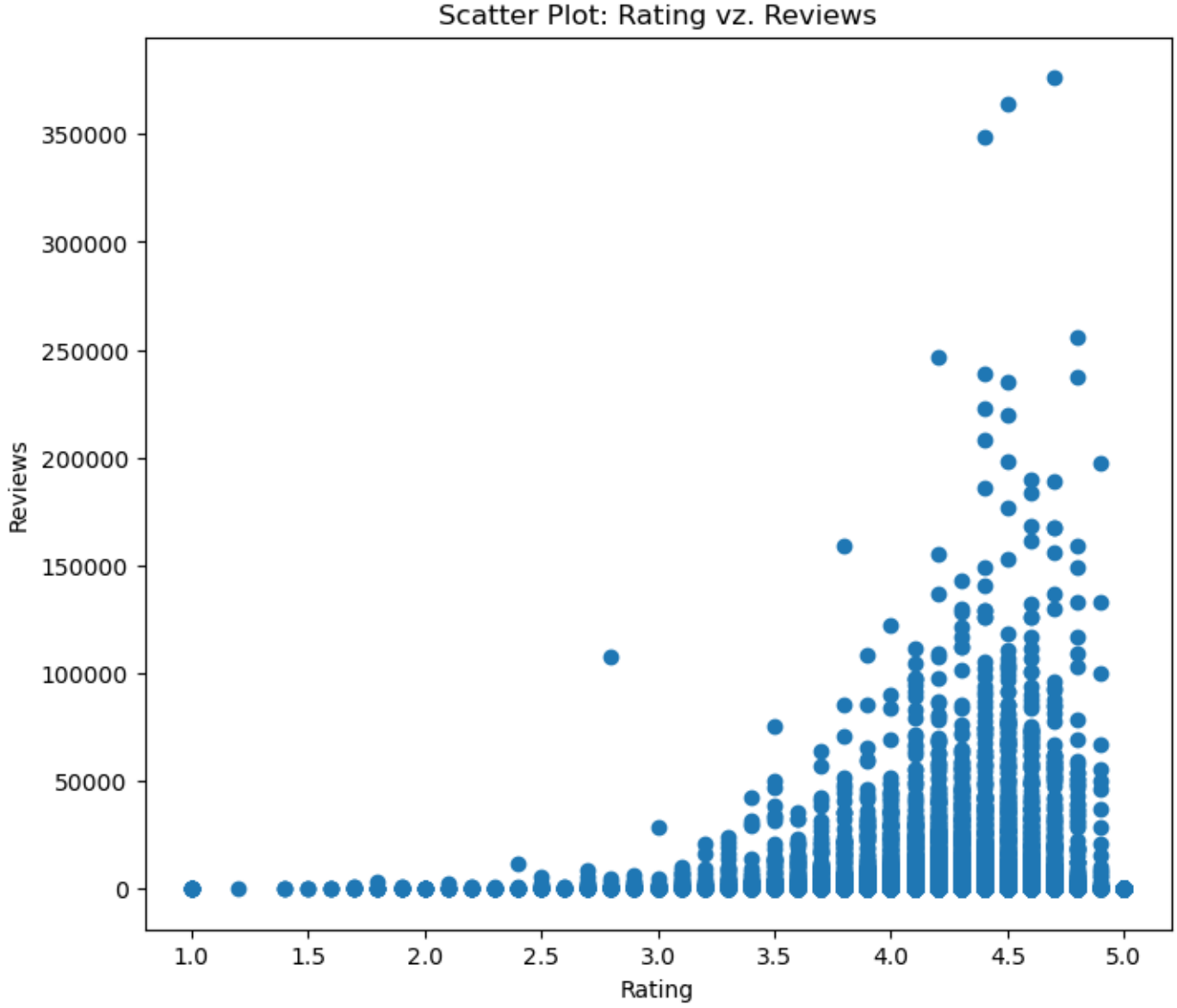


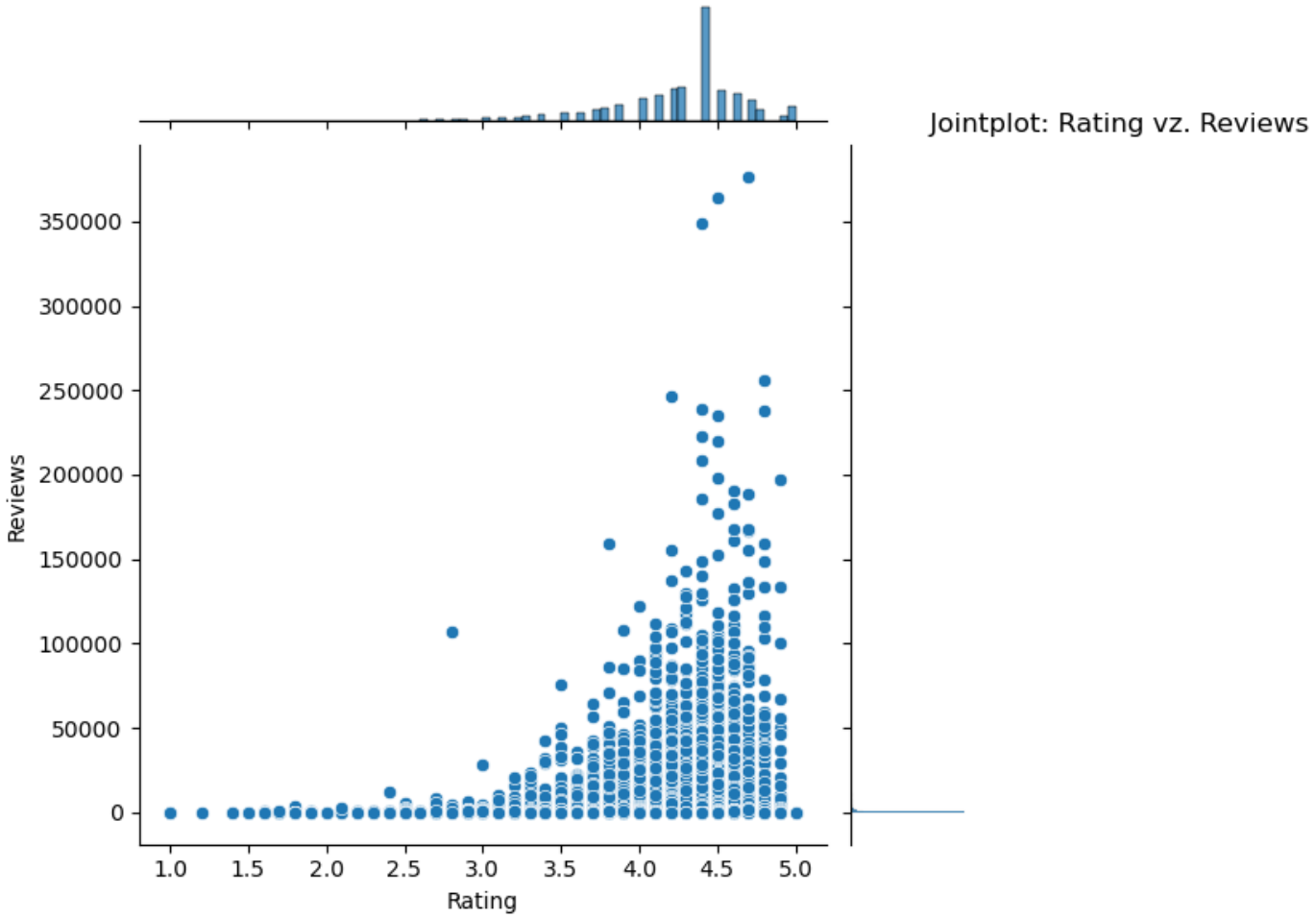

80000
60000
40000
20000
0
Size
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Rating



Q)7.2.1 Are heavier apps rated better? Scatterplot/joinplot observation of Rating vz Size: From the scatterplot and the bin plot from jointplot we can see that, 1) The majority of apps fall into the size range of 0-10,000 kb, while the remaining apps are dispersed between 10,000-100,000 kb. The count of apps reduces as the size increases, indicating that there are fewer apps with larger sizes compared to smaller ones. 2) Apps with ratings of 4.3-4.4 have the highest count, and they exhibit a wide range of app sizes, spanning from 0-100,000 kb. This rating bin (4.3-4.4) contains the largest number of apps. Conclusion: Although heavier apps are more common in higher ratings, it's important to note that lighter apps are also prevalent in the majority. The presence of heavier apps is relatively less compared to lighter apps in higher ratings. While the majority of heavier apps receive better ratings, not all of them do; some have average ratings. Therefore, it is not appropriate to conclude that heavier apps are consistently rated better than lighter apps. This is because there are more lighter apps with higher ratings than heavier apps with high ratings.

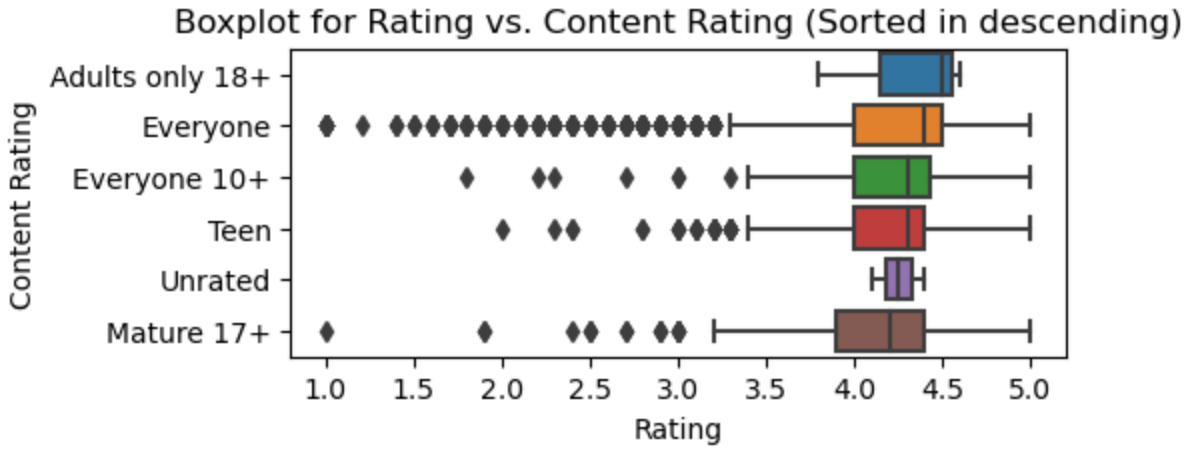
In [60]:
```
#Q)7.3 scatter plot/joinplot for Rating vs. Reviews
plt.figure(figsize=(8,7))
plt.scatter(x=df["Rating"],y=df["Reviews"])
plt.xlabel("Rating")
plt.ylabel("Reviews")
plt.title("Scatter Plot: Rating vz. Reviews")
plt.show()
#------------------------jointplot---------------------------------
sns.jointplot(x=df["Rating"], y=df["Reviews"])
plt.title("                                                Jointplot: Rating vz. Revie
plt.show()
```


Scatter Plot: Rating vz. Reviews
Reviews
Rating
0
50000
100000
150000
200000
250000
300000
350000
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0



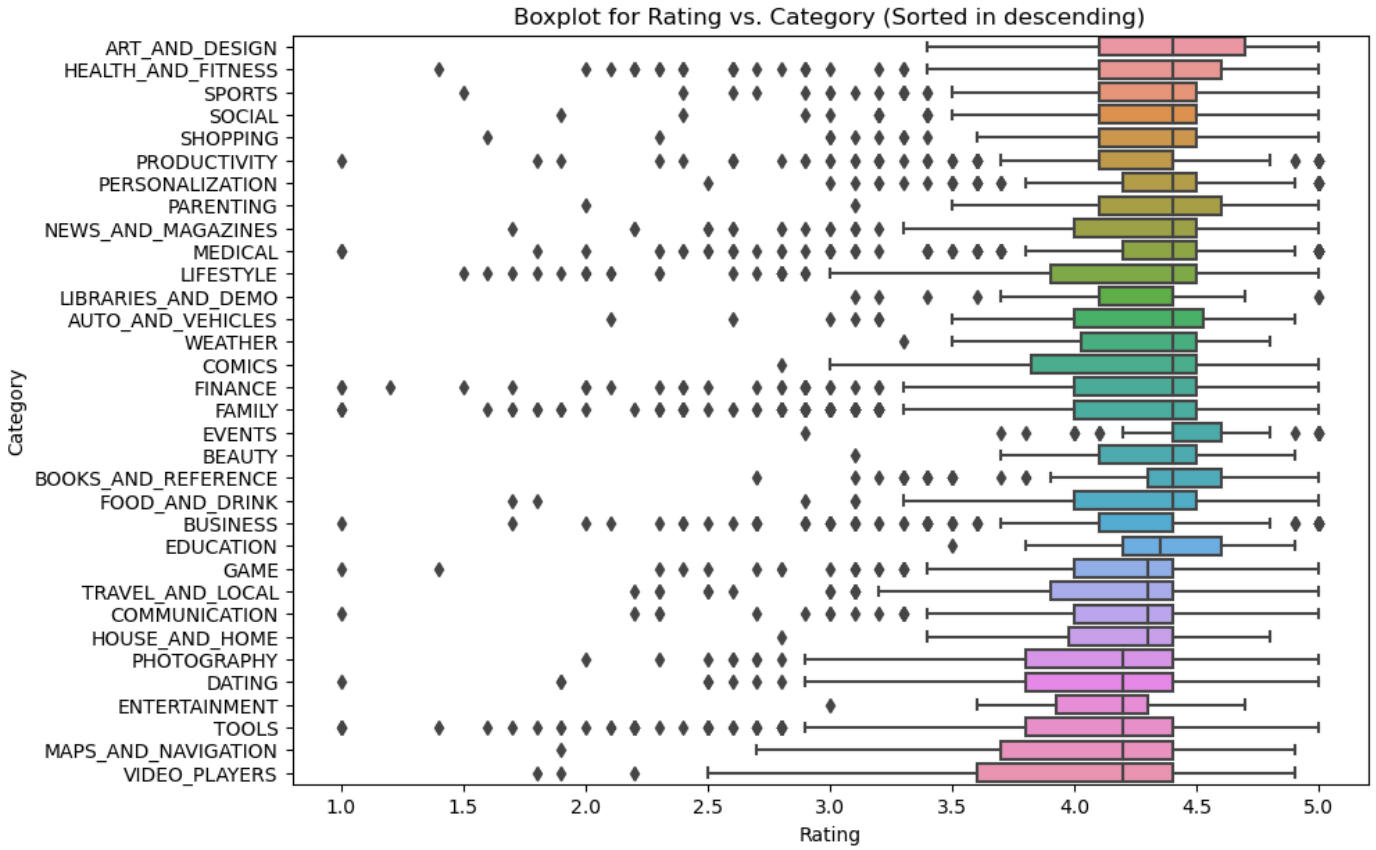
Q) 7.3.1 Does more review mean a better rating always? Scatterplot/joinplot observation of Rating vz Reviews: From the scatterplot and the bin plot from jointplot we can see that, 1) Apps with 0 reviews are uniformly distributed across the rating range of 1 to 5, indicating that there is no specific correlation between the absence of reviews and the app's rating. 2) While the majority of apps with higher reviews tend to have higher ratings, it is important to note that apps with 0 reviews also have higher ratings. In fact, the count of apps with 0 reviews and higher ratings is greater than the count of apps with high reviews and higher ratings. Conclusion: It is evident that having more reviews generally contributes to a better rating, but this relationship is not true in all cases. There are instances where apps with fewer reviews still manage to receive better ratings in greater numbers. Therefore, while more reviews typically aid in achieving a higher rating, it is not a guarantee. Other factors may influence the rating of an app.

In [61]:
```
#7.4) boxplot for Rating vs. Content Rating (sorted)
plt.figure(figsize=(5,2))
sorted_categories = df.groupby("Content Rating")["Rating"].median().sort_values(ascendin
sns.boxplot(x=df["Rating"],y=df["Content Rating"],order =sorted_categories)
plt.title("Boxplot for Rating vs. Content Rating (Sorted in descending)")
plt.show()
```



7.4.1. Is there any difference in the ratings? Are some types liked better? 1) There are notable differences in ratings across all content categories. Ranking the categories based on the median, we have: Adult only 18+ > Everyone > Everyone 10+ > Teen > Unrated > Mature 17+. 2) "Adult only 18+" content category stands out as it is more favored compared to other content types. It has a higher median rating and a smaller interquartile range (IQR) when compared to the "Everyone" category.

In [62]:
```
#7.5) boxplot for Ratings vs. Category (sorted)
plt.figure(figsize=(10,7))
sorted_categories2 = df.groupby("Category")["Rating"].median().sort_values(ascending=Fal
sns.boxplot(x=df["Rating"],y=df["Category"], order =sorted_categories2)
plt.title("Boxplot for Rating vs. Category (Sorted in descending)")
plt.show()
```



In [82]:
```
sorted_genr = df.groupby("Genres")["Rating"].median().sort_values(ascending=False).index
print("Genres sorted based on rating in Descending order:- \n\n", sorted_genr)
```

```
Genres sorted based on rating in Descending order:-

 Index(['Comics;Creativity', 'Board;Pretend Play', 'Health & Fitness;Education',
       'Casual;Creativity', 'Art & Design;Creativity',
       'Role Playing;Action & Adventure', 'Strategy;Action & Adventure',
       'Puzzle;Education', 'Adventure;Brain Games', 'Arcade;Pretend Play',
       ...
       'Travel & Local;Action & Adventure', 'Education;Music & Video',
       'Board;Action & Adventure', 'Educational', 'Educational;Creativity',
       'Role Playing;Pretend Play', 'Health & Fitness;Action & Adventure',
       'Art & Design;Pretend Play', 'Entertainment;Pretend Play',
       'Parenting;Brain Games'],
      dtype='object', name='Genres', length=112)
```

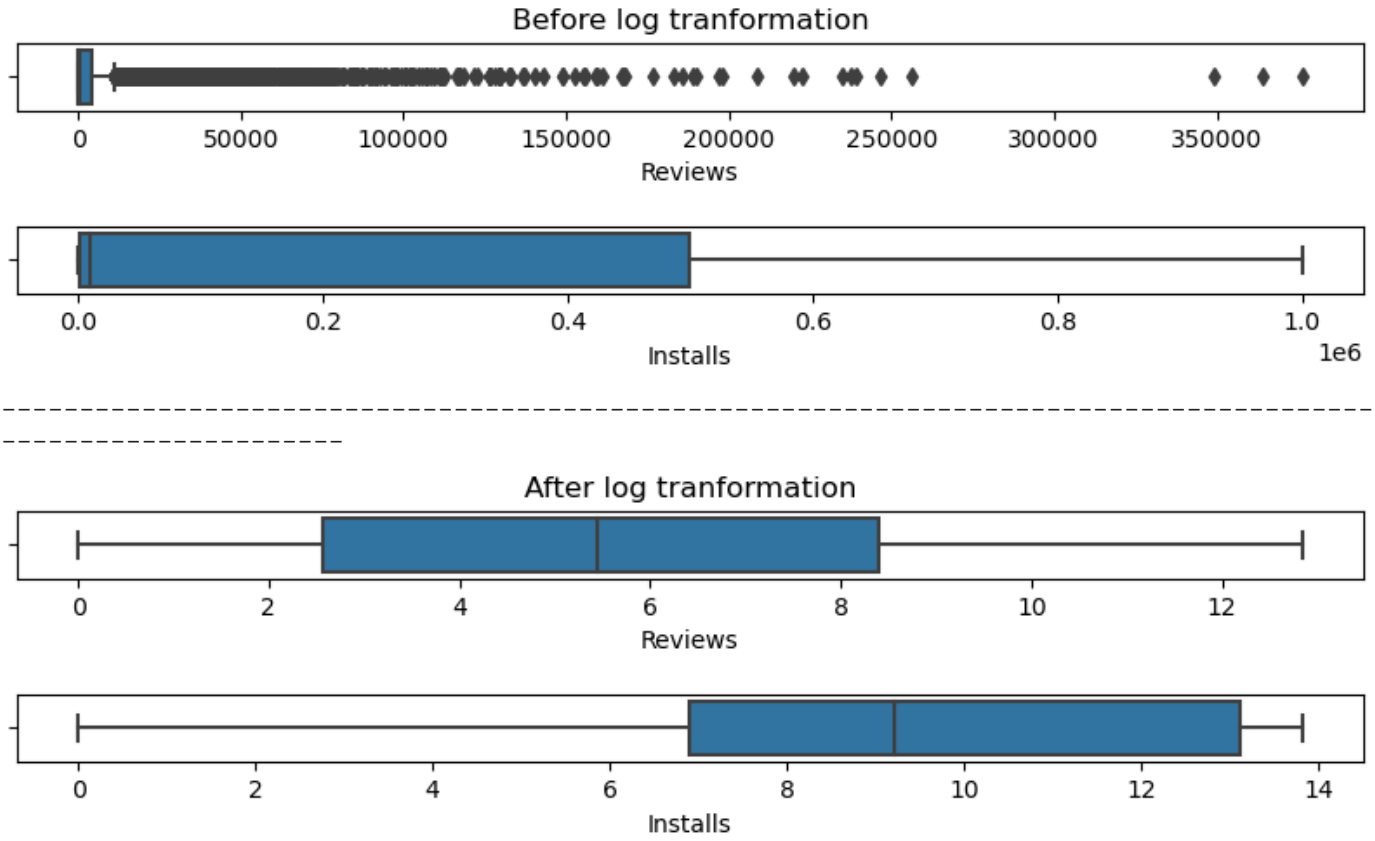
7.5.1) Which genre has the best ratings? 1) The genre "Comics;Creativity and Board;Pretend Play" has the highest ratings among the genres considered. It showcases consistently positive ratings across various metrics. 2) From the boxplot, it is evident that the category "ART and DESIGN" has the best ratings. It demonstrates higher values in the upper quartiles compared to other genres and boasts the highest maximum rating value, indicating its strong performance.

In [254…
```
#8)copy of the dataframe
inp1 = df.copy()
```

In [255…
```
#8.1) log transformation applied to Reviews & Installs to reduce the skew for linear reg
```

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

In [256…
```
#boxplot of before and after comparison
plt.figure(figsize=(10,.5))
sns.boxplot(x=df["Reviews"])
plt.title("Before log tranformation")
plt.figure(figsize=(10,.5))
sns.boxplot(x=df["Installs"])
plt.show()
print("----------------------------------------------------------------------------------------")
plt.figure(figsize=(10,.5))
sns.boxplot(x=inp1["Reviews"])
plt.title("After log tranformation")
plt.figure(figsize=(10,.5))
sns.boxplot(x=inp1["Installs"])
plt.show()
```




```
----------------------------------------------------------------------------------------
----------------------
```


In [257…
```
#8.2) Dropping columns App, Last Updated, Current Ver, and Android Ver. These variables
inp1.drop(["App","Last Updated","Android Ver","Current Ver","Type"],axis =1, inplace =Tr
```

In [258…
```
#8.3) Performing one-hot encoding : Getting dummy columns for Category, Genres, and Cont
inp2 = pd.get_dummies(inp1)
```

In [259…
```
inp2.head()
```

Out[259]:

	Rating	Reviews	Installs	Price	Size	Category_ART_AND_DESIGN	Category_AUTO_AND_VEHICLES	Categ
0	4.1	5.075174	9.210440	0.0	19000.0	1	0	
1	3.9	6.875232	13.122365	0.0	14000.0	1	0	
4	4.3	6.875232	11.512935	0.0	2800.0	1	0	

5	4.4	5.123964	10.819798	0.0	5600.0	1	0
6	3.8	5.187386	10.819798	0.0	19000.0	1	0

5 rows × 156 columns

In [260…
```
#x =inp2.drop(["Rating"],axis =1)
#y =inp2["Rating"]
#inputs for alternate method
```

In [270…
```
#9) Train test split  and apply 70-30 split. Name the new dataframes df_train and df_tes
df_train, df_test = train_test_split(inp2, test_size=0.3, random_state=100)
#aletrnate method - x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,r
```

In [272…
```
#9)Separate the dataframes into X_train, y_train, X_test, and y_test.
x_train = df_train.drop(["Rating"],axis =1)
x_test = df_test.drop(["Rating"],axis =1)
y_train = df_train["Rating"]
y_test = df_test["Rating"]
#splitting the data into training and testing sets for both the input features (x) and t
```

In [273…
```
#11) Model Building - Using linear regression model
model = LinearRegression()
model.fit(x_train,y_train)
#training the model on train set
```

Out[273]:
```
LinearRegression()
```

In [274…
```
pred_train = model.predict(x_train)
#using x train set for model prediction, this value is required for calculating R2 score
```

In [275…
```
#11) Report the R2 on the train set
print ("R2 on train set:", r2_score(y_train,pred_train))
#The R2 score on train set is poor 0.11. The model fit is poor, so we can expect model p
```

```
R2 on train set: 0.11647129738666806
```

In [276…
```
#12) Make predictions on test set
pred_test = model.predict(x_test)
#using x test set for model prediction
```

In [277…
```
#12) R2 of prediction model
print ("R2 on test set:",r2_score(y_test,pred_test))
#Poor R2 score of 0.07.
```

```
R2 on test set: 0.07213891304910547
```

Final Observations: The R2 score of the training set is 0.116, indicating that approximately 11.65% of the variance in the target variable can be explained by the predictor variables in the model. This suggests a relatively weak relationship between the predictor variables and the target variable in the training data. Similarly, the R2 score of the test set is 0.072, indicating that approximately 7.21% of the variance in the target variable can be explained by the predictor variables in the model when applied to unseen test data. These results suggest that the model's performance on the test data is consistent with its performance on the training data, but still reflects a relatively weak predictive ability. Considering the correlation coefficients between the predictor variables and the target variable, none of the variables show a strong correlation. Therefore, it would be more appropriate to consider either using a different dataset or exploring alternative modeling techniques and evaluation metrics. Additionally, the correlation coefficient values for the target variable "Rating" are provided below: Correlation coefficients for target variable "Rating": 1.000000 -0.028403 -0.119659 0.028496 0.002698 0.028610 0.004383 0.017988 0.047664 0.009433 ... 0.004496 0.006606 -0.080340 -0.025298 -0.001833 -0.006362 0.004496 -0.045823 0.010178 0.007054

In [279…
```
inp2.corr()
```

Out[279]:

	Rating	Reviews	Installs	Price	Size	Category_ART_AND_DESIGN	Categor

Rating	1.000000	-0.028403	-0.119659	0.028496	0.002698	0.028610
Reviews	-0.028403	1.000000	0.937648	-0.055021	0.213612	0.011504
Installs	-0.119659	0.937648	1.000000	-0.109818	0.183302	0.028140
Price	0.028496	-0.055021	-0.109818	1.000000	0.007408	-0.009374
Size	0.002698	0.213612	0.183302	0.007408	1.000000	-0.027922
...	...	...	...	...	...	...
Genres_Trivia	-0.006362	-0.006090	-0.008652	-0.009464	0.004185	-0.005611
Genres_Trivia;Education	0.004496	-0.010983	-0.014645	-0.001620	0.004972	-0.000960
Genres_Video Players & Editors	-0.045823	0.020600	0.031433	-0.014054	-0.027069	-0.010282
Genres_Weather	0.010178	0.053528	0.050303	0.001792	-0.030176	-0.007340
Genres_Word	0.007054	0.014703	0.007504	-0.005614	0.011663	-0.003329

156 rows × 156 columns

In [ ]: