In [148…]
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [149…]
```
data= pd.read_csv('googleplaystore.csv')
```

In [150…]
```
data.shape
```

Out[150]:
```
(10841, 13)
```

In [151…]
```
data.head()
```

Out[151]:

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

In [152…]
```
data.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 [153…]
```
# Check for null values in the data. Get the number of null values for each column.
data.isnull().sum ()
```

Out[153]:
```
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 [154…]
```
# 3. Drop records with nulls in any of the columns.
data = data.dropna()
```

In [155…]
```
data.shape
```

Out[155]:
```
(9360, 13)
```

In [156…]
```
# Drop records with nulls in any of the columns check
data.isnull().sum ()
```

Out[156]:
```
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 [157…
```
data= data[data.Size != 'Varies with device']
```

In [158…
```
# 4.1 Size column has sizes in Kb as well as Mb. To analyze, you'll need to convert th
# 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 [159…
```
data.Size = data.Size.apply(MtoK)
```

In [160…
```
data.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 [161…
```
data.size
```

Out[161]:
```
100399
```

In [162…
```
data = data[data.Size != 'Varies with device']
```

In [163…
```
data.shape
```

Out[163]:
```
(7723, 13)
```

In [164…
```
# 4.2 Reviews is a numeric field that is loaded as a string field. Convert it to numer
#converts to int

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

In [165…
```
data ["Reviews"].dtype
```

Out[165]:
```
dtype('int64')
```

In [166…
```
""" 4.3 Installs field is currently stored as string and has values like 1,000,000+.
Treat 1,000,000+ as 1,000,000
remove ‘+’, ‘,’ from the field, convert it to integer"""

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

In [167…
```
data.Installs = data.Installs.map (remove_char)
```

In [168…
```
data.Installs
```

Out[168]:
```
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 [169…
```
# 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 [170…
```
data.Price = data.Price.apply(remove_symbol)
```

In [171…
```
data ["Price"].dtype
```

Out[171]:
```
dtype('float64')
```

In [172…
```
data.Price
```

Out[172]:
```
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 [173…
```
data.shape
```

Out[173]:
```
(7723, 13)
```

In [174…
```
"""5.1 Sanity checks:
Average rating should be between 1 and 5 as only these values are allowed on the play
Drop the rows that have a value outside this range.
"""
```

Out[174]:
```
'5.1 Sanity checks:\nAverage rating should be between 1 and 5 as only these values ar
e allowed on the play store. \nDrop the rows that have a value outside this range.\n'
```

In [175…
```
#We can check with below formula
data.loc[data.Rating < 1] & data.loc[data.Rating > 5]
# Result : no rating is out side the range
```

Out[175]:

| App | Category | Rating | Reviews | Size | Installs | Type | Price | Content Rating | Genres | Last Updated | Current Ver | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

In [176…
```
#5.2 """Reviews should not be more than installs as only those who installed can revie
```

In [177…
```
data.loc[data.Reviews > data.Installs]
```

Out[177]:

| | App | Category | Rating | Reviews | Size | Installs | Type | Price | Content Rating | Genres | Las Updated |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **2454** | KBA-EZ Health Guide | MEDICAL | 5.0 | 4 | 25000.0 | 1 | Free | 0.00 | Everyone | Medical | Augus 2, 201 |
| **5917** | Ra Ga Ba | GAME | 5.0 | 2 | 20000.0 | 1 | Paid | 1.49 | Everyone | Arcade | Februar 8, 201 |
| **6700** | Brick Breaker BR | GAME | 5.0 | 7 | 19000.0 | 5 | Free | 0.00 | Everyone | Arcade | July 23 201 |
| **7402** | Trovami se ci riesci | GAME | 5.0 | 11 | 6100.0 | 10 | Free | 0.00 | Everyone | Arcade | March 11, 201 |
| **8591** | DN Blog | SOCIAL | 5.0 | 20 | 4200.0 | 10 | Free | 0.00 | Teen | Social | July 23 201 |
| **10697** | Mu.F.O. | GAME | 5.0 | 2 | 16000.0 | 1 | Paid | 0.99 | Everyone | Arcade | March 3 201 |

In [178…

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

Out[178]:

| | 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 [179…

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

Out[179]:

| | App | Category | Rating | Reviews | Size | Installs | Type | Price | Content Rating | Genres | Las Updated |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **2454** | KBA-EZ Health Guide | MEDICAL | 5.0 | 4 | 25000.0 | 1 | Free | 0.00 | Everyone | Medical | Augus 2, 201 |
| **5917** | Ra Ga Ba | GAME | 5.0 | 2 | 20000.0 | 1 | Paid | 1.49 | Everyone | Arcade | Februar 8, 201 |
| **6700** | Brick Breaker BR | GAME | 5.0 | 7 | 19000.0 | 5 | Free | 0.00 | Everyone | Arcade | July 23 201 |
| **7402** | Trovami se ci riesci | GAME | 5.0 | 11 | 6100.0 | 10 | Free | 0.00 | Everyone | Arcade | Marc 11, 201 |
| **8591** | DN Blog | SOCIAL | 5.0 | 20 | 4200.0 | 10 | Free | 0.00 | Teen | Social | July 23 201 |
| **10697** | Mu.F.O. | GAME | 5.0 | 2 | 16000.0 | 1 | Paid | 0.99 | Everyone | Arcade | March 3 201 |

In [180…
```
data.columns
```

Out[180]:
```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type',
       'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver',
       'Android Ver'],
      dtype='object')
```

In [181…
```
data[['Reviews','Installs']]
```

Out[181]:

| | 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 [182…
```
# Created a column to easily identify results for the syntax
data['RAI'] = np.where((data['Reviews'] <= data['Installs']), data['Installs'], np.nar
```

In [183…
```
data['RAI'].shape
```

Out[183]:
```
(7723,)
```

In [184…
```
data['RAI'].describe()
```

Out[184]:
```
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 [185…
```
data = data.dropna()
```

In [186…
```
# Still have 14 columns and need to drop RAI
data.columns
```

Out[186]:
```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type',
       'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver',
       'Android Ver', 'RAI'],
      dtype='object')
```

In [187…
```
# to drop RAI column
data = data.drop(['RAI'], axis = 1)
```

In [188…
```
data.shape
```

Out[188]:
```
(7717, 13)
```

In [189…
```
data.columns
```

Out[189]:
```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type',
       'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver',
       'Android Ver'],
      dtype='object')
```

In [190…
```
# 5.3 For free apps (type = “Free”), the price should not be >0. Drop any such rows.
```

In [191…
```
# To determine the rows with Price above $0
data.loc[data.Price > 0]
```

Out[191]:

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

575 rows × 13 columns

In [192…

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

Out[192]:

| | App | Category | Rating | Reviews | Size | Installs | Type | Price | Content Rating | Genres | Last Updated | Current Ver | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

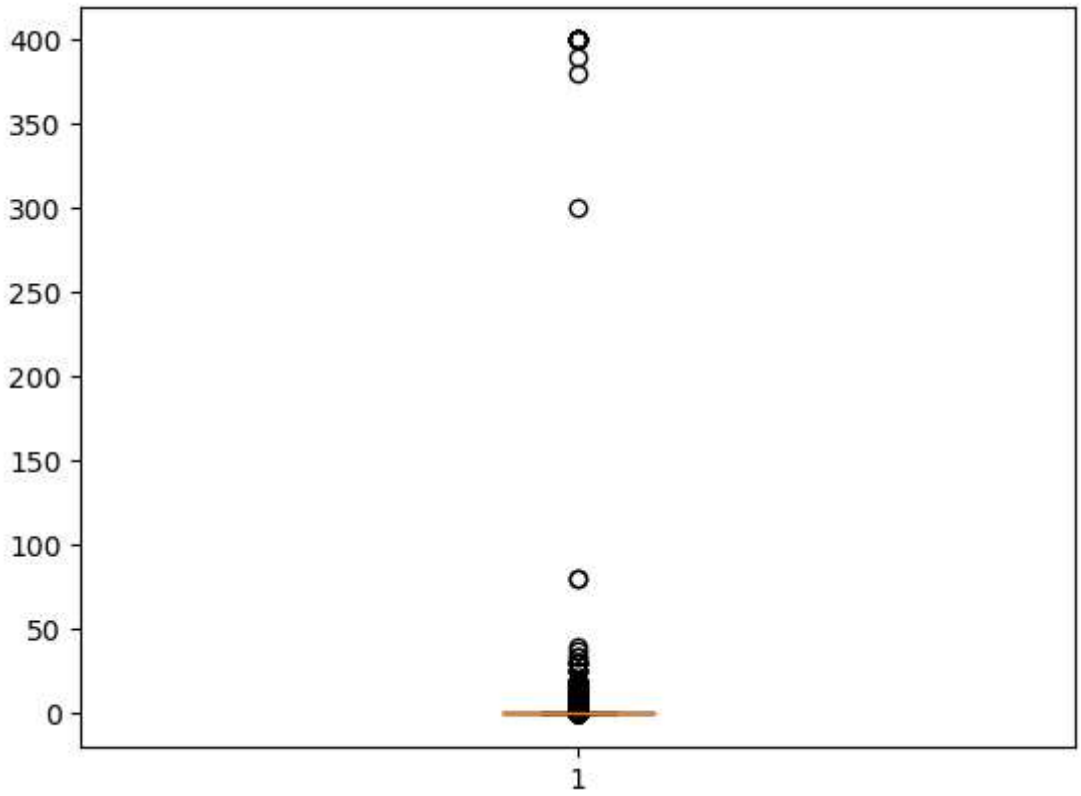
In [193…

```
#Performing univariate analysis:
#Boxplot for Price
```

```
#Are there any outliers? Think about the price of usual apps on Play Store.
```

In [194…
```
plt.boxplot(data['Price'])
# most app prices are less that $100
# some outlier prices above $200
```

Out[194]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x1111f9c8a60>,
  <matplotlib.lines.Line2D at 0x1111f9c8d00>],
 'caps': [<matplotlib.lines.Line2D at 0x1111f9c8fa0>,
  <matplotlib.lines.Line2D at 0x1111f9c9240>],
 'boxes': [<matplotlib.lines.Line2D at 0x1111f9c87c0>],
 'medians': [<matplotlib.lines.Line2D at 0x1111f9c94e0>],
 'fliers': [<matplotlib.lines.Line2D at 0x1111f9c9780>],
 'means': []}
```



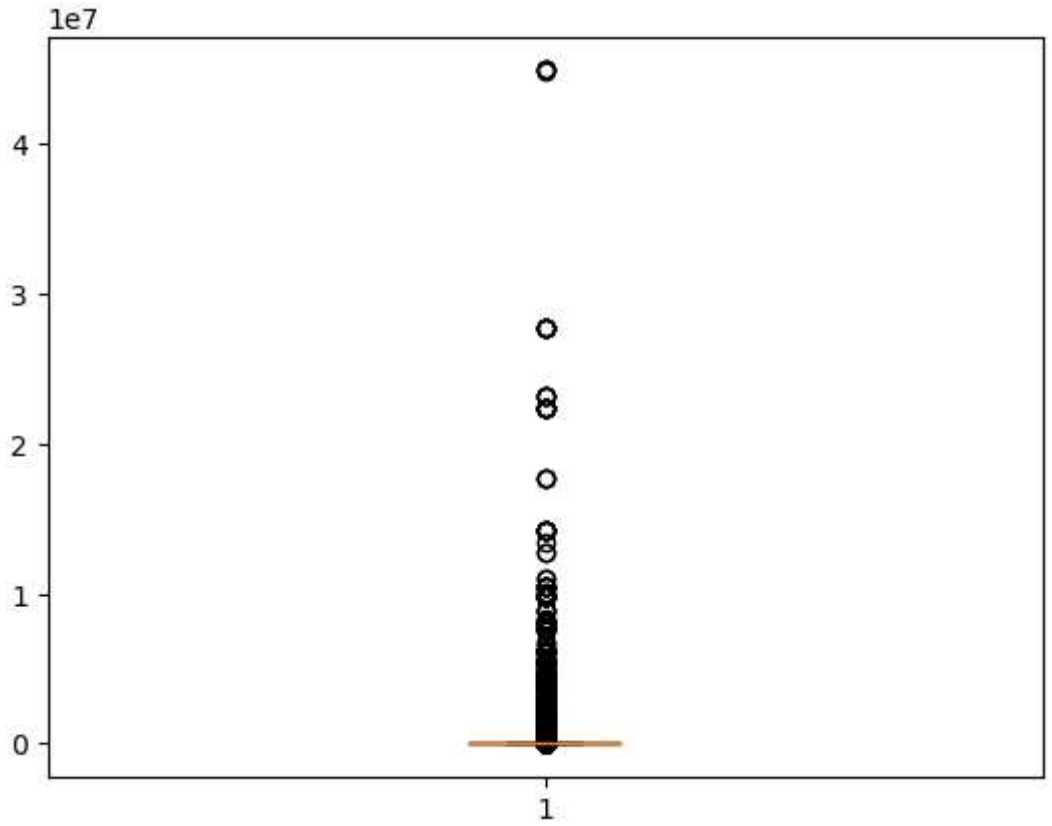

In [195…
```
data['Price'].describe()
```

Out[195]:
```
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 [196…
```
#Boxplot for Reviews
#Are there any apps with very high number of reviews? Do the values seem right?

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

Out[196]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x1111fb7d390>,
  <matplotlib.lines.Line2D at 0x1111fb7d630>],
 'caps': [<matplotlib.lines.Line2D at 0x1111fa1dc60>,
  <matplotlib.lines.Line2D at 0x1111fb7da50>],
 'boxes': [<matplotlib.lines.Line2D at 0x1111fb7d0f0>],
 'medians': [<matplotlib.lines.Line2D at 0x1111fb7dcf0>],
 'fliers': [<matplotlib.lines.Line2D at 0x1111fb7df90>],
 'means': []}
```



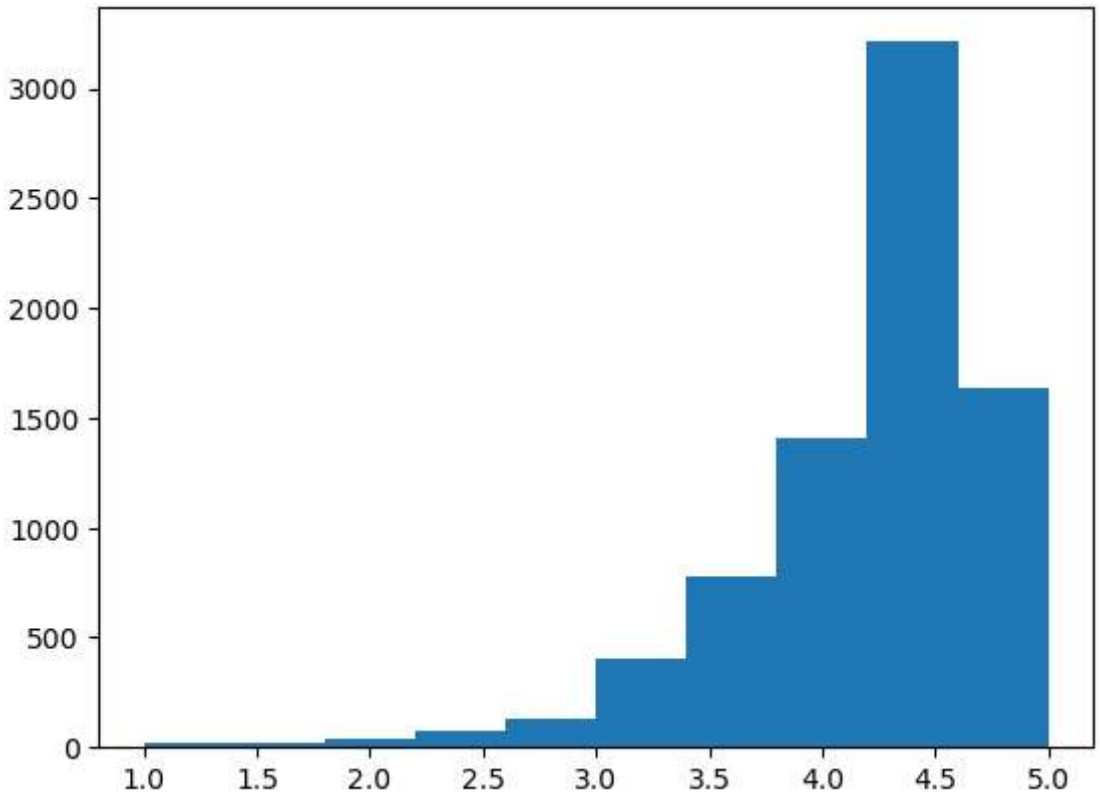

In [197…
```
#Histogram for Rating
#How are the ratings distributed? Is it more toward higher ratings?

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

Out[197]:
```
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
```

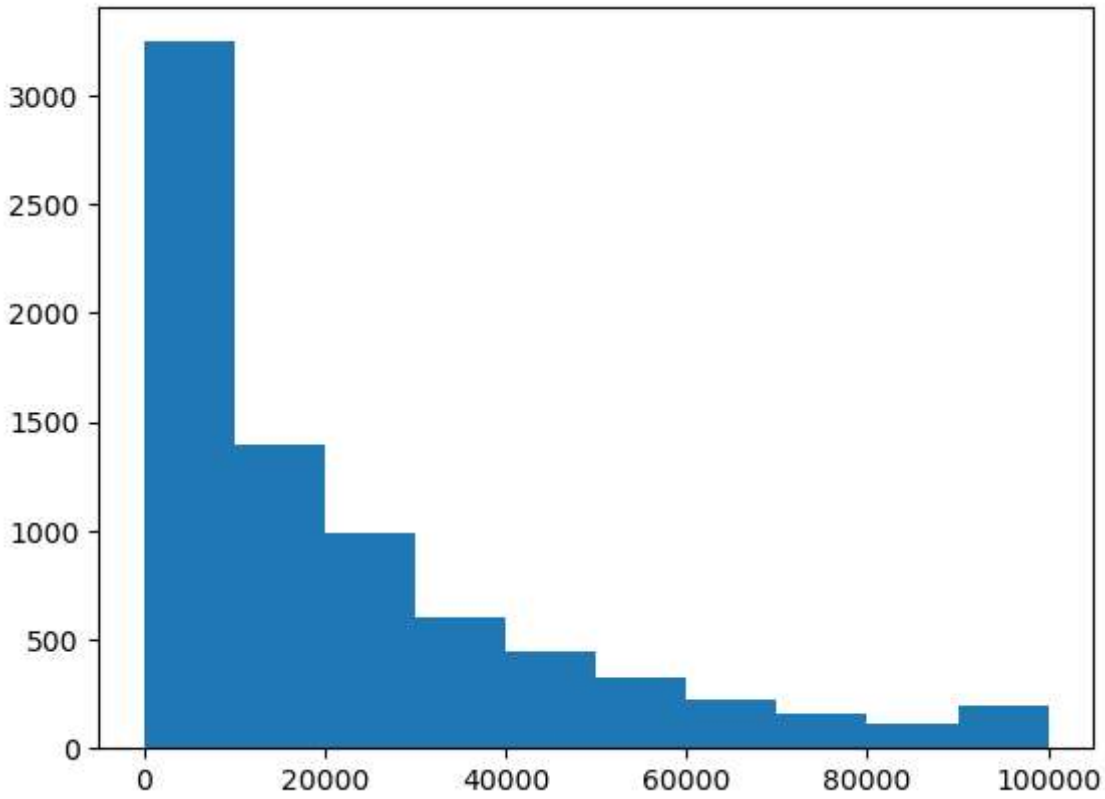
In [198…
```
plt.hist(data['Rating'])
```

Out[198]:
```
(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 [199…
```
plt.hist(data['Size'])
```

Out[199]:
```
(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 [200…

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

Out[200]:

| | App | Category | Rating | Reviews | Size | Installs | Type | Price | Content Rating | Genr |
|---|---|---|---|---|---|---|---|---|---|---|
| **4197** | most expensive app (H) | FAMILY | 4.3 | 6 | 1500.0 | 100 | Paid | 399.99 | Everyone | Entertainme |
| **4362** | 💎 I'm rich | LIFESTYLE | 3.8 | 718 | 26000.0 | 10000 | Paid | 399.99 | Everyone | Lifesty |
| **4367** | I'm Rich - Trump Edition | LIFESTYLE | 3.6 | 275 | 7300.0 | 10000 | Paid | 400.00 | Everyone | Lifesty |
| **5351** | I am rich | LIFESTYLE | 3.8 | 3547 | 1800.0 | 100000 | Paid | 399.99 | Everyone | Lifesty |
| **5354** | I am Rich Plus | FAMILY | 4.0 | 856 | 8700.0 | 10000 | Paid | 399.99 | Everyone | Entertainme |
| **5355** | I am rich VIP | LIFESTYLE | 3.8 | 411 | 2600.0 | 10000 | Paid | 299.99 | Everyone | Lifesty |
| **5356** | I Am Rich Premium | FINANCE | 4.1 | 1867 | 4700.0 | 50000 | Paid | 399.99 | Everyone | Finan |
| **5357** | I am extremely Rich | LIFESTYLE | 2.9 | 41 | 2900.0 | 1000 | Paid | 379.99 | Everyone | Lifesty |
| **5358** | I am Rich! | FINANCE | 3.8 | 93 | 22000.0 | 1000 | Paid | 399.99 | Everyone | Finan |
| **5359** | I am rich(premium) | FINANCE | 3.5 | 472 | 965.0 | 5000 | Paid | 399.99 | Everyone | Finan |
| **5362** | I Am Rich Pro | FAMILY | 4.4 | 201 | 2700.0 | 5000 | Paid | 399.99 | Everyone | Entertainme |
| **5364** | I am rich (Most expensive app) | FINANCE | 4.1 | 129 | 2700.0 | 1000 | Paid | 399.99 | Teen | Finan |
| **5366** | I Am Rich | FAMILY | 3.6 | 217 | 4900.0 | 10000 | Paid | 389.99 | Everyone | Entertainme |
| **5369** | I am Rich | FINANCE | 4.3 | 180 | 3800.0 | 5000 | Paid | 399.99 | Everyone | Finan |
| **5373** | I AM RICH PRO PLUS | FINANCE | 4.0 | 36 | 41000.0 | 1000 | Paid | 399.99 | Everyone | Finan |

In [201…

```
data[data.Price>100].shape
# 15 rows with outlier prices
```

Out[201]:

```
(15, 13)
```

In [202…

```
#data shape before the drop
data.shape
```

Out[202]: (7717, 13)

In [203…
```
#df shape after the drop
data[data.Price <=100].shape
```

Out[203]: (7702, 13)

In [204…
```
# data accommodates all the required conditions on Price, Rating Reviews and Insatlls
data.describe()
```

Out[204]:

| | Rating | Reviews | Size | Installs | Price |
|---|---|---|---|---|---|
| **count** | 7717.000000 | 7.717000e+03 | 7717.000000 | 7.717000e+03 | 7717.000000 |
| **mean** | 4.173293 | 2.951275e+05 | 22976.614293 | 8.430620e+06 | 1.128725 |
| **std** | 0.544362 | 1.864640e+06 | 23456.770600 | 5.017636e+07 | 17.414784 |
| **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 | 2.351000e+03 | 14000.000000 | 1.000000e+05 | 0.000000 |
| **75%** | 4.500000 | 3.910900e+04 | 33000.000000 | 1.000000e+06 | 0.000000 |
| **max** | 5.000000 | 4.489389e+07 | 100000.000000 | 1.000000e+09 | 400.000000 |

In [205…
```
# 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, will skew it.
#Drop records having more than 2 million reviews.
```

In [206…
```
data['Reviews'].describe()
```

Out[206]:
```
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 [207…
```
# rows with Reviews more that 2million
data.loc[data.Reviews > 2000000]
```

Out[207]:

| | App | Category | Rating | Reviews | Size | Installs | Type | Price | Content Rating | |
|---|---|---|---|---|---|---|---|---|---|---|
| **345** | Yahoo Mail – Stay Organized | COMMUNICATION | 4.3 | 4187998 | 16000.0 | 100000000 | Free | 0.0 | Everyone | Co |
| **347** | imo free video calls and chat | COMMUNICATION | 4.3 | 4785892 | 11000.0 | 500000000 | Free | 0.0 | Everyone | Co |
| **366** | UC Browser Mini -Tiny Fast Private & Secure | COMMUNICATION | 4.4 | 3648120 | 3300.0 | 100000000 | Free | 0.0 | Teen | Co |
| **378** | UC Browser - Fast Download Private & Secure | COMMUNICATION | 4.5 | 17712922 | 40000.0 | 500000000 | Free | 0.0 | Teen | Co |
| **383** | imo free video calls and chat | COMMUNICATION | 4.3 | 4785988 | 11000.0 | 500000000 | Free | 0.0 | Everyone | Co |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **9142** | Need for Speed™ No Limits | GAME | 4.4 | 3344300 | 22000.0 | 50000000 | Free | 0.0 | Everyone 10+ | |
| **9166** | Modern Combat 5: eSports FPS | GAME | 4.3 | 2903386 | 58000.0 | 100000000 | Free | 0.0 | Mature 17+ | |
| **10186** | Farm Heroes Saga | FAMILY | 4.4 | 7615646 | 71000.0 | 100000000 | Free | 0.0 | Everyone | |
| **10190** | Fallout Shelter | FAMILY | 4.6 | 2721923 | 25000.0 | 10000000 | Free | 0.0 | Teen | |
| **10327** | Garena Free Fire | GAME | 4.5 | 5534114 | 53000.0 | 100000000 | Free | 0.0 | Teen | |

219 rows × 13 columns

In [208…

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

Out[208]:

```
(219, 13)
```

In [209…

```
data.shape
```

Out[209]: (7717, 13)

In [210…
```
# new data
data = data[data.Reviews <= 2000000]
```

In [211…
```
data.shape
```

Out[211]: (7498, 13)

In [212…
```
data.describe()
```

Out[212]:

| | Rating | Reviews | Size | Installs | Price |
|---|---|---|---|---|---|
| **count** | 7498.000000 | 7.498000e+03 | 7498.000000 | 7.498000e+03 | 7498.000000 |
| **mean** | 4.165191 | 7.246246e+04 | 22001.031275 | 3.939597e+06 | 1.161692 |
| **std** | 0.549805 | 2.121838e+05 | 22573.540980 | 2.779102e+07 | 17.666227 |
| **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 | 2.009000e+03 | 14000.000000 | 1.000000e+05 | 0.000000 |
| **75%** | 4.500000 | 3.237150e+04 | 31000.000000 | 1.000000e+06 | 0.000000 |
| **max** | 5.000000 | 1.986068e+06 | 100000.000000 | 1.000000e+09 | 400.000000 |

In [213…
```
# Installs:  There seems to be some outliers in this field too.
#Apps having very high number of installs should be dropped from the analysis.
#Find out the different percentiles - 10, 25, 50, 70, 90, 95, 99
#Decide a threshold as cutoff for outlier and drop records having values more than tha
```

In [214…
```
data['Installs'].describe()
```

Out[214]:
```
count    7.498000e+03
mean     3.939597e+06
std      2.779102e+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 [215…
```
np.arange(0,1,0.05)
```

Out[215]:
```
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 [216…
```
# To get the percentiles by 0.05
data['Installs'].quantile(q = np.arange(0,1,0.05))
```

Out[216]:
```
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 [217…
```
data['Installs'].quantile(0.99)
```

Out[217]:
```
50000000.0
```

In [218…
```
data.shape
```

Out[218]:
```
(7498, 13)
```

In [219…
```
# Installations above 10,000,000 are outliers and should be dropped
data[data.Installs > 10000000]
```

Out[219]:

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

176 rows × 13 columns

In [220…
```
#new data without outlier
data[data.Installs <= 10000000].shape
```

Out[220]:
```
(7322, 13)
```

In [221…
```
data = data[data.Installs <= 10000000]
```

In [222…
```
data.describe()
```

Out[222]:

| | Rating | Reviews | Size | Installs | Price |
|---|---|---|---|---|---|
| **count** | 7322.000000 | 7.322000e+03 | 7322.000000 | 7.322000e+03 | 7322.000000 |
| **mean** | 4.162292 | 5.080803e+04 | 21661.613289 | 1.712523e+06 | 1.189616 |
| **std** | 0.555119 | 1.456091e+05 | 22451.105449 | 3.203616e+06 | 17.876388 |
| **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 | 1.739500e+03 | 13000.000000 | 1.000000e+05 | 0.000000 |
| **75%** | 4.500000 | 2.743900e+04 | 30000.000000 | 1.000000e+06 | 0.000000 |
| **max** | 5.000000 | 1.736105e+06 | 100000.000000 | 1.000000e+07 | 400.000000 |

In [223…

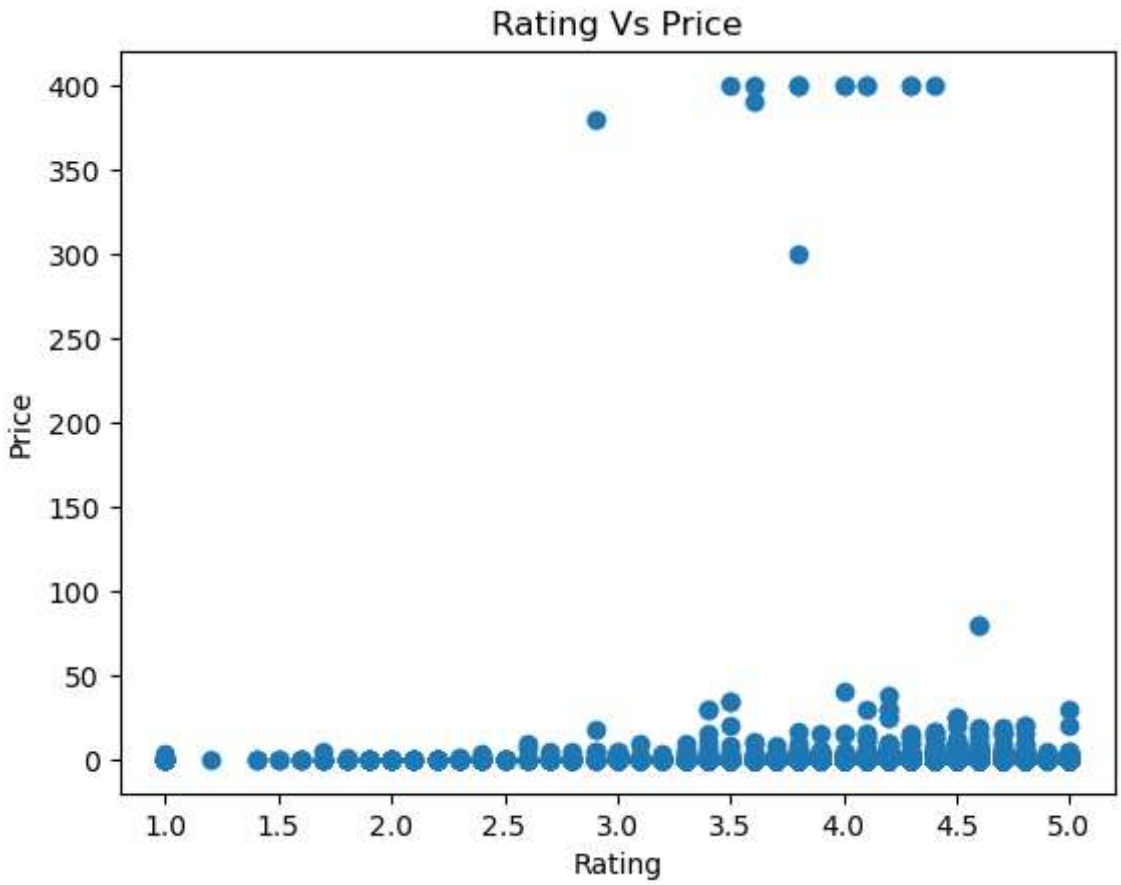
```
# 7.1 Make scatter plot/joinplot for Rating vs. Price
# What pattern do you observe? Does rating increase with price?

#plt.scatter(x,y)

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

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

```
form the plot below, rating does not increase with price
```

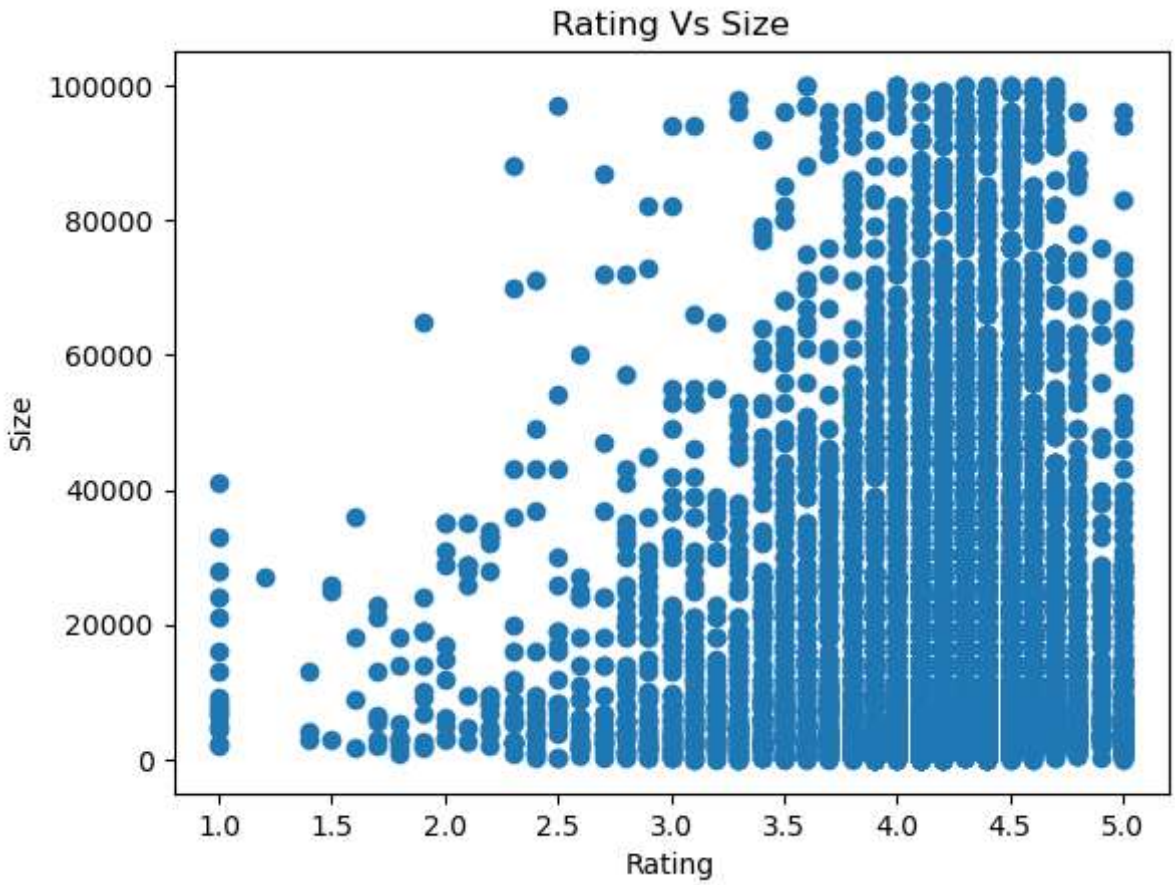



In [224…

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

plt.scatter(data['Rating'], data['Size'])
plt.xlabel('Rating')
plt.ylabel('Size')
plt.title('Rating Vs Size')
print('from the plot, lighter apps have less ratings than the heavier apps and are lik
```

```
from the plot, lighter apps have less ratings than the heavier apps and are likely to
be rated lower
```




In [225…

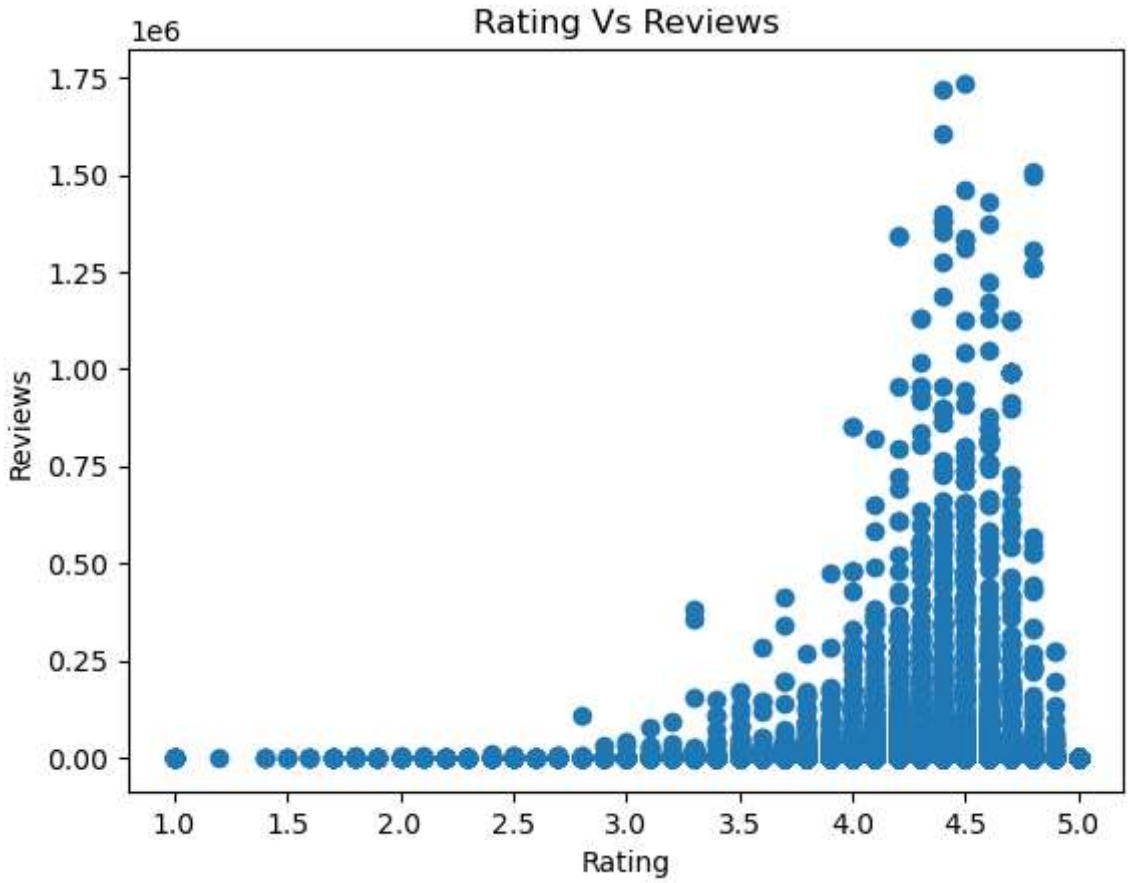
```
# 7.3 Make scatter plot/joinplot for Rating vs. Reviews
# Does more review mean a better rating always?

#plt.scatter(x,y)

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

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

```
from the plot, apps with the most reviews are rated highly
```

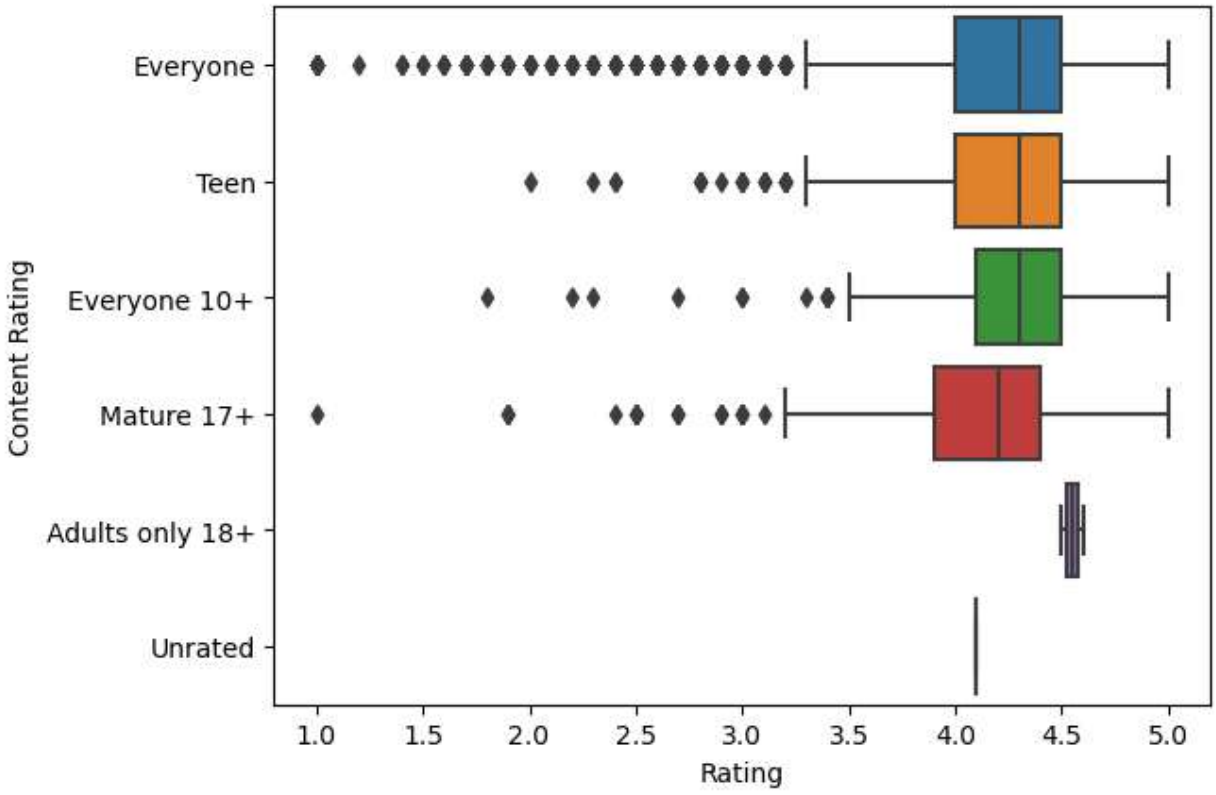



In [226…

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

sns.boxplot(x="Rating", y="Content Rating", data=data)
print('Apps for Teens  Content Rating are generally rated higher than others, while th
```

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

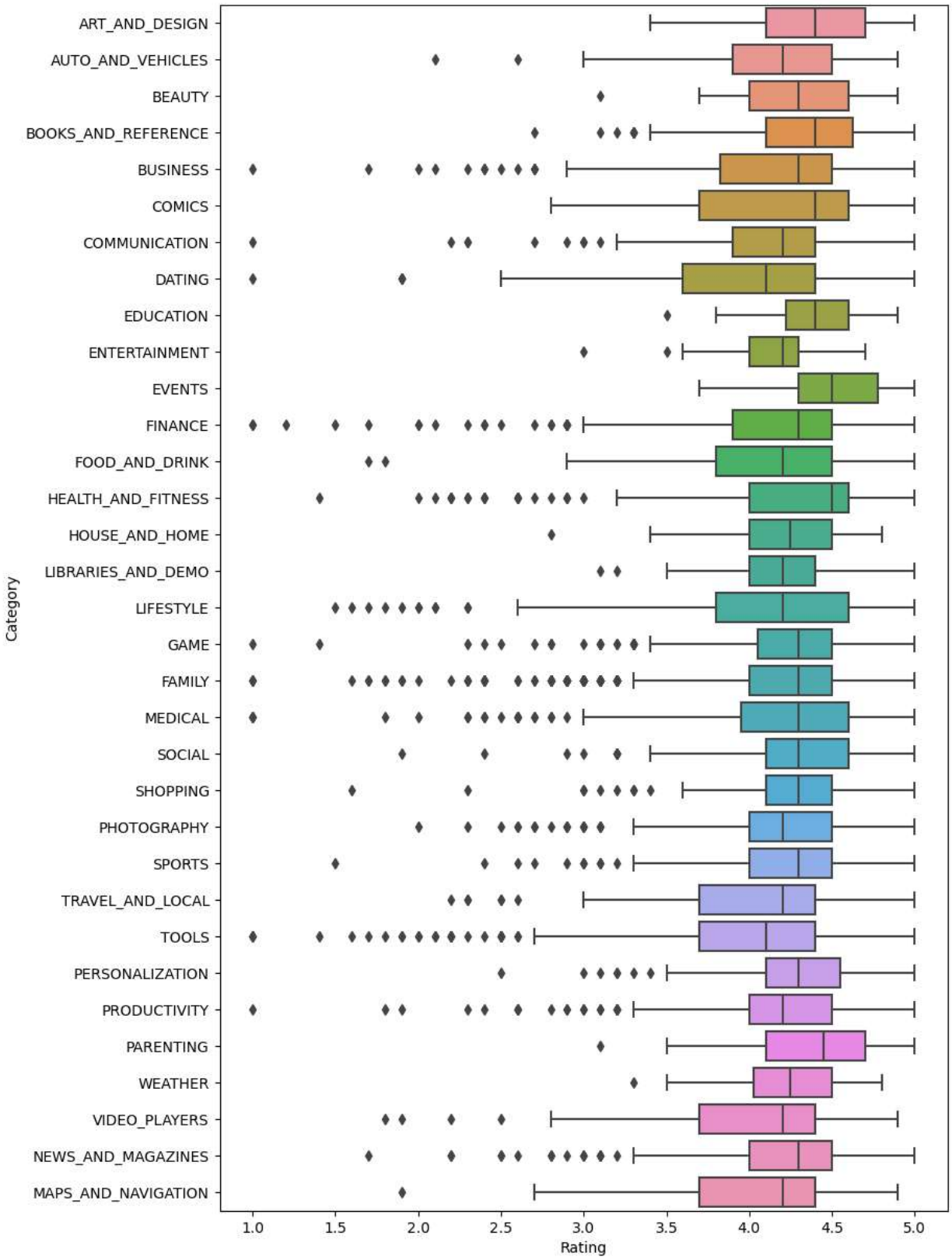



In [227…

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


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

```
Apps for parenting and events show the highest ratings
```




In [228…

```
# 8 For the steps below, create a copy of the dataframe to make all the edits. Name it

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

In [229…

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

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

In [230…
```
# 8.2 Drop columns App, Last Updated, Current Ver, and Android Ver. These variables ar

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

In [231…
```
inp1.columns
```

Out[231]:
```
Index(['Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type', 'Price',
       'Content Rating', 'Genres'],
      dtype='object')
```

In [232…
```
inp1.head()
```

Out[232]:

| | Category | Rating | Reviews | Size | Installs | Type | Price | Content Rating | Genres |
|---|---|---|---|---|---|---|---|---|---|
| **0** | ART_AND_DESIGN | 4.1 | 5.075174 | 9.852247 | 9.210440 | Free | 0.0 | Everyone | Art & Design |
| **1** | ART_AND_DESIGN | 3.9 | 6.875232 | 9.546884 | 13.122365 | Free | 0.0 | Everyone | Art & Design;Pretend Play |
| **2** | ART_AND_DESIGN | 4.7 | 11.379520 | 9.071193 | 15.424949 | Free | 0.0 | Everyone | Art & Design |
| **3** | ART_AND_DESIGN | 4.3 | 6.875232 | 7.937732 | 11.512935 | Free | 0.0 | Everyone | Art & Design;Creativity |
| **4** | ART_AND_DESIGN | 4.4 | 5.123964 | 8.630700 | 10.819798 | Free | 0.0 | Everyone | Art & Design |

In [233…
```
inp1.shape
```

Out[233]:
```
(7322, 9)
```

In [234…
```
# 8.3 Get dummy columns for Category, Genres, and Content Rating.
#This needs to be done as the models do not understand categorical data, and all data
#Dummy encoding is one way to convert character fields to numeric. Name of dataframe s
```

In [235…
```
categorical_cols = ['Category', 'Genres', 'Content Rating', 'Type']

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

In [236…
```
inp2.head()
```

Out[236]:

| | Rating | Reviews | Size | Installs | Price | Category_AUTO_AND_VEHICLES | Category_BEAUTY | C |
|---|---|---|---|---|---|---|---|---|
| **0** | 4.1 | 5.075174 | 9.852247 | 9.210440 | 0.0 | 0 | 0 | |
| **1** | 3.9 | 6.875232 | 9.546884 | 13.122365 | 0.0 | 0 | 0 | |
| **2** | 4.7 | 11.379520 | 9.071193 | 15.424949 | 0.0 | 0 | 0 | |
| **3** | 4.3 | 6.875232 | 7.937732 | 11.512935 | 0.0 | 0 | 0 | |
| **4** | 4.4 | 5.123964 | 8.630700 | 10.819798 | 0.0 | 0 | 0 | |

5 rows × 154 columns

In [237…
```
# 9 Train test split  and apply 70-30 split. Name the new dataframes data_train and da
```

In [238…
```
from sklearn.model_selection import train_test_split
data_train, data_test = train_test_split(inp2, train_size = 0.7, random_state = 32)
```

In [239…
```
data_train
```

Out[239]:

| | Rating | Reviews | Size | Installs | Price | Category_AUTO_AND_VEHICLES | Category_BEAUT |
|---|---|---|---|---|---|---|---|
| **7194** | 4.6 | 4.499810 | 8.648397 | 8.517393 | 0.00 | 0 | |
| **3256** | 4.3 | 6.588926 | 9.852247 | 8.517393 | 4.99 | 0 | |
| **6149** | 4.2 | 6.364751 | 7.650169 | 9.210440 | 0.00 | 0 | |
| **3553** | 4.8 | 3.091042 | 8.594339 | 8.517393 | 0.00 | 0 | |
| **6382** | 4.3 | 8.582981 | 10.545368 | 13.815512 | 0.00 | 0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **4030** | 4.3 | 2.995732 | 7.824446 | 6.908755 | 0.00 | 0 | |
| **2940** | 3.2 | 6.748760 | 10.518700 | 11.512935 | 0.00 | 0 | |
| **1334** | 4.5 | 10.981829 | 11.251574 | 16.118096 | 0.00 | 0 | |
| **1579** | 4.1 | 7.335634 | 10.645449 | 10.819798 | 11.99 | 0 | |
| **2775** | 4.5 | 6.492240 | 8.101981 | 9.210440 | 3.99 | 0 | |

5125 rows × 154 columns

In [240…
```
data_test
```

Out[240]:

| | Rating | Reviews | Size | Installs | Price | Category_AUTO_AND_VEHICLES | Category_BEAUT |
|---|---|---|---|---|---|---|---|
| **2538** | 4.4 | 11.063101 | 9.082621 | 15.424949 | 0.00 | 0 | |
| **5723** | 2.8 | 1.791759 | 6.445720 | 3.931826 | 0.99 | 0 | |
| **6962** | 4.6 | 5.379897 | 10.596660 | 9.210440 | 0.00 | 0 | |
| **2561** | 4.2 | 7.019297 | 8.006701 | 10.819798 | 0.00 | 0 | |
| **2523** | 4.5 | 12.629207 | 10.571343 | 16.118096 | 0.00 | 0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **490** | 4.3 | 8.061171 | 9.852247 | 13.815512 | 0.00 | 0 | |
| **1394** | 4.8 | 4.584967 | 8.160804 | 6.908755 | 2.99 | 0 | |
| **5278** | 4.6 | 6.647688 | 8.630700 | 11.512935 | 0.00 | 0 | |
| **4054** | 2.3 | 3.931826 | 7.937732 | 9.210440 | 0.00 | 0 | |
| **869** | 4.6 | 8.598036 | 9.952325 | 13.122365 | 0.00 | 0 | |

2197 rows × 154 columns

In [241…
```
# 10. Separate the dataframes into X_train, y_train, X_test, and y_test.
y_train = data_train.Rating
X_train = data_train.drop(['Rating'], axis=1)

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

In [242…
```
# 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[242]:
```
▾ LinearRegression

LinearRegression()
```

In [243…
```
# 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[243]:
```
0.16454206015448924
```

In [244…
```
# 12 Make predictions on test set and report R2.
```

In [245…

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

Out[245]:

```
0.10764642709228844
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: