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.

In [1]:
```
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 [2]:
```
inp0 = pd.read_csv("googleplaystore.csv")
```

In [3]:
```
inp0.head()
```

Out[3]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art & Design	January 7, 2018	1.0.0	4.0.3 and up
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	0	Everyone	Art & Design;Pretend Play	January 15, 2018	2.0.0	4.0.3 and up
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+	Free	0	Everyone	Art & Design	August 1, 2018	1.2.4	4.0.3 and up
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000,000+	Free	0	Teen	Art & Design	June 8, 2018	Varies with device	4.2 and up
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100,000+	Free	0	Everyone	Art & Design;Creativity	June 20, 2018	1.1	4.4 and up

In [4]:
```
inp0.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 [5]:

```
# refer the ipynb file 16
inp0.isnull().sum()
```

Out[5]:
```
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 [6]:
```
# refer the ipynb file 16
# work with dropna

inp0.dropna(how ='any', inplace = True)
```

In [7]:
```
# refer the ipynb file 16
# recheck the number of missing values
inp0.isnull().sum()
```

Out[7]:
```
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

In [8]:
```
inp0.shape
```

Out[8]:
```
(9360, 13)
```

Change variable to correct types

In [9]:
```
inp0.dtypes
```

Out[9]:
```
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
```

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

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

In [10]:
```
inp0.Price.describe()
```

Out[10]:
```
count     9360
unique      73
top          0
freq      8715
Name: Price, dtype: object
```

In [11]:
```
inp0.Price.value_counts()[:5]
```

Out[11]:
```
0        8715
$2.99     114
$0.99     106
$4.99      70
$1.99      59
Name: Price, 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

In [12]:
```
# Write a function named 'clean_price' if price is 0 it remains 0 otherwise delete the $

# delete the $ = removing the element at index 0

'$200'[1:] # example

# use map to apply the function to the column as shown in the next line

def clean_price(x):
    if '$' in x:
        x = x[1:]
        x = float(x)
        return(x)
    elif x == 0:
        x = float(x)
        return x
    else:
        return float(x)
```

In [13]:
```
inp0['Price'] = inp0.Price.map(clean_price)
```

In [14]:
```
inp0.Price.describe()
```

Out[14]:
```
count    9360.000000
mean        0.961279
std        15.821640
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max       400.000000
Name: Price, dtype: float64
```

4.2 Converting reviews to numeric

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

In [15]:
```
inp0.Reviews.describe() # object == categorical variable
```

Out[15]:
```
count     9360
unique    5990
top          2
freq        83
Name: Reviews, dtype: object
```

In [16]:
```
inp0.Reviews = inp0.Reviews.astype("int32")
```

In [17]:
```
inp0.Reviews.describe()
```

Out[17]:
```
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

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

In [18]:
```
inp0.Installs.describe() # object == categorical variable
```

Out[18]:
```
count           9360
unique            19
top       1,000,000+
freq            1576
Name: Installs, dtype: object
```

In [19]:
```
inp0.Installs.value_counts()
```

Out[19]:
```
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 [20]:
```python
# define a function 'clean_installs' where replace(",","") and replace("+","")

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

In [21]:
```python
# use map to apply the function to the column as shown earlier
inp0.Installs = inp0.Installs.map(clean_installs)
```

In [22]:
```python
inp0.Installs.describe()
```

Out[22]:
```
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
```

4.1 Handling the app size field

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
```

In [23]:
```
inp0.Size.describe() # object == categorical variable
```

Out[23]:
```
count                   9360
unique                   413
top       Varies with device
freq                    1637
Name: Size, dtype: object
```

In [24]:
```
# write a function 'change_size',
# if there is M which is size in MB, delete the last element, mutiply it with 1000 and convert it to float
# if there is k which is size in kB, delete the last element and convert it to float
# otherwise return None

def change_size(size):
    if 'M' in size:
        x = size[:-1]
        x = float(x)*1000
        return(x)
    elif 'k' == size[-1:]:
        x = size[:-1]
        x = float(x)
        return(x)
```

```
    else:
        return None
```

In [25]:
```
change_size("19k")
```

Out[25]:
```
19.0
```

In [26]:
```
# use map to apply the function to the column as shown earlier
inp0["Size"] = inp0["Size"].map(change_size)
```

In [27]:
```
inp0.Size.describe()
```

Out[27]:
```
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 [28]:
```
inp0["Size"].isnull().sum()
```

Out[28]:
```
1637
```

In [29]:
```
#filling Size which had NA
inp0.Size.fillna(method = 'ffill', inplace = True)
```

In [30]:
```
inp0.dtypes
```

Out[30]:
```
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 [31]:
```
# work wih describe.Describe()
inp0.Rating.describe()
```

Out[31]:
```
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 [32]:
```
# check for how many rows reviews are more than installs.
# print(inp0.Reviews.count())
# inp0.Installs.count()
inp0[inp0.Reviews <= inp0.Installs].count()

inp0 = inp0[inp0.Reviews <= inp0.Installs]
```

In [33]:
```
inp0[inp0.Reviews > inp0.Installs]
```

Out[33]:

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

In [34]:
```
# retain that part of data where reviews are less than installs
```

In [35]:
```
inp0.shape
```

Out[35]:
```
(9353, 13)
```

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

In [36]:
```
len(inp0[(inp0.Type == "Free") & (inp0.Price>0)])
```

Out[36]:
```
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?

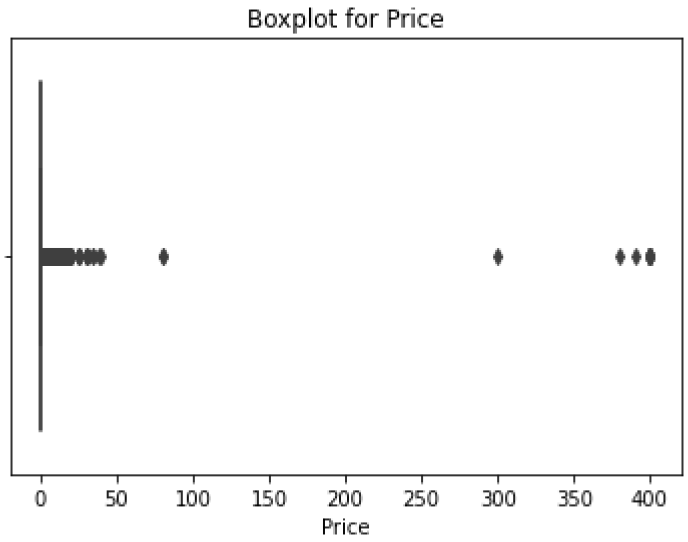
- Histogram for Size

Note down your observations for the plots made. 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.

In [37]:
```
sns.boxplot(x = inp0.Price).set(title = "Boxplot for Price");
```



In [82]:
```
inp0["Price"].describe()
```

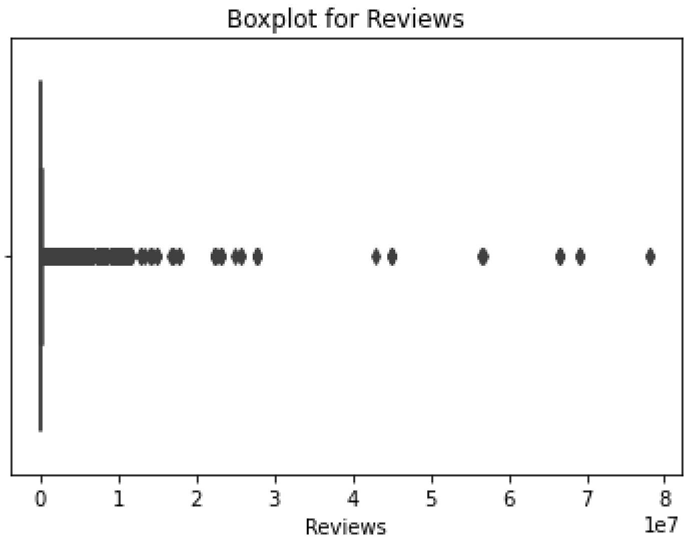
Out[82]:
```
count    9353.000000
mean        0.961467
std        15.827539
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max       400.000000
Name: Price, dtype: float64
```

usually the apps are free and very few are paid

Box plot for Reviews

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

In [47]:
```
sns.boxplot(x = inp0.Reviews).set(title = "Boxplot for Reviews");
```



In [71]:
```
min_value = inp0['Reviews'].min()
Q1 = inp0['Reviews'].quantile(0.25)
median_value = inp0['Reviews'].median()
Q3 = inp0['Reviews'].quantile(0.75)
max_value = inp0['Reviews'].max()


print ("min_value    :", min_value)
print ("Q1           :", Q1)
print ("median_value :", median_value)
print ("Q3           :", Q3)
print ("max_value    :", max_value)

IQR = Q3 - Q1
print("IQR:",round(IQR,2))

lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR

# data points less than lower_limit are outliers
# data points greater than upper limit are outliers

print ("Lower Limit",lower_limit)
print ("Upper Limit",upper_limit)
```

```
min_value    : 1
Q1           : 187.0
median_value : 5967.0
Q3           : 81747.0
max_value    : 78158306
IQR: 81560.0
Lower Limit -122153.0
Upper Limit 204087.0
```

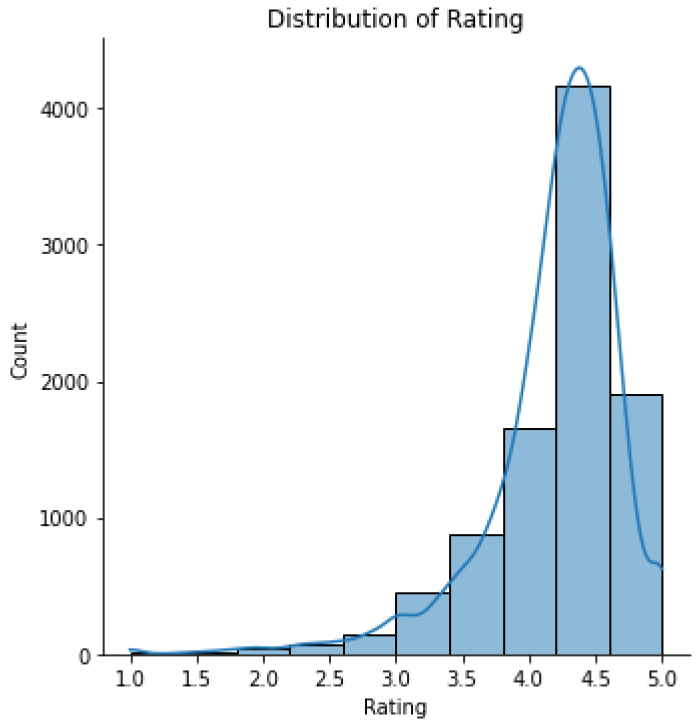
In [88]:
```
Q4 = inp0['Reviews'].quantile(.83)
print(Q4)
Q4 = inp0['Reviews'].quantile(.99)
print(Q4)

print("17% of the apps have reviews more than the upper limit and VERY FEW of the apps have very high reviews")
```

```
215301.0
9882988.44
17% of the apps have reviews more than the upper limit and VERY FEW of the apps have very high reviews
```

Histogram for Rating

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

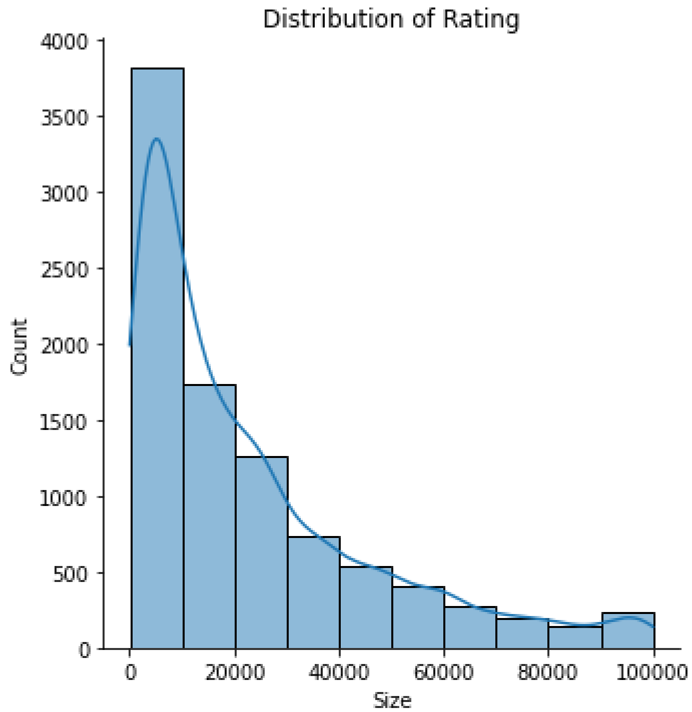
In [165…
```
sns.displot(inp0['Rating'], bins = 10, kde = True). set(title = "Distribution of Rating");
```



Histogram of Size

In [166…

```
sns.displot(inp0['Size'], bins = 10, kde = True). set(title = "Distribution of Rating");
```



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 [167…

```
# check for how many rows Price > 200?
inp0[inp0.Price > 200].count()
```

Out[167]:

```
App               15
Category          15
Rating            15
Reviews           15
Size              15
Installs          15
Type              15
Price             15
Content Rating    15
Genres            15
Last Updated      15
Current Ver       15
Android Ver       15
dtype: int64
```

In [168…

```
inp0[inp0.Price > 200]
```

Out[168]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
4197	most expensive app (H)	FAMILY	4.3	6	1500.0	100	Paid	399.99	Everyone	Entertainment	July 16, 2018	1	7.0 and up
4362	💎 I'm rich	LIFESTYLE	3.8	718	26000.0	10000	Paid	399.99	Everyone	Lifestyle	March 11, 2018	1.0.0	4.4 and up
4367	I'm Rich - Trump Edition	LIFESTYLE	3.6	275	7300.0	10000	Paid	400.00	Everyone	Lifestyle	May 3, 2018	1.0.1	4.1 and up
5351	I am rich	LIFESTYLE	3.8	3547	1800.0	100000	Paid	399.99	Everyone	Lifestyle	January 12, 2018	2	4.0.3 and up
5354	I am Rich Plus	FAMILY	4.0	856	8700.0	10000	Paid	399.99	Everyone	Entertainment	May 19, 2018	3	4.4 and up
5355	I am rich VIP	LIFESTYLE	3.8	411	2600.0	10000	Paid	299.99	Everyone	Lifestyle	July 21, 2018	1.1.1	4.3 and up
5356	I Am Rich Premium	FINANCE	4.1	1867	4700.0	50000	Paid	399.99	Everyone	Finance	November 12, 2017	1.6	4.0 and up
5357	I am extremely Rich	LIFESTYLE	2.9	41	2900.0	1000	Paid	379.99	Everyone	Lifestyle	July 1, 2018	1	4.0 and up
5358	I am Rich!	FINANCE	3.8	93	22000.0	1000	Paid	399.99	Everyone	Finance	December 11, 2017	1	4.1 and up
5359	I am rich(premium)	FINANCE	3.5	472	965.0	5000	Paid	399.99	Everyone	Finance	May 1, 2017	3.4	4.4 and up
5362	I Am Rich Pro	FAMILY	4.4	201	2700.0	5000	Paid	399.99	Everyone	Entertainment	May 30, 2017	1.54	1.6 and up
5364	I am rich (Most expensive app)	FINANCE	4.1	129	2700.0	1000	Paid	399.99	Teen	Finance	December 6, 2017	2	4.0.3 and up
5366	I Am Rich	FAMILY	3.6	217	4900.0	10000	Paid	389.99	Everyone	Entertainment	June 22, 2018	1.5	4.2 and up
5369	I am Rich	FINANCE	4.3	180	3800.0	5000	Paid	399.99	Everyone	Finance	March 22, 2018	1	4.2 and up
5373	I AM RICH PRO PLUS	FINANCE	4.0	36	41000.0	1000	Paid	399.99	Everyone	Finance	June 25, 2018	1.0.2	4.1 and up

In [169…
```
inp0 = inp0[inp0.Price <= 200].copy()

inp0.shape
```

Out[169]:
```
(9338, 13)
```

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 [170…
```
inp0 = inp0[inp0.Reviews <= 2000000].copy()

inp0.shape
```

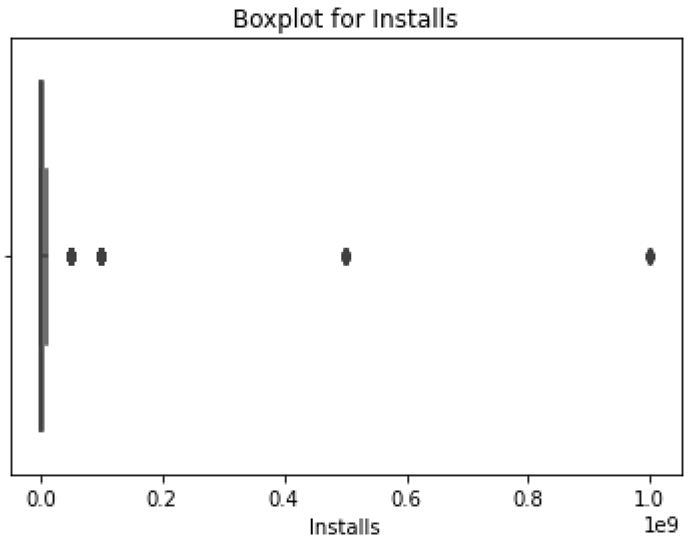
Out[170]:
```
(8885, 13)
```

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 [171…
```
sns.boxplot(x = inp0.Installs).set(title = "Boxplot for Installs");
```



In [172…
```
inp0.Installs.describe()
```

Out[172]:
```
count    8.885000e+03
mean     6.267379e+06
std      3.539960e+07
min      5.000000e+00
25%      1.000000e+04
50%      5.000000e+05
75%      5.000000e+06
max      1.000000e+09
Name: Installs, dtype: float64
```

In [173…
```
inp0.Installs.quantile([0.1, 0.25, 0.5, 0.70, 0.9, 0.95, 0.99])
```

Out[173]:
```
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 [174…

```
# check how many row have installs greater than corresponding to 99 percentile.
inp0[inp0.Installs > 100000000].count()
```

Out[174]:

```
App               20
Category          20
Rating            20
Reviews           20
Size              20
Installs          20
Type              20
Price             20
Content Rating    20
Genres            20
Last Updated      20
Current Ver       20
Android Ver       20
dtype: int64
```

In [175…

```
# retain installs less than corresponding to 99 percentile. check shape
inp0 = inp0[inp0.Installs < 100000000].copy()

inp0.shape
```

Out[175]:

```
(8743, 13)
```

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?
```

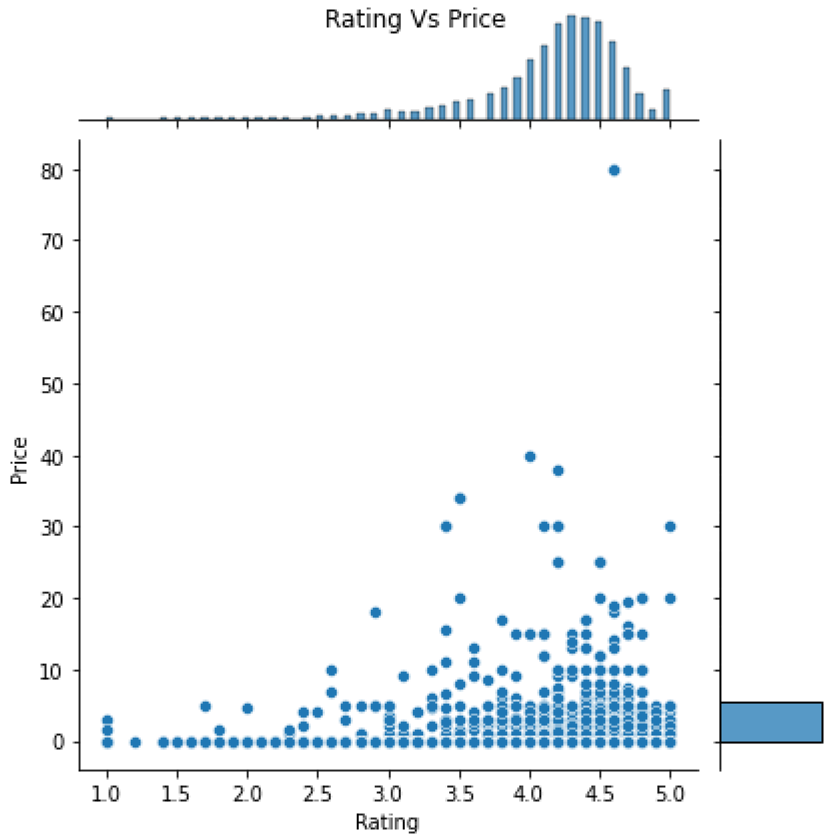
For each of the plots above, note down your observation.

7.1. Make scatter plot/joinplot for Rating vs Price

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

In [176…

```
plot = sns.jointplot(x = inp0.Rating, y = inp0.Price, kind = 'scatter');
plot.fig.suptitle("Rating Vs Price");
```



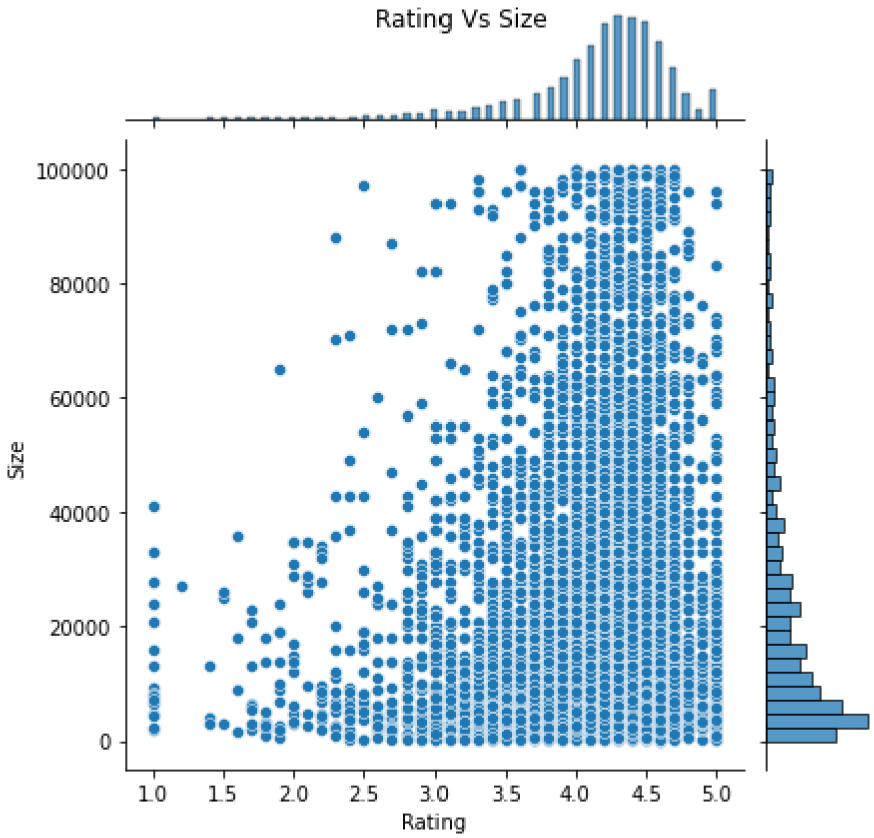
Paid apps have higher rating in comparison to free apps

7.2 Make scatter plot/joinplot for Rating vs Size

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

In [177…

```
plot = sns.jointplot(x = inp0.Rating, y = inp0.Size, kind = 'scatter');
plot.fig.suptitle("Rating Vs Size");
```



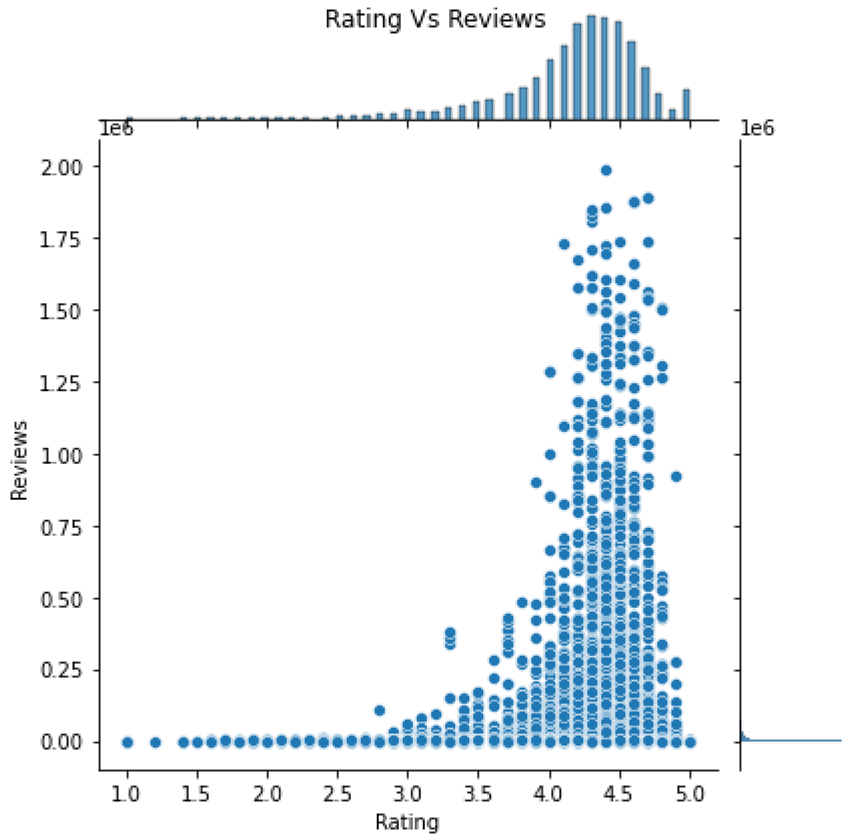
The heavier apps are rated better

7.3 Make scatter plot/joinplot for Rating vs Reviews

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

In [178…

```
plot = sns.jointplot(x = inp0.Rating, y = inp0.Reviews, kind = 'scatter');
plot.fig.suptitle("Rating Vs Reviews");
```



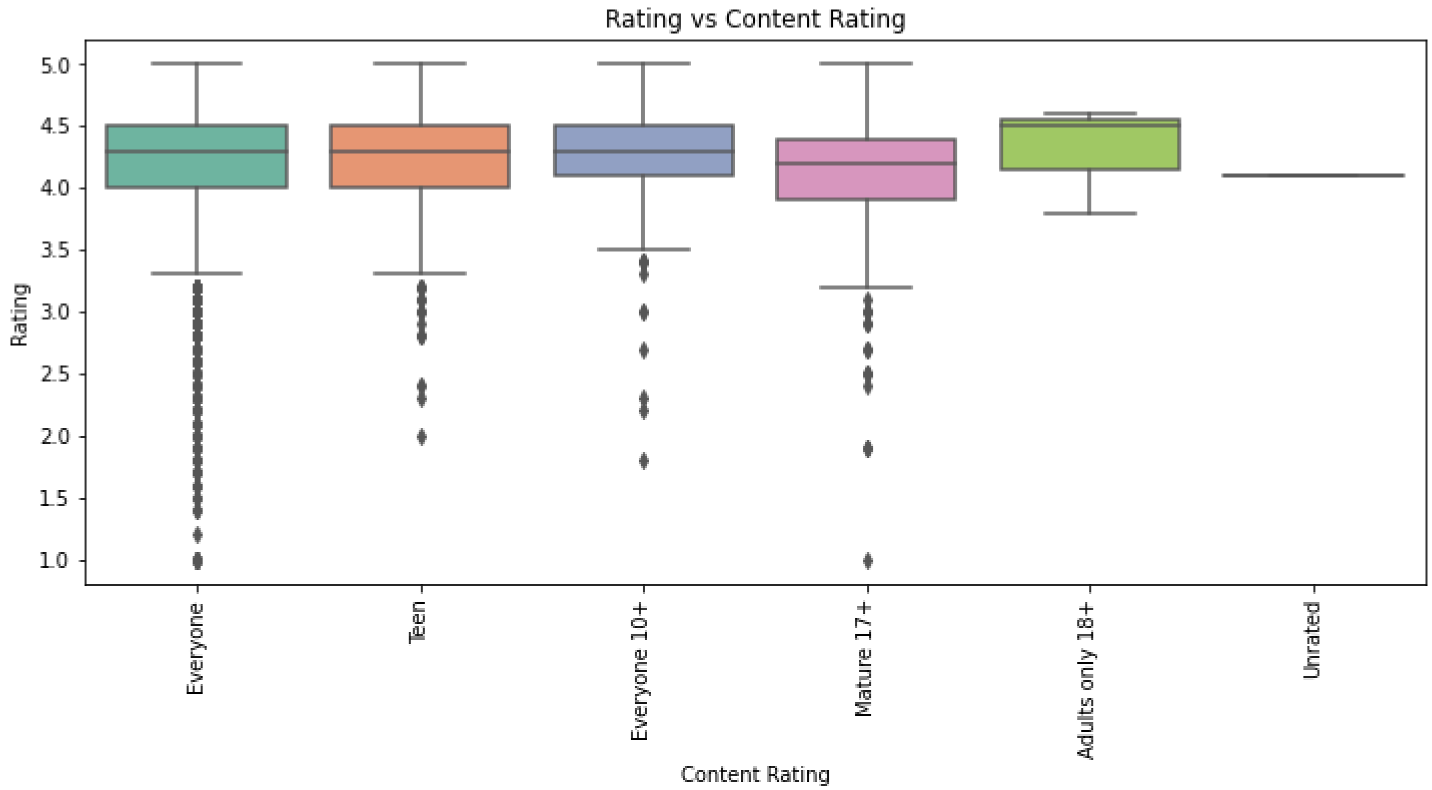
Rating is higher for apps with higer Reviews

7.4 Make boxplot for Rating vs Content Rating

```
a.      Is there any difference in the ratings? Are some types liked better?
```

In [179…

```
plt.figure(figsize=[12,5])
sns.boxplot(y = inp0.Rating, x = inp0['Content Rating'], palette ='Set2').set(title = "Rating vs Content Rating");
plt.xticks(rotation=90);
```



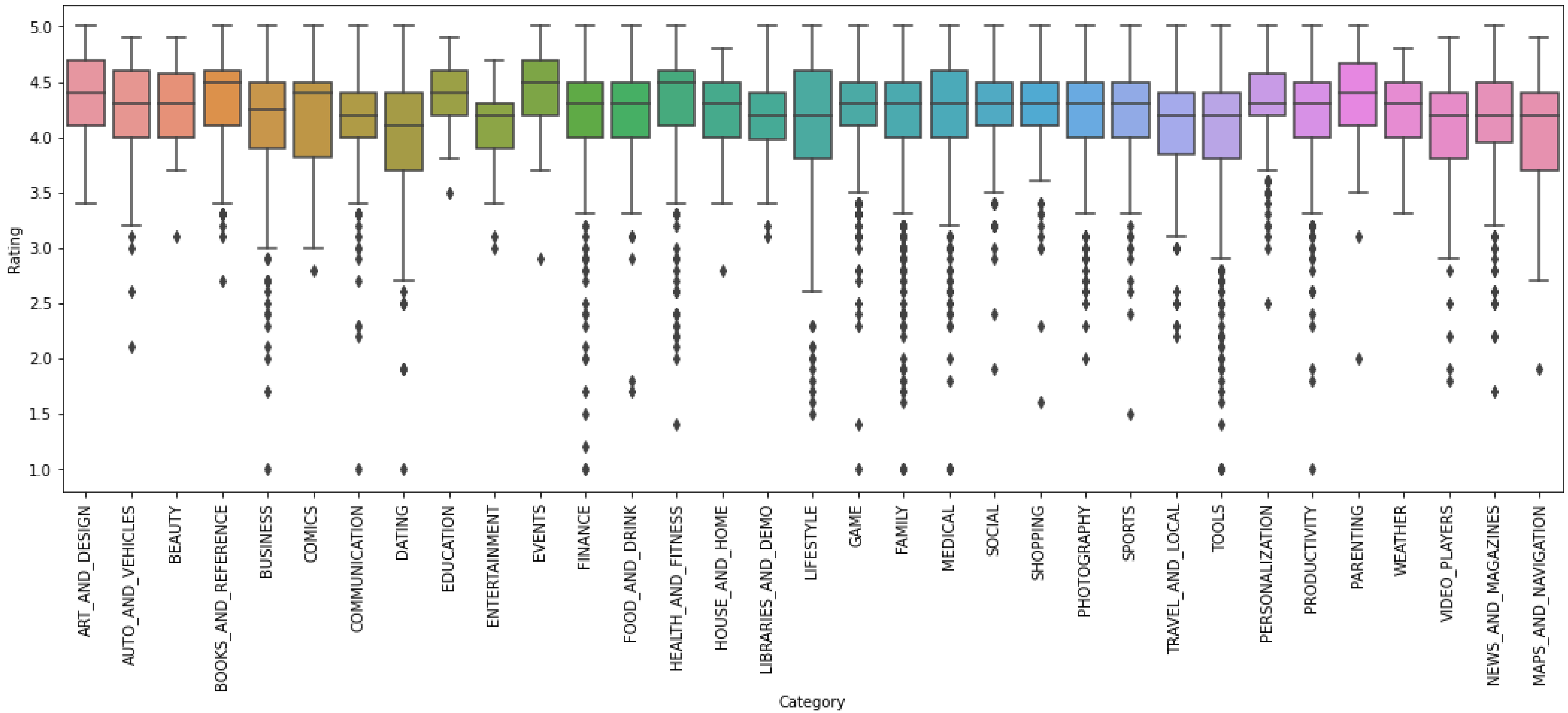
Apps for Everyone have more outliers and bad ratings while apps for Adults have better ratings and no outliers

7.5 Make boxplot for Ratings vs. Category

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

In [180…

```
plt.figure(figsize=[18,6])
g = sns.boxplot(x=inp0.Category, y=inp0.Rating, data=inp0);
plt.xticks(rotation=90);
```



Parenting, Events and Ars and Design have good ratings

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 [181…

```
inp1 = inp0.copy()
```

In [182…

```
inp1.skew()
```

Out[182]:

```
Rating      -1.777070
Reviews      4.149314
Size         1.488843
Installs     4.401787
Price       16.495052
dtype: float64
```

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 [183…

```
# check describe for installs
inp1.Installs.describe()
```

Out[183]:
```
count    8.743000e+03
mean     3.486865e+06
std      8.659419e+06
min      5.000000e+00
25%      1.000000e+04
50%      1.000000e+05
75%      5.000000e+06
max      5.000000e+07
Name: Installs, dtype: float64
```

In [184…
```
inp1.Installs = inp1.Installs.apply(np.log1p)
```

In [185…
```
inp1.Installs.skew()
```

Out[185]:
```
-0.46306064681638187
```

In [186…
```
# do same for reviews
inp1.Reviews.describe()
```

Out[186]:
```
count    8.743000e+03
mean     8.957859e+04
std      2.320521e+05
min      1.000000e+00
25%      1.490000e+02
50%      3.878000e+03
75%      5.023650e+04
max      1.986068e+06
Name: Reviews, dtype: float64
```

In [187…
```
inp1.Reviews = inp1.Reviews.apply(np.log1p)
```

In [119…
```
inp1.Reviews.skew()
```

Out[119]:
```
-0.1911443092583795
```

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

In [188…
```
inp1.drop(["App", "Last Updated", "Current Ver", "Android Ver"], axis=1, inplace=True)
```

In [189…

```
inp1.shape
```

Out[189]:

```
(8743, 9)
```

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 [190…

```
# check types
inp2 = inp1.copy()
inp2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8743 entries, 0 to 10840
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Category        8743 non-null   object 
 1   Rating          8743 non-null   float64
 2   Reviews         8743 non-null   float64
 3   Size            8743 non-null   float64
 4   Installs        8743 non-null   float64
 5   Type            8743 non-null   object 
 6   Price           8743 non-null   float64
 7   Content Rating  8743 non-null   object 
 8   Genres          8743 non-null   object 
dtypes: float64(5), object(4)
memory usage: 683.0+ KB
```

In [191…

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

In [192…

```
# display col names
inp2.columns
```

Out[192]:
```
Index(['Rating', 'Reviews', 'Size', 'Installs', 'Price',
       'Category_AUTO_AND_VEHICLES', 'Category_BEAUTY',
       'Category_BOOKS_AND_REFERENCE', 'Category_BUSINESS', 'Category_COMICS',
       ...
       'Genres_Tools', 'Genres_Tools;Education', 'Genres_Travel & Local',
       'Genres_Travel & Local;Action & Adventure', 'Genres_Trivia',
       'Genres_Video Players & Editors',
       'Genres_Video Players & Editors;Creativity',
       'Genres_Video Players & Editors;Music & Video', 'Genres_Weather',
       'Genres_Word'],
      dtype='object', length=157)
```

In [193…
```
display(inp2.head())
```

	Rating	Reviews	Size	Installs	Price	Category_AUTO_AND_VEHICLES	Category_BEAUTY	Category_BOOKS_AND_REFERENCE	Category_BUSINESS
0	4.1	5.075174	19000.0	9.210440	0.0	0	0	0	(
1	3.9	6.875232	14000.0	13.122365	0.0	0	0	0	(
2	4.7	11.379520	8700.0	15.424949	0.0	0	0	0	(
3	4.5	12.281389	25000.0	17.727534	0.0	0	0	0	(
4	4.3	6.875232	2800.0	11.512935	0.0	0	0	0	(

5 rows × 157 columns

In [200…
```
inp2.shape
```

Out[200]:
```
(8743, 157)
```

In [199…
```
pd.set_option('display.max_columns', None) # to see all the columns in the df
```

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

Train - test split

In [201…
```
from sklearn.model_selection import train_test_split
```

?train_test_split

In [202…
```
df_train, df_test = train_test_split(inp2, train_size = 0.7, random_state = 5)
```

In [203…
```
df_train.shape, df_test.shape
```

Out[203]:
```
((6120, 157), (2623, 157))
```

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

In [204…
```
y_train = df_train.pop("Rating")
X_train = df_train
```

In [221…
```
y_train
```

Out[221]:
```
5705     3.3
2981     4.3
8381     4.2
10045    4.1
1822     4.3
        ...
399      4.3
81       4.4
9869     3.4
8516     3.8
6791     3.8
Name: Rating, Length: 6120, dtype: float64
```

In [210…
```
y_test = df_test.pop("Rating")
X_test = df_test
```

In [211…
```
print(X_train.shape)
print(y_train.shape)
```

```
print(X_test.shape)
print(y_test.shape)
```

```
(6120, 156)
(6120,)
(2623, 156)
(2623,)
```

Build the model

11 . Model building

In [224…

```
X_train.columns
```

Out[224]:

```
Index(['Reviews', 'Size', 'Installs', 'Price', 'Category_AUTO_AND_VEHICLES',
       'Category_BEAUTY', 'Category_BOOKS_AND_REFERENCE', 'Category_BUSINESS',
       'Category_COMICS', 'Category_COMMUNICATION',
       ...
       'Genres_Tools', 'Genres_Tools;Education', 'Genres_Travel & Local',
       'Genres_Travel & Local;Action & Adventure', 'Genres_Trivia',
       'Genres_Video Players & Editors',
       'Genres_Video Players & Editors;Creativity',
       'Genres_Video Players & Editors;Music & Video', 'Genres_Weather',
       'Genres_Word'],
      dtype='object', length=156)
```

In [212…

```
from sklearn.linear_model import LinearRegression
# instantiate
linreg = LinearRegression()

# fit the model to the training data
linreg.fit(X_train, y_train)                    # training the model on training data.

# print the intercept and coefficients
print (round(linreg.intercept_,3)) # B0
print (np.round(linreg.coef_,3)) # B1, B2 and B3 etc
```

```
4.382
[ 0.169 -0.    -0.144 -0.002  0.177  0.27   0.236  0.146  0.448  0.106
  0.058  0.071  0.018  0.307  0.11   0.106  0.148  0.299  0.176  0.198
  0.194  0.119  0.117  0.202  0.126  0.25   0.215  0.119  0.146  0.162
  0.154  0.226  0.191  0.199  0.075  0.169 -0.06  -0.116 -0.109 -0.149
 -0.123 -0.     0.21  -0.087  0.135  0.448 -0.049  0.019  0.383  0.443
  0.658  0.408  0.253  0.177  0.27   0.053 -0.01   0.347  0.895  0.236
 -0.307  0.146 -0.065 -0.079  0.     0.089  0.068  0.109  0.469  0.312
  0.188  0.265  0.12  -0.143  0.59   0.106 -0.     0.058  0.358  0.452
  0.26   0.633  0.387  0.228  0.59  -0.078  0.456  0.203  0.498  0.359
  0.251  0.156  0.317  0.379  0.537  0.509  0.249  0.127  0.307  0.106
  0.148  0.176 -0.206  0.446  0.198  0.194  0.153  0.22  -0.034  0.117
  0.202 -0.165 -0.     0.46   0.126  0.312 -0.114 -0.05   0.101  0.215
  0.119  0.146  0.255  0.147  0.315  0.     0.751 -0.156  0.318  0.
  0.058  0.245  0.245 -0.067  0.162  0.099  0.375  0.163  0.131  0.154
  0.064  0.236  0.007  0.57   0.     0.737 -0.014  0.205  0.054  0.145
  0.074  0.089 -0.076  0.011  0.169  0.072]
```

In [217…

```
R2_train = round(linreg.score(X_train,y_train),3)
print("The R2 value of the Training Set is :{}" .format(R2_train))
```

```
The R2 value of the Training Set is :0.163
```

In [223…

```
R2_test = round(linreg.score(X_test,y_test),3)
print("The R2 value of the Testing Set is :{}" .format(R2_test))
```

```
The R2 value of the Testing Set is :0.14
```

12. Make predictions on test set and report R2.

In [218…

```
# make predictions on the testing data
y_pred = linreg.predict(X_test)
y_pred[:5]
```

Out[218]:

```
array([3.95592582, 4.2206979 , 4.12975991, 4.06385956, 4.20130229])
```

In [219…

```
y_test[:5]
```

Out[219]:
```
10304    4.0
779      4.2
6210     4.3
4290     4.8
827      4.0
Name: Rating, dtype: float64
```

In [220…
```
from sklearn.metrics import r2_score
print (r2_score(y_pred, y_test))
```

```
-4.503382679774025
```

Model Evaluation

RMSE - Root Mean Squared Error

In [221…
```
from sklearn.metrics import mean_squared_error


print (mean_squared_error(y_pred, y_test)) # MSE
round (mean_squared_error(y_pred, y_test, squared=False),3) # RMSE
```

```
0.24970016698925357
```

Out[221]:
```
0.5
```

R_squared

In [222…
```
from sklearn.metrics import r2_score
print (r2_score(y_pred, y_test)) # test data
```

```
-4.503382679774025
```