DESCRIPTION

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

Problem Statement:

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

Domain: General

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

Fields in the data –

```
App: Application name

Category: Category to which the app belongs

Rating: Overall user rating of the app

Reviews: Number of user reviews for the app

Size: Size of the app

Installs: Number of user downloads/installs for the app

Type: Paid or Free

Price: Price of the app

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

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

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

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

```
Android Ver: Minimum required Android version
```

In [181…

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

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Steps to perform:

1. Load the data file using pandas.

In [182…

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

In [183…

```
data.head()
```

Out[183…

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

In [184…

```
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
```

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

Dropping the records with null ratings

- this is done because ratings is our target variable

In [185…

```
data.isnull().sum(axis=0)
```

Out[185…

```
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
```

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

In [186…

```
data.dropna(how ='any', inplace = True)
```

In [187…

```
data.isnull().sum(axis=0)
```

Out[187…

```
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
```

Confirming that the null records have been dropped

Change variable to correct types

In [188…

```
data.dtypes
```

Out[188…

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

inp0.head()

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

1. Size column has sizes in Kb as well as Mb. To analyze, you'll need to convert these to numeric. a. Extract the numeric value from the column b. Multiply the value by 1,000, if size is mentioned in Mb
2. Reviews is a numeric field that is loaded as a string field. Convert it to numeric (int/float).
3. Installs field is currently stored as string and has values like 1,000,000+. a. Treat 1,000,000+ as 1,000,000 b. remove '+', ',' from the field, convert it to integer
4. Price field is a string and has $symbol. Remove$'' sign, and convert it to numeric.

4.4 Price column needs to be cleaned

In [189…

```
data.Price.value_counts()[:5]
```

Out[189…

```
0        8715
$2.99     114
$0.99     106
$4.99      70
```

```
$1.99       59
Name: Price, dtype: int64
```

In [190…

```
data.Price.value_counts()
```

Out[190…

```
0           8715
$2.99        114
$0.99        106
$4.99         70
$1.99         59
            ...
$1.29          1
$299.99        1
$379.99        1
$37.99         1
$1.20          1
Name: Price, Length: 73, dtype: int64
```

In [191…

```
def change_Price(val):
    if val == '0':
        return 0
    else:
        return(float(val[1:]))
```

In [192…

```
data["Price"] = data["Price"].map(change_Price)
```

In [193…

```
data.Price.value_counts()
```

Out[193…

```
0.00      8715
2.99       114
0.99       106
4.99        70
1.99        59
          ...
1.29         1
299.99       1
379.99       1
37.99        1
1.20         1
Name: Price, Length: 73, dtype: int64
```

In [194…

```
# data["Price"] = data.Price.map(lambda x: 0 if x == '0' else float(x[1:]))
```

In [195…

```
data["Price"].value_counts()
```

Out[195…

```
0.00      8715
2.99       114
0.99       106
4.99        70
1.99        59
          ...
1.29         1
299.99       1
379.99       1
```

```
37.99        1
1.20         1
Name: Price, Length: 73, dtype: int64
```

Some have dollars, some have 0

- we need to conditionally handle this
- first, let's modify the column to take 0 if value is 0, else take the first letter onwards

4.2 Converting reviews to numeric

In [196…

```
#Use .astype to change data type of value
data.Reviews = data.Reviews.astype("int32")
```

In [197…

```
data.Reviews.describe()
```

Out[197…

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

4.3 Now, handling the installs column

In [198…

```
data.Installs.value_counts()
```

Out[198…

```
1,000,000+        1576
10,000,000+       1252
100,000+          1150
10,000+           1009
5,000,000+         752
1,000+             712
500,000+           537
50,000+            466
5,000+             431
100,000,000+       409
100+               309
50,000,000+        289
500+               201
500,000,000+        72
10+                 69
1,000,000,000+      58
50+                 56
5+                   9
1+                   3
Name: Installs, dtype: int64
```

We'll need to remove the commas and the plus signs

Defining function for the same

In [199…

```
def clean_values(val):
    return int(val.replace(",","").replace("+",""))
```

In [200…

```
data.Installs = data.Installs.map(clean_values)
```

In [201…

```
data.Installs.describe()
```

Out[201…

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

In [202…

```
#data.Installs = data.Installs.astype("float")
```

In [203…

```
def change_size(size):
    if 'M' in size:
        x = size[:-1]
        x = float(x)*1000
        return x
    elif 'k' in size:
        x = size[:-1]
        x = float(x)
        return x
    else:
        return None
```

In [204…

```
change_size("19M")
```

Out[204…

```
19000.0
```

In [205…

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

In [206…

```
data.Size.describe()
```

Out[206…

```
count      7723.000000
mean      22970.456105
std       23449.628935
min           8.500000
25%        5300.000000
50%       14000.000000
75%       33000.000000
max      100000.000000
Name: Size, dtype: float64
```

In [207…

```
data["Size"].isnull().sum()
```

Out[207…

```
1637
```

In [208…

```
data.Size
```

Out[208…

```
0        19000.0
1        14000.0
2         8700.0
3        25000.0
4         2800.0
          ...
10834     2600.0
10836    53000.0
10837     3600.0
10839        NaN
10840    19000.0
Name: Size, Length: 9360, dtype: float64
```

In [209…

```
#filling Size which had NA using forward fill method-fill values with value that precee
data.Size.fillna(method = 'ffill', inplace = True)
```

In [210…

```
data.dtypes
```

Out[210…

```
App                object
Category           object
Rating            float64
Reviews             int32
Size              float64
Installs            int64
Type               object
Price             float64
Content Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
dtype: object
```

5. Some sanity checks

1. Average rating should be between 1 and 5 as only these values are allowed on the play store. Drop the rows that have a value outside this range.
2. Reviews should not be more than installs as only those who installed can review the app. If there are any such records, drop them.
3. For free apps (type = “Free”), the price should not be >0. Drop any such rows.

5.1 Avg. rating should be between 1 and 5, as only these values are allowed on the play store. Drop any rows that have a value outside this range.

In [211…

```
data.Rating.describe()
```

Out[211…

```
count    9360.000000
mean        4.191838
std         0.515263
```

```
min        1.000000
25%        4.000000
50%        4.300000
75%        4.500000
max        5.000000
Name: Rating, dtype: float64
```

Min is 1 and max is 5. Looks good.

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

Checking if reviews are more than installs. Counting total rows like this.

In [212…

```python
len(data[data.Reviews > data.Installs])
```

Out[212…

```
7
```

In [213…

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

Out[213…

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated
2454	KBA-EZ Health Guide	MEDICAL	5.0	4	25000.0	1	Free	0.00	Everyone	Medical	2-Aug-18
4663	Alarmy (Sleep If U Can) - Pro	LIFESTYLE	4.8	10249	30000.0	10000	Paid	2.49	Everyone	Lifestyle	30-Jul-18
5917	Ra Ga Ba	GAME	5.0	2	20000.0	1	Paid	1.49	Everyone	Arcade	8-Feb-17
6700	Brick Breaker BR	GAME	5.0	7	19000.0	5	Free	0.00	Everyone	Arcade	23-Jul-18
7402	Trovami se ci riesci	GAME	5.0	11	6100.0	10	Free	0.00	Everyone	Arcade	11-Mar-17
8591	DN Blog	SOCIAL	5.0	20	4200.0	10	Free	0.00	Teen	Social	23-Jul-18
10697	Mu.F.O.	GAME	5.0	2	16000.0	1	Paid	0.99	Everyone	Arcade	3-Mar-17

In [214…

```python
data = data[data.Reviews <= data.Installs].copy()
```

In [215…

```python
data.shape
```

Out[215…

```
(9353, 13)
```

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

In [216…

```
len(data[(data.Type == "Free") & (data.Price>0)])
```

Out[216…

```
0
```

5.A. Performing univariate analysis:

5.A. Performing univariate analysis:

- Boxplot for Price o Are there any outliers? Think about the price of usual apps on Play Store.
- Boxplot for Reviews o Are there any apps with very high number of reviews? Do the values seem right?
- Histogram for Rating o How are the ratings distributed? Is it more toward higher ratings?
- Histogram for Size ### Note down your observations for the plots made above. Which of these seem to have outliers?

Box plot for price

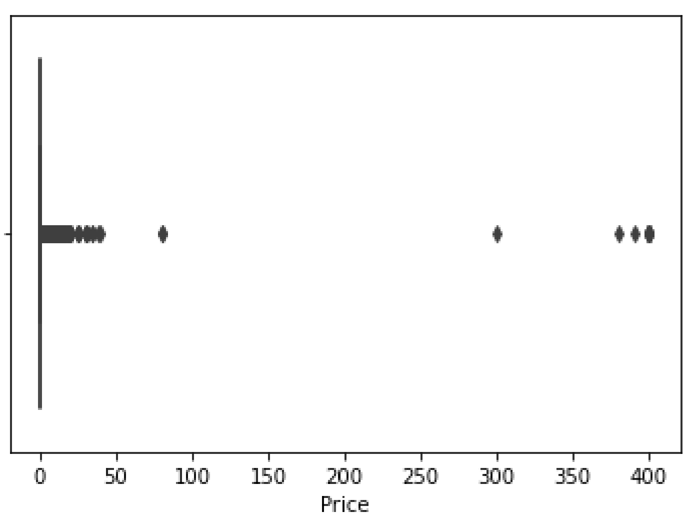
o Are there any outliers? Think about the price of usual apps on Play Store.

There are outliers on this plot. Most apps are free, so the prices $75,$300, and around $ 400 are outliers. These apps raise suspicion and should be further investigated and most likely taken out of the dataset.

The mean price of the dataset is 0.96 & the Std 15.82. This means most of the prices are free or very low cost. The outliers are not even within 3 SD's of the mean. This means these outliers represent less than 0.3 % of the data.

In [217…

```
sns.boxplot(data.Price);
```



In [218…

```
print(sum(data.Price != 0))
print(sum(data.Price > 50))
print(sum(data.Price > 200))
```

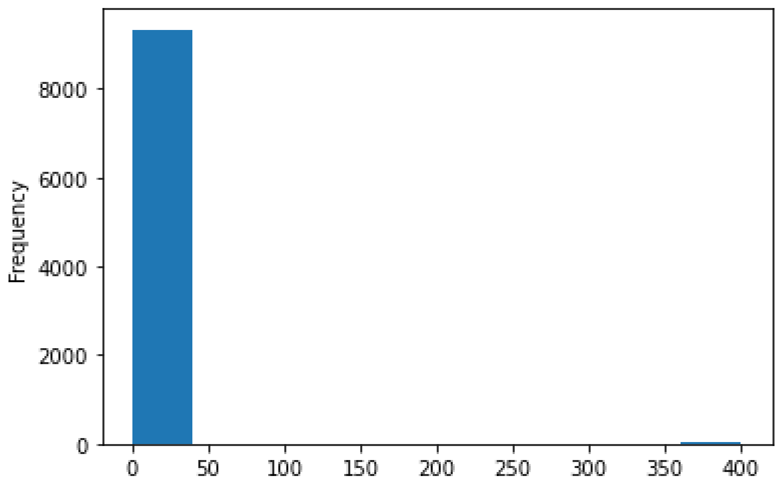
```
642
17
15
```

In [219…

```
data.Price.plot.hist()
```

Out[219…

```
<AxesSubplot:ylabel='Frequency'>
```



Box plot for Reviews

o Are there any apps with very high number of reviews? Do the values seem right?

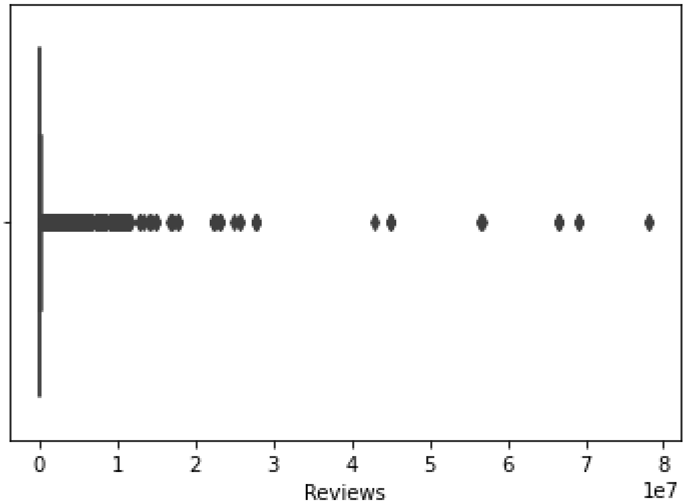
There are quite a few apps with a high number of reviews. Obviously, there are a lot of installs, so this makes sense. Further, there are many more installs outliers then reviews outliers (despite the fact the box plots look the opposite of that).

In [220…

```
sns.boxplot(data.Reviews)
```

Out[220…

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



In [221…

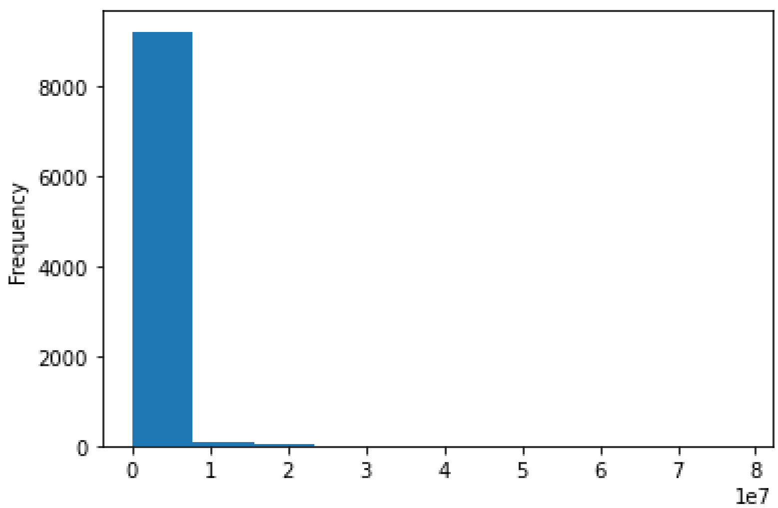
```
#Approximately 45 outliers for reviews
sum(data.Reviews > 15000000)
```

Out[221… 45

In [222…

```
data.Reviews.plot.hist()
```

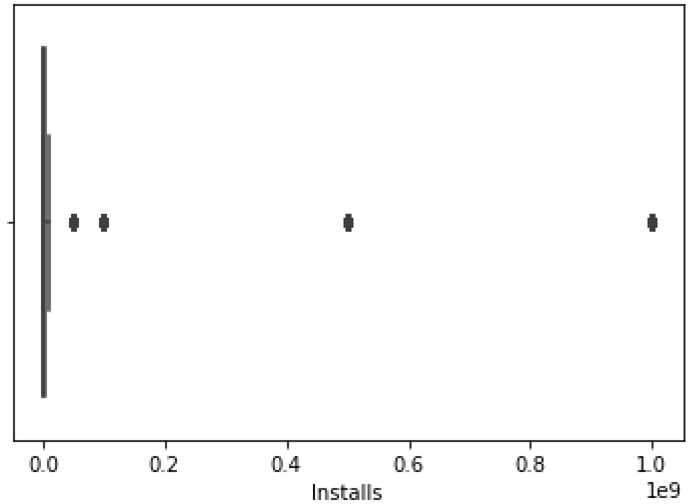
Out[222… <AxesSubplot:ylabel='Frequency'>



In [223…

```
sns.boxplot(data.Installs)
```

Out[223… <AxesSubplot:xlabel='Installs'>



In [224…

```
sum(data.Installs > 200000000)
```

Out[224…

```
130
```

Histogram for Rating

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

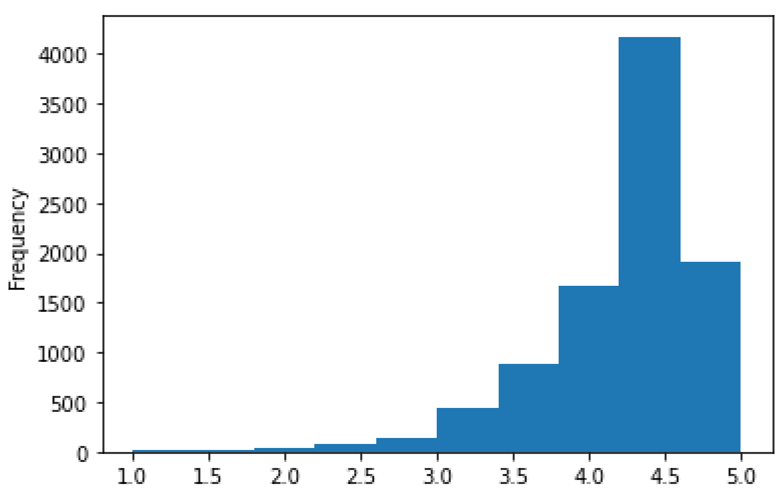
In [225…

```
#Calculate the mean, median, and mode for ratings
import statistics
mean = sum(data.Rating)/len(data.Rating)
median = statistics.median(data.Rating)
mode = statistics.mode(data.Rating)
print("The mean is: ", mean)
print("The mode is: ", mode)
print("The median is: ", median)
```

```
The mean is:  4.191254143055709
The mode is:  4.4
The median is:  4.3
```

In [226…

```
data.Rating.plot.hist();
```



The values are skewed to the left or negatively skewed (towards higher ratings). This means the median 4.3 is greater than the mean 4.19. However, most of the values are at 3.5 or above, which means most apps have at least an average rating. There do appear to be outliers, but most of the values center around the mean.

Histogram of Size

In [227…

```
mean = sum(data.Size)/len(data.Size)
median = statistics.median(data.Size)
mode = statistics.mode(data.Size)
print("The mean is: ", mean)
print("The mode is: ", mode)
print("The median is: ", median)
```

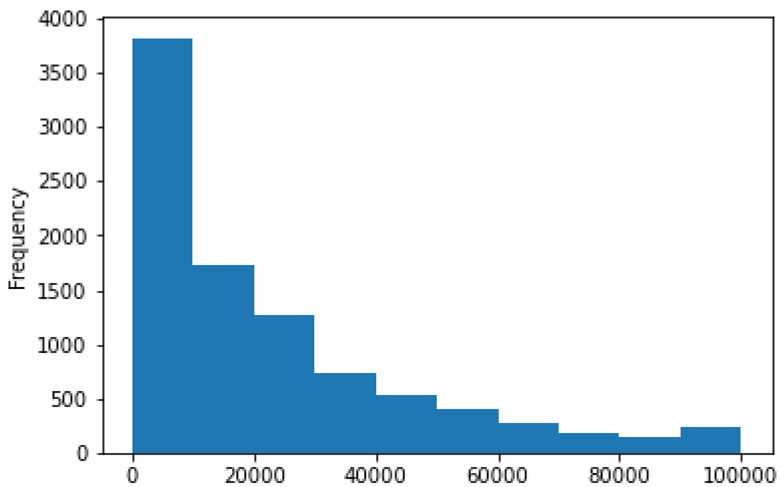
```
The mean is:  23147.92499732706
The mode is:  14000.0
The median is:  15000.0
```

In [228…

```
data.Size.plot.hist()
```

Out[228…

```
<AxesSubplot:ylabel='Frequency'>
```



As can be seen from the plot above, the data is skewed to the right, or positively skewed. This can also be vertified by the fact the mean is greater than the median.

This is opposite to the ratings graph, which is negatively skewed. There do appear to be a higher number of outliers in the size histogram, then in the ratings histogram.

Overall Conclusion

The boxplots for Price and Reviews show respectively approximately 17 and 45 outliers. The histograms for rating and size show opposite skewness and both show some outliers. They also show that size clearly has more outliers than what ratings does. But for the number of values in the dataset, the number of ouliers is a small in comparison.

6. Outlier treatment:

1. Price: From the box plot, it seems like there are some apps with very high price. A price of $200 for an application on the Play Store is very high and suspicious! a. Check out the records with very high price
   `i.  Is 200 indeed a high price?`
   b. Drop these as most seem to be junk apps
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.
3. Installs: There seems to be some outliers in this field too. Apps having very high number of installs should be dropped from the analysis.
   ```
   a.  Find out the different percentiles – 10, 25, 50, 70, 90, 95, 99
   b.  Decide a threshold as cutoff for outlier and drop records having values
   more than that
   ```

6.1. Price: From the box plot, it seems like there are some apps with very high price. A price of $200 for an application on the Play Store is very high and suspicious!

```
a.  Check out the records with very high price
    i.  Is 200 indeed a high price?
b.  Drop these as most seem to be junk apps
```

In [229…

```
len(data[data.Price > 200])
```

Out[229…

```
15
```

In [230…

```
data[data.Price > 200]
```

Out[230…

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
4197	most expensive app (H)	FAMILY	4.3	6	1500.0	100	Paid	399.99	Everyone	Entertainment
4362	💎 I'm rich	LIFESTYLE	3.8	718	26000.0	10000	Paid	399.99	Everyone	Lifestyle
4367	I'm Rich - Trump Edition	LIFESTYLE	3.6	275	7300.0	10000	Paid	400.00	Everyone	Lifestyle
5351	I am rich	LIFESTYLE	3.8	3547	1800.0	100000	Paid	399.99	Everyone	Lifestyle
5354	I am Rich Plus	FAMILY	4.0	856	8700.0	10000	Paid	399.99	Everyone	Entertainment
5355	I am rich VIP	LIFESTYLE	3.8	411	2600.0	10000	Paid	299.99	Everyone	Lifestyle
5356	I Am Rich Premium	FINANCE	4.1	1867	4700.0	50000	Paid	399.99	Everyone	Finance
5357	I am extremely Rich	LIFESTYLE	2.9	41	2900.0	1000	Paid	379.99	Everyone	Lifestyle

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
5358	I am Rich!	FINANCE	3.8	93	22000.0	1000	Paid	399.99	Everyone	Finance
5359	I am rich(premium)	FINANCE	3.5	472	965.0	5000	Paid	399.99	Everyone	Finance
5362	I Am Rich Pro	FAMILY	4.4	201	2700.0	5000	Paid	399.99	Everyone	Entertainment
5364	I am rich (Most expensive app)	FINANCE	4.1	129	2700.0	1000	Paid	399.99	Teen	Finance
5366	I Am Rich	FAMILY	3.6	217	4900.0	10000	Paid	389.99	Everyone	Entertainment
5369	I am Rich	FINANCE	4.3	180	3800.0	5000	Paid	399.99	Everyone	Finance
5373	I AM RICH PRO PLUS	FINANCE	4.0	36	41000.0	1000	Paid	399.99	Everyone	Finance

In [231…

```
data = data[data.Price < 200].copy()
data[data.Price > 200]
```

Out[231…

App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	And

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 [232…

```
data = data[data.Reviews <= 2000000].copy()
data.shape
```

Out[232…

```
(8885, 13)
```

In [233…

```
#To verify data was deleted
data[data.Reviews > 2000000]
```

Out[233…

App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	And

6.3 Installs: There seems to be some outliers in this field too. Apps having very high number of installs should be dropped from the analysis.

```
    a.  Find out the different percentiles – 10, 25, 50, 70, 90, 95, 99
    b.  Decide a threshold as cutoff for outlier and drop records having
values more than that
```

Dropping very high Installs values

In [234…

```
data.Installs.quantile([0.1, 0.25, 0.5, 0.70, 0.9, 0.95, 0.99])
```

Out[234…

```
0.10         1000.0
0.25        10000.0
0.50       500000.0
0.70      1000000.0
0.90     10000000.0
0.95     10000000.0
0.99    100000000.0
Name: Installs, dtype: float64
```

Looks like there are just 1% apps having more than 100M installs. These apps might be genuine, but will definitely skew our analysis.

We need to drop these.

In [235…

```
len(data[data.Installs >= 100000000])
```

Out[235…

```
142
```

In [236…

```
data = data[data.Installs < 100000000]
len(data)
```

Out[236…

```
8743
```

7. Bivariate analysis: Let’s look at how the available predictors relate to the variable of interest, i.e., our target variable rating. Make scatter plots (for numeric features) and box plots (for character features) to assess the relations between rating and the other features.

```
1.  Make scatter plot/joinplot for Rating vs. Price
    a.  What pattern do you observe? Does rating increase with price?
2.  Make scatter plot/joinplot for Rating vs. Size
    a.  Are heavier apps rated better?
3.  Make scatter plot/joinplot for Rating vs. Reviews
    a.  Does more review mean a better rating always?
4.  Make boxplot for Rating vs. Content Rating
    a.  Is there any difference in the ratings? Are some types liked
better?
5.  Make boxplot for Ratings vs. Category
    a.  Which genre has the best ratings?
```

For each of the plots above, note down your observation.

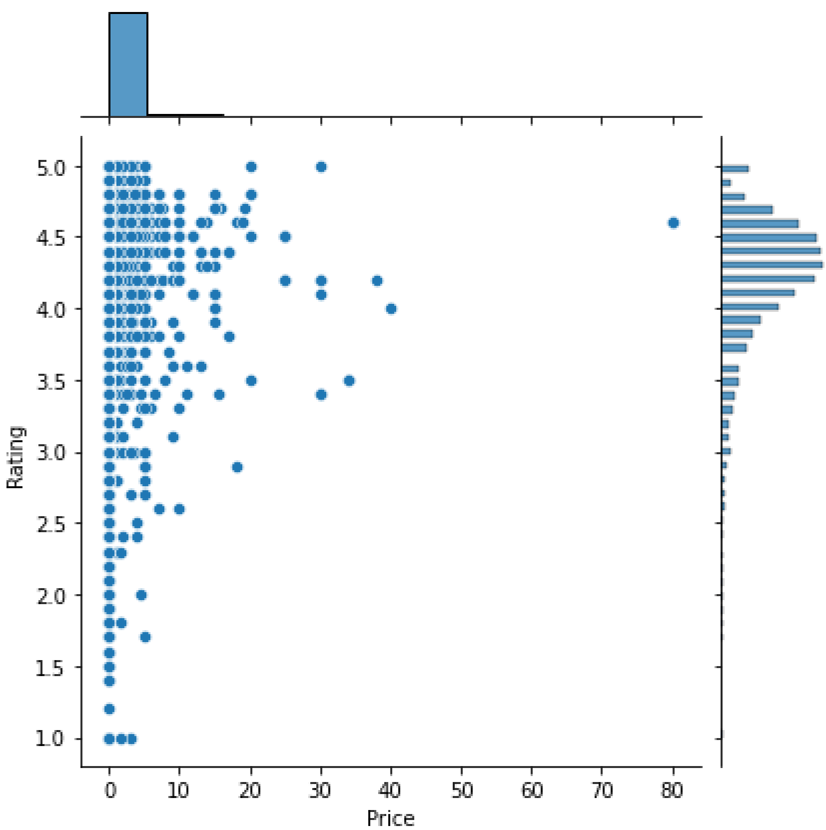
7.1. Make scatter plot/joinplot for Rating vs Price

```
a.  What pattern do you observe? Does rating increase with price?
```

It seems that there is no direct correlation or a very weak positive correlation between rating and price. It seems that both free and paid apps have high ratings. So rating does not neccessarily increase with price. Also, the $80 app appears to be an outlier, as there are no points clustered around it to warrant the price.

In [237…

```
#can add into joint plot ylim = (,), xlim = (,) to set axii limits
sns.jointplot(data.Price, data.Rating);
```

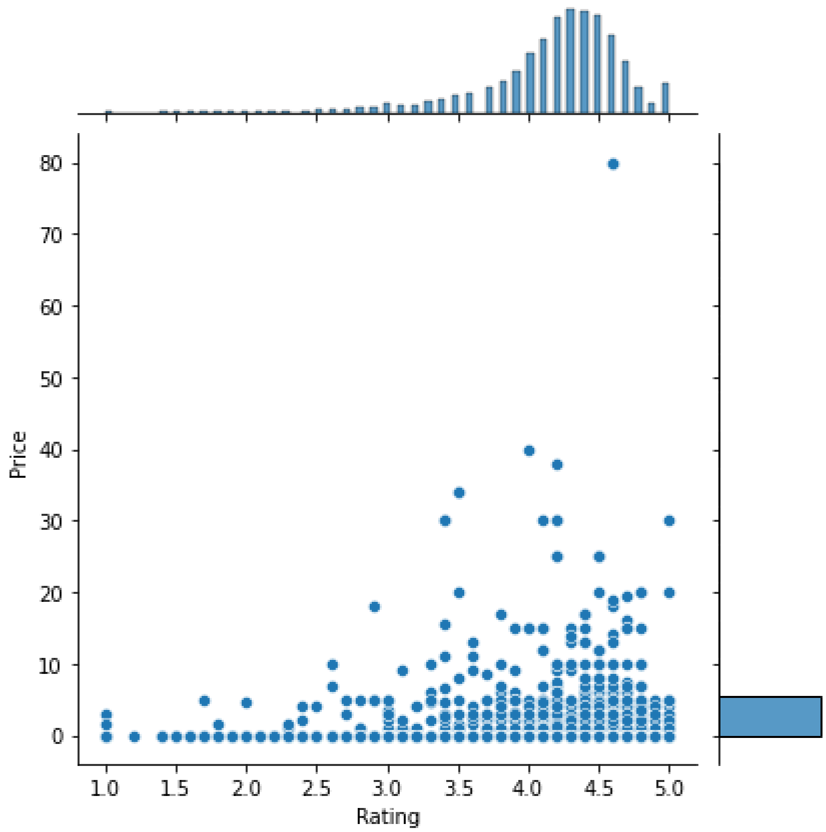


In [238…

```
sns.jointplot(data.Rating, data.Price)
```

Out[238…

```
<seaborn.axisgrid.JointGrid at 0x20bafa44bb0>
```



7.2 Make scatter plot/joinplot for Rating vs Size

```
a. Are heavier apps rated better?
```

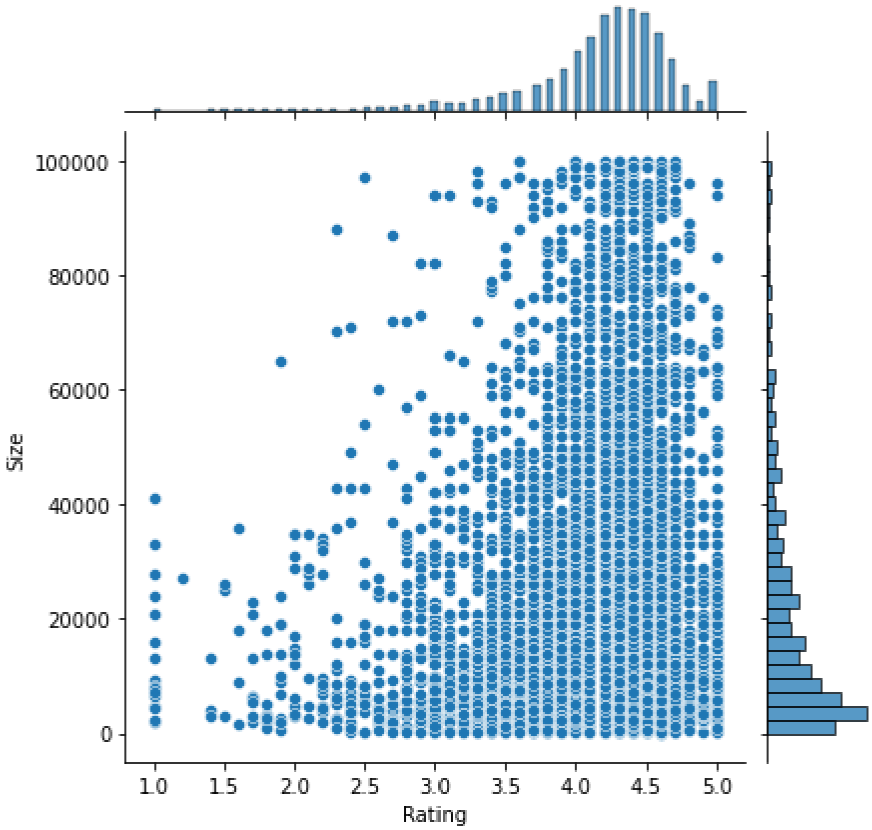
Heavier apps are not rated better. The data is too scattered to draw any kind of relationship or correlation.

In [239…

```
sns.jointplot(data.Rating, data.Size)
```

Out[239…

```
<seaborn.axisgrid.JointGrid at 0x20bbcc3d7c0>
```



7.3 Make scatter plot/joinplot for Rating vs Reviews

```
a.  Does more review mean a better rating always?
```

Reviews and Ratings do not seem to have a close correlation, as the linear regression line is practically straight up. Most apps that have a high number of reviews have a high ratings, but there are a few exceptions to that. But also many of the apps with a low number of reviews have high ratings as well.

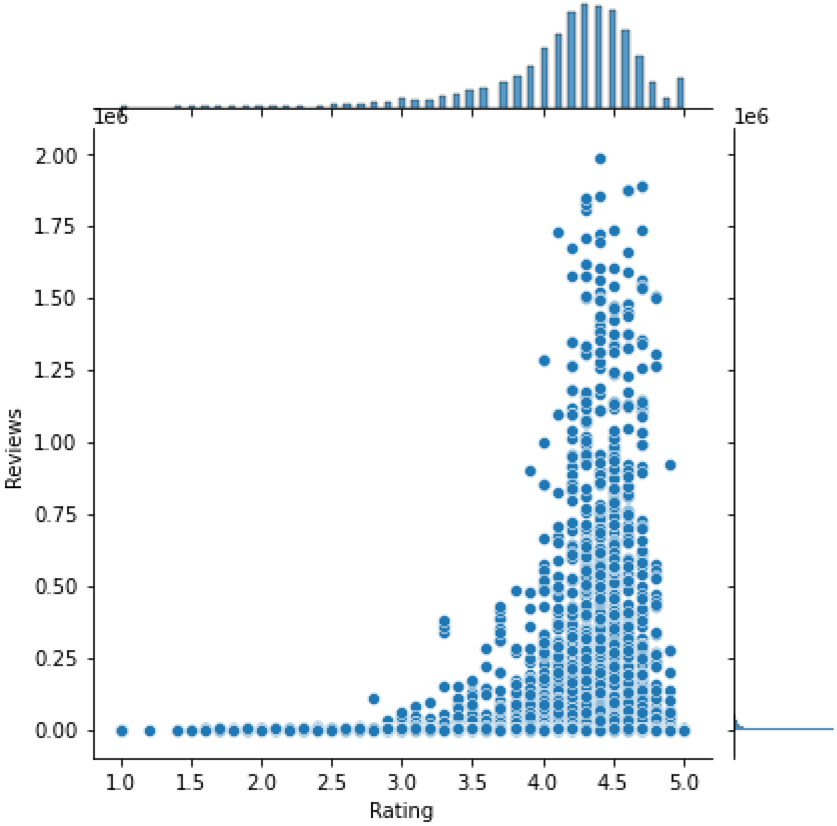
Most apps that have no ratings, however, have a rating below 3.0. This may be a good method to sort out junk apps.

In [240…

```
sns.jointplot(data.Rating, data.Reviews)
```

Out[240…

```
<seaborn.axisgrid.JointGrid at 0x20bbce73400>
```



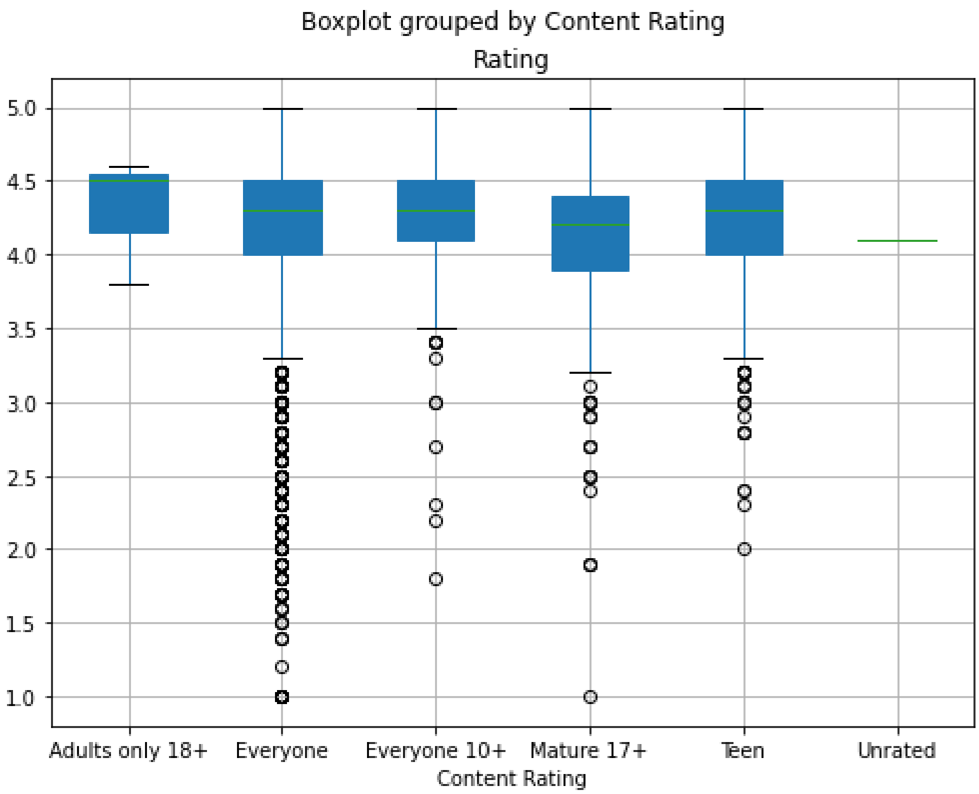
7.4 Make boxplot for Rating vs Content Rating

```
a.  Is there any difference in the ratings? Are some types liked better?
The values for categories Everyone, Everyone 10+, Mature 17+, and Teen
roughly have the same median at about 4.25 and all max out at about 5.0.

However, the adults only 18+ and Unrated don't max out at 5.0 rating and
have lower mimimum ratings than other apps.
```

In [241…

```
box = data.boxplot(column = 'Rating', by = 'Content Rating', figsize = (8,6), patch_art
```



7.5 Make boxplot for Ratings vs. Category

```
a.  Which genre has the best ratings?
```

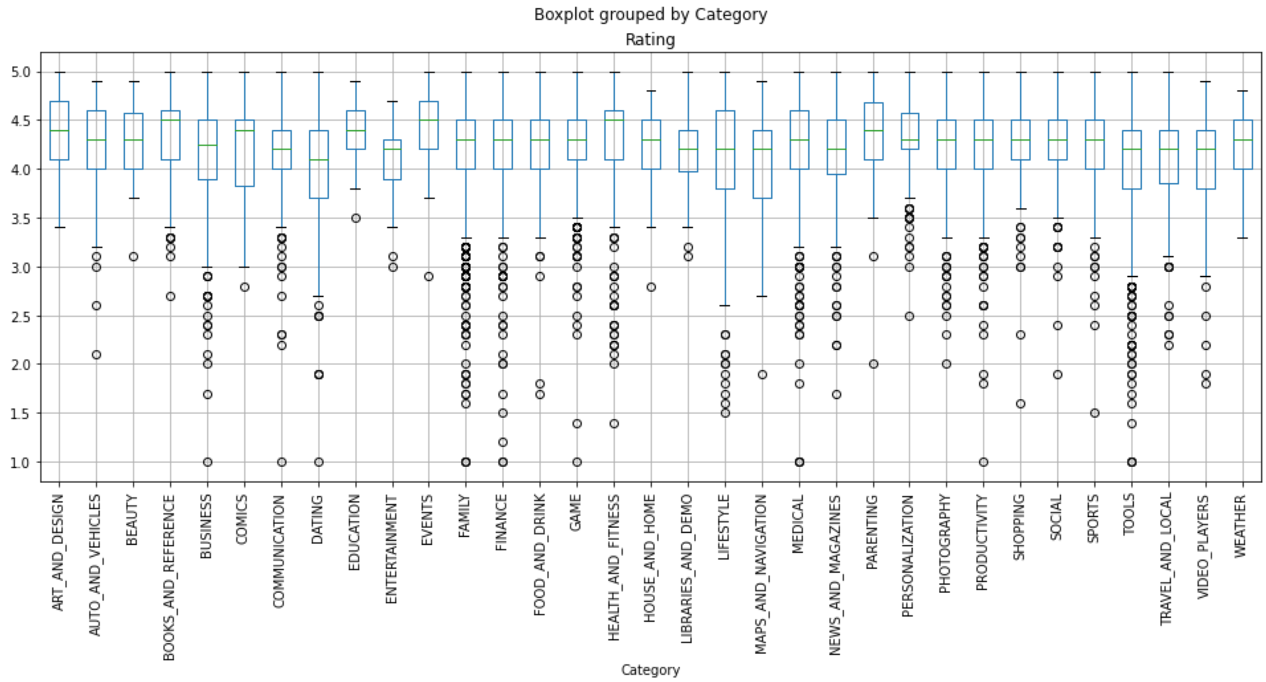
In [242…

```
data.boxplot(column = "Rating", by = "Category", figsize=(16,6));
plt.xticks(rotation=90)
```

Out[242…

```
(array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33]),
 [Text(1, 0, 'ART_AND_DESIGN'),
  Text(2, 0, 'AUTO_AND_VEHICLES'),
  Text(3, 0, 'BEAUTY'),
  Text(4, 0, 'BOOKS_AND_REFERENCE'),
  Text(5, 0, 'BUSINESS'),
  Text(6, 0, 'COMICS'),
  Text(7, 0, 'COMMUNICATION'),
  Text(8, 0, 'DATING'),
  Text(9, 0, 'EDUCATION'),
  Text(10, 0, 'ENTERTAINMENT'),
  Text(11, 0, 'EVENTS'),
  Text(12, 0, 'FAMILY'),
  Text(13, 0, 'FINANCE'),
  Text(14, 0, 'FOOD_AND_DRINK'),
  Text(15, 0, 'GAME'),
  Text(16, 0, 'HEALTH_AND_FITNESS'),
  Text(17, 0, 'HOUSE_AND_HOME'),
  Text(18, 0, 'LIBRARIES_AND_DEMO'),
  Text(19, 0, 'LIFESTYLE'),
  Text(20, 0, 'MAPS_AND_NAVIGATION'),
  Text(21, 0, 'MEDICAL'),
  Text(22, 0, 'NEWS_AND_MAGAZINES'),
  Text(23, 0, 'PARENTING'),
  Text(24, 0, 'PERSONALIZATION'),
```

```
Text(25, 0, 'PHOTOGRAPHY'),
Text(26, 0, 'PRODUCTIVITY'),
Text(27, 0, 'SHOPPING'),
Text(28, 0, 'SOCIAL'),
Text(29, 0, 'SPORTS'),
Text(30, 0, 'TOOLS'),
Text(31, 0, 'TRAVEL_AND_LOCAL'),
Text(32, 0, 'VIDEO_PLAYERS'),
Text(33, 0, 'WEATHER')])
```



The best apps are those with up to 5.0 in ratings:

Art_And_Design, Books_And_Reference, Business, Comics, Communication, Dating, Events, Family, Finance, Food_and_Drink, Game, Health_and_Fitness, Libraries_And_Demo, Lifestyle, Medical, News_and_Magazine, Parenting, Personalizational, Photography, Productivity, Shopping, Social, Sports, Tools, Travel_and_Local

All apps have a median roughly at about 4.25, so the apps with up to 5.0 in ratings is probably the best way to sort out the best ones.

8 Data preprocessing

For the steps below, create a copy of the dataframe to make all the edits. Name it inp1.

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.
2. Drop columns App, Last Updated, Current Ver, and Android Ver. These variables are not useful for our task.
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 should be numeric. Dummy encoding is one way to convert character fields to numeric. Name of dataframe should be inp2.

Making a copy of the dataset

In [243…

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

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.

In [244…

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

In [245…

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

8.2 Drop columns App, Last Updated, Current Ver, and Android Ver. These variables are not useful for our task.

In [246…

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

In [247…

```
inp1.shape
```

Out[247…

```
(8743, 8)
```

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 should be numeric. Dummy encoding is one way to convert character fields to numeric. Name of dataframe should be inp2.

Getting dummy variables for Category, Genres, Content Rating

In [248…

```
inp1.dtypes
```

Out[248…

```
Category           object
Rating            float64
Reviews           float64
Size              float64
Installs          float64
Price             float64
Content Rating     object
Genres             object
dtype: object
```

In [249…

```
inp2 = pd.get_dummies(inp1, columns=['Category', 'Genres', 'Content Rating'])
inp2
```

Out[249…

	Rating	Reviews	Size	Installs	Price	Category_ART_AND_DESIGN	Category_AUTO_AND_V
0	4.1	5.075174	19000.0	9.210440	0.0	1	

	Rating	Reviews	Size	Installs	Price	Category_ART_AND_DESIGN	Category_AUTO_AND_V
1	3.9	6.875232	14000.0	13.122365	0.0	1	
2	4.7	11.379520	8700.0	15.424949	0.0	1	
3	4.5	12.281389	25000.0	17.727534	0.0	1	
4	4.3	6.875232	2800.0	11.512935	0.0	1	
...	...	...	...	...	...	...	
10834	4.0	2.079442	2600.0	6.216606	0.0	0	
10836	4.5	3.663562	53000.0	8.517393	0.0	0	
10837	5.0	1.609438	3600.0	4.615121	0.0	0	
10839	4.5	4.744932	3600.0	6.908755	0.0	0	
10840	4.5	12.894981	19000.0	16.118096	0.0	0	

8743 rows × 159 columns

In [250…

```
inp2.columns
```

Out[250…

```
Index(['Rating', 'Reviews', 'Size', 'Installs', 'Price',
       'Category_ART_AND_DESIGN', 'Category_AUTO_AND_VEHICLES',
       'Category_BEAUTY', 'Category_BOOKS_AND_REFERENCE', 'Category_BUSINESS',
       ...
       'Genres_Video Players & Editors;Creativity',
       'Genres_Video Players & Editors;Music & Video', 'Genres_Weather',
       'Genres_Word', 'Content Rating_Adults only 18+',
       'Content Rating_Everyone', 'Content Rating_Everyone 10+',
       'Content Rating_Mature 17+', 'Content Rating_Teen',
       'Content Rating_Unrated'],
      dtype='object', length=159)
```

In [251…

```
inp2.shape
```

Out[251…

```
(8743, 159)
```

9. Train test split and apply 70-30 split. Name the new dataframes df_train and df_test.

Train - test split

In [252…

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Below line takes entire dataset (np2) and randomly separates it into train and test dataset according to test_size = 0.3. df_train & df_test make up all paramters in the dataset.

These values will be further split into x and y variables for train and test. Y is the variable of interest which is rating, where as x is all the other values affecting ratings.

In [253…

```
df_train, df_test = train_test_split(inp2, test_size = 0.3, random_state = 42)
display(df_train.head())
display(df_test.head())
```

	Rating	Reviews	Size	Installs	Price	Category_ART_AND_DESIGN	Category_AUTO_AND_V
2202	4.6	4.304065	12000.0	6.908755	2.99	0	
4576	4.1	9.622318	14000.0	13.815512	0.00	0	
2811	4.7	11.868044	22000.0	16.118096	0.00	0	
10760	4.4	3.583519	2400.0	6.908755	7.99	0	
10527	4.9	7.259116	20000.0	9.210440	0.00	0	

5 rows × 159 columns

	Rating	Reviews	Size	Installs	Price	Category_ART_AND_DESIGN	Category_AUTO_AND_VE
7792	4.1	2.079442	3400.0	4.615121	3.49	0	
9008	5.0	2.302585	1500.0	4.615121	0.00	0	
5584	4.0	6.107023	3100.0	10.819798	0.00	0	
1875	4.5	11.911339	46000.0	16.118096	0.00	0	
3740	4.5	12.788135	12000.0	16.118096	0.00	0	

5 rows × 159 columns

In [291…

```
display(df_train.shape)
display(df_test.shape)
```

```
(6120, 159)
(2623, 159)
```

In [ ]:

10. Separate the dataframes into X_train, y_train, X_test, and y_test.

In [255…

```
y_train = df_train.Rating
x_train = df_train.drop("Rating", axis = 1)
```

In [256…

```
y_test = df_test.Rating
x_test = df_test.drop("Rating", axis=1)
```

In [258…

```
print(y_train.shape)
print(x_train.shape)

print(y_test.shape)
print(x_test.shape)
```

```
(6120,)
(6120, 158)
(2623,)
(2623, 158)
```

In [292…

```
display(x_train[:5])
```

	Reviews	Size	Installs	Price	Category_ART_AND_DESIGN	Category_AUTO_AND_VEHICLES
2202	4.304065	12000.0	6.908755	2.99	0	0
4576	9.622318	14000.0	13.815512	0.00	0	0
2811	11.868044	22000.0	16.118096	0.00	0	0
10760	3.583519	2400.0	6.908755	7.99	0	0
10527	7.259116	20000.0	9.210440	0.00	0	0

5 rows × 158 columns

11 . Model building

- Use linear regression as the technique
- Report the R2 on the train set

In [280…

```
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(x_train, y_train)     #training the data, training the m

print(model.intercept_)         #Bo
print(model.coef_)              #B1, B2, B3, ... Bn
```

```
4.606462296107818
[ 1.74528780e-01 -4.62448738e-07 -1.47966147e-01 -5.04838166e-03
  1.74692933e-01  4.58696399e-02  9.86748252e-02  7.84259914e-02
 -3.72716139e-02  2.13870348e-01 -5.31102144e-02 -1.22478050e-01
 -5.10116508e-02 -1.28648595e-01  1.38717147e-01 -1.50524723e-02
 -4.75384420e-02 -4.70925051e-02  1.72606680e-01 -9.87762563e-03
 -1.28189234e-03  1.68772629e-02 -9.78111503e-02 -5.66393627e-02
  2.00835593e-02 -5.39833873e-02  4.89846022e-02  4.38074211e-02
 -3.82018000e-02 -3.17704997e-02  1.04181438e-02 -1.63249304e-02
 -1.63383332e-01 -3.20165919e-02 -3.44191907e-02 -7.90738946e-03
 -1.72078577e-02 -2.30827210e-01  1.43677281e-03 -2.99307428e-01
 -9.53531654e-02  1.80075828e-01 -2.60235293e-01 -2.12256614e-01
 -4.80029917e-02  2.35054606e-01  1.08343635e-01  2.01274407e-01
 -1.34925108e-01  4.58696399e-02  9.86748252e-02 -1.76623801e-01
 -1.49377782e-01  2.43334473e-01  6.33015151e-01  7.84259914e-02
 -3.86659907e-02 -3.72716139e-02 -2.39819586e-01 -2.80641104e-01
  2.25902630e-13 -1.92957381e-01 -1.76965103e-01  5.73100598e-02
  3.59283091e-01 -3.95280405e-02  6.06172666e-02  5.97585005e-02
 -8.09763162e-02 -2.79862820e-01  4.93733168e-01 -5.31102144e-02
  1.28314058e-01 -1.22478050e-01  1.33253477e-01  5.07975570e-01
  3.63250552e-02  2.07377693e-01  1.96586925e-01  4.03853045e-02
  1.32541603e-01 -2.94272780e-01  9.85598945e-02 -9.66404533e-02
 -2.23793221e-01  1.43195963e-01  4.91749781e-02 -7.27740593e-02
  4.50869783e-02  1.04636800e-01  3.33317339e-01  3.02413806e-01
 -2.69449939e-02 -3.21876647e-01  1.38717147e-01 -4.75384420e-02
 -4.70925051e-02 -9.87762563e-03 -4.16063108e-01  2.30609732e-01
 -1.28189234e-03  1.68772629e-02  5.09449095e-02  7.26112437e-03
 -1.48756060e-01 -5.66393627e-02  2.00835593e-02 -3.30616237e-01
  3.92258245e-01  1.19665052e-01 -5.39833873e-02  1.90929823e-01
 -2.41564556e-01 -2.48602467e-01  3.48221801e-01  4.38074210e-02
 -3.82018000e-02 -3.17704997e-02  3.48677357e-02 -6.66198658e-02
  1.04540208e-01  8.70725504e-02  5.46174162e-01 -3.18104045e-01
  9.26864417e-02  4.44089210e-15 -1.58704777e-01 -1.21426265e-01
  3.48160713e-02 -1.26226697e-01  1.04181438e-02 -1.03770349e-01
  9.96442899e-02 -2.50841760e-01 -1.70448117e-01 -1.63249304e-02
  8.78794243e-02 -1.88632450e-01 -1.50499690e-01  1.88024518e-01
  0.00000000e+00  5.35705653e-01 -1.18067699e-01  8.60511068e-02
 -8.19716744e-02  4.75524837e-02 -3.11281822e-01 -1.66469421e-01
 -2.94912595e-01 -2.31542272e-01 -1.72078577e-02 -7.96337298e-02
  1.19332139e-02  1.98413367e-03 -2.52602268e-03 -1.73874430e-02
  5.99611809e-03  0.00000000e+00]
```

In [289…

```
from sklearn.metrics import r2_score
y_train_pred = model.predict(x_train)

print (r2_score(y_train, y_train_pred))
print(y_train_pred[:5])
print(y_train[:5])
```

```
0.16369604316845499
[4.43421292 4.17357448 4.20825171 4.1504101  4.42778486]
2202     4.6
```

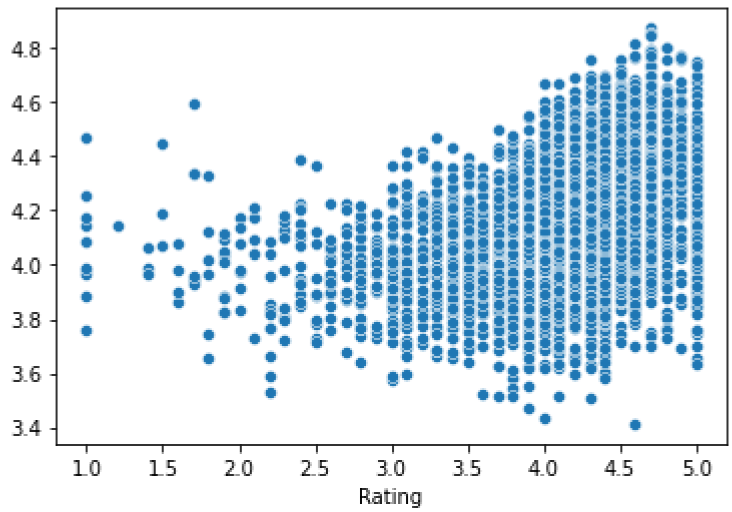
```
4576     4.1
2811     4.7
10760    4.4
10527    4.9
Name: Rating, dtype: float64
```

In [277…

```
sns.scatterplot(y_train, y_train_pred)
```

Out[277…

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



12. Make predictions on test set and report R2.

In [66]:

```
```

Out[66]:

```
0.15866991924983265
```

In [290…

```
y_test_pred = model.predict(x_test)

print (r2_score(y_test, y_test_pred))
print(y_test_pred[:5])
print(y_test[:5])
```

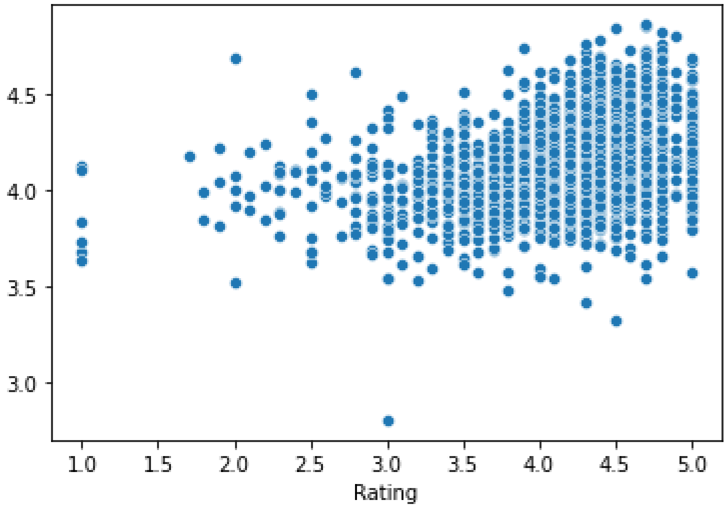
```
0.14189646461457917
[4.19379214 4.17665423 4.02503399 4.27675432 4.34189528]
7792    4.1
9008    5.0
5584    4.0
1875    4.5
3740    4.5
Name: Rating, dtype: float64
```

In [276…

```
sns.scatterplot(y_test, y_test_pred)
```

Out[276…

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



In [ ]:

In this model, the R^2 value is 0.142 which is far away from 1 (not good). Also, the scatter plot of test values vs. predicted test values shows a lack of linear relation to eachother. Therefore, this model developed can be considered low accuracy.

In [ ]: