1. Import required libraries and read the dataset.

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv('Apps_data+(1).csv')
```

2. Check the first few samples, shape, info of the data and try to familiarize yourself with different features.

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

Out[3]:

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

In [4]:
```
df.shape     #---The dataset has 10841 Rows and 13 Columns.
```

Out[4]:
```
(10841, 13)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10841 entries, 0 to 10840
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   App             10841 non-null  object 
 1   Category        10841 non-null  object 
 2   Rating          9367 non-null   float64
 3   Reviews         10841 non-null  object 
 4   Size            10841 non-null  object 
 5   Installs        10841 non-null  object 
 6   Type            10840 non-null  object 
 7   Price           10841 non-null  object 
 8   Content Rating  10840 non-null  object 
 9   Genres          10841 non-null  object 
 10  Last Updated    10841 non-null  object 
 11  Current Ver     10833 non-null  object 
 12  Android Ver     10838 non-null  object 
dtypes: float64(1), object(12)
memory usage: 1.1+ MB
```

3. Check summary statistics of the dataset. List out the columns that need to be worked upon for model building.

In [6]:
```
df.describe(include='object')
```

Out[6]:

	App	Category	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
count	10841	10841	10841	10841	10841	10840	10841	10840	10841	10841	10833	10838
unique	9660	34	6002	462	22	3	93	6	120	1378	2832	33
top	ROBLOX	FAMILY	0	Varies with device	1,000,000+	Free	0	Everyone	Tools	August 3, 2018	Varies with device	4.1 and up
freq	9	1972	596	1695	1579	10039	10040	8714	842	326	1459	2451

In [7]:
```
df.describe()
```

Out[7]:

	Rating
count	9367.000000
mean	4.193338
std	0.537431
min	1.000000
25%	4.000000
50%	4.300000
75%	4.500000
max	19.000000

In [8]:
```
df.isnull().sum()    #---There are null values present in the Dataset.
```

Out[8]:
```
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 [9]:
```
df.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. Check if there are any duplicate records in the dataset? if any drop them.

In [10]:
```
df.duplicated().sum()   #---There are Duplicate Values in the Dataset.
```

Out[10]:
```
483
```

In [11]:
```
df.drop_duplicates(inplace= True)
df.duplicated().sum()
```

Out[11]:
```
0
```

In [12]:
```
df.shape
```

Out[12]:
```
(10358, 13)
```

5. Check the unique categories of the column 'Category', Is there any invalid category? If yes, drop them.

In [13]:
```
df['Category'].unique()
```

Out[13]:
```
array(['ART_AND_DESIGN', 'AUTO_AND_VEHICLES', 'BEAUTY',
       'BOOKS_AND_REFERENCE', 'BUSINESS', 'COMICS', 'COMMUNICATION',
       'DATING', 'EDUCATION', 'ENTERTAINMENT', 'EVENTS', 'FINANCE',
       'FOOD_AND_DRINK', 'HEALTH_AND_FITNESS', 'HOUSE_AND_HOME',
       'LIBRARIES_AND_DEMO', 'LIFESTYLE', 'GAME', 'FAMILY', 'MEDICAL',
       'SOCIAL', 'SHOPPING', 'PHOTOGRAPHY', 'SPORTS', 'TRAVEL_AND_LOCAL',
       'TOOLS', 'PERSONALIZATION', 'PRODUCTIVITY', 'PARENTING', 'WEATHER',
       'VIDEO_PLAYERS', 'NEWS_AND_MAGAZINES', 'MAPS_AND_NAVIGATION',
       '1.9'], dtype=object)
```

In [14]:
```
df[df['Category']=='1.9']
```

Out[14]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
10472	Life Made WI-Fi Touchscreen Photo Frame	1.9	19.0	3.0M	1,000+	Free	0	Everyone	NaN	February 11, 2018	1.0.19	4.0 and up	NaN

In [15]:
```
df= df.drop(df[df['Category']=='1.9'].index)
```

In [16]:
```
df[df['Category']=='1.9']
```

Out[16]:

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

6. Check if there are missing values present in the column Rating, If any? drop them and and create a new column as 'Rating_category' by converting ratings to high and low categories(>3.5 is high rest low)

In [17]:
```
df['Rating'].isnull().sum()
```

Out[17]:
```
1465
```

In [18]:
```
df.dropna(inplace=True)
```

In [19]:
```
df.isnull().sum()
```

Out[19]:
```
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 [20]:
```
df.shape
```

Out[20]:
```
(8886, 13)
```

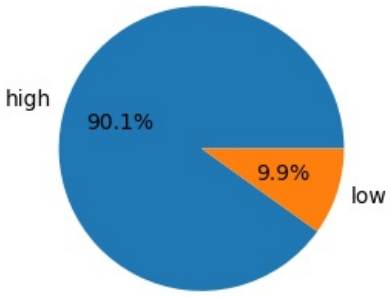
In [21]:
```
df['Rating_category']= df['Rating'].apply(lambda a: 'high' if a>3.5 else 'low')
```

In [22]:
```
df['Rating_category']
```

Out[22]:
```
0        high
1        high
2        high
3        high
4        high
         ...
10834    high
10836    high
10837    high
10839    high
10840    high
Name: Rating_category, Length: 8886, dtype: object
```

7. Check the distribution of the newly created column 'Rating_category' and comment on the distribution.

In [23]:
```
plt.figure(figsize=(5,3))
plt.pie(df['Rating_category'].value_counts(),labels=df.groupby(['Rating_category'])['Rating_category'].count().
```



Comments:

- Ratings are split into two groups: 'high' and 'low'.
- 'High' ratings are those above 3.5, while the rest are 'low' ratings.
- In our data distribution, most ratings (90%) fall under 'high', and only a small portion (10%) are 'low'.

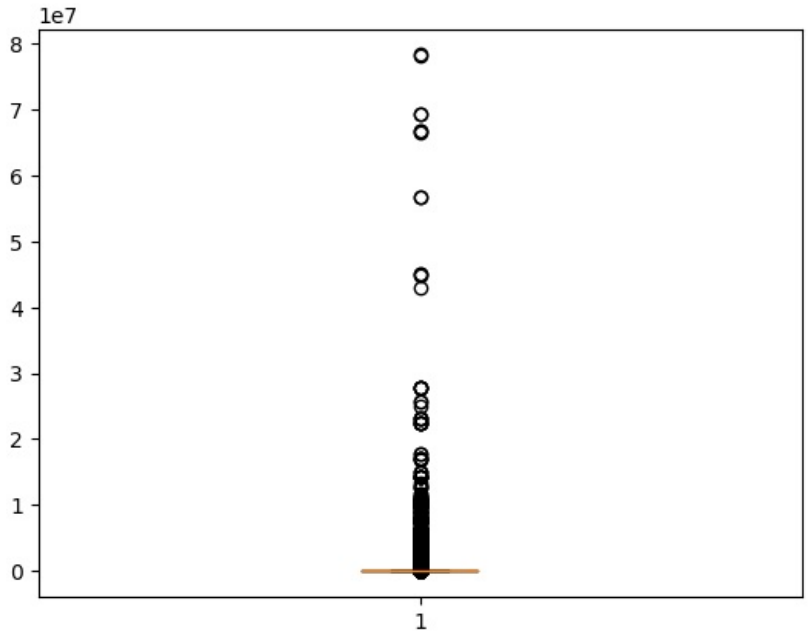
8. Convert the column "Reviews" to numeric data type and check the presence of outliers in the column and handle the outliers using a transformation approach.(Hint: Use log transformation)

In [24]:
```
df['Reviews']=df['Reviews'].astype('int')
```

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

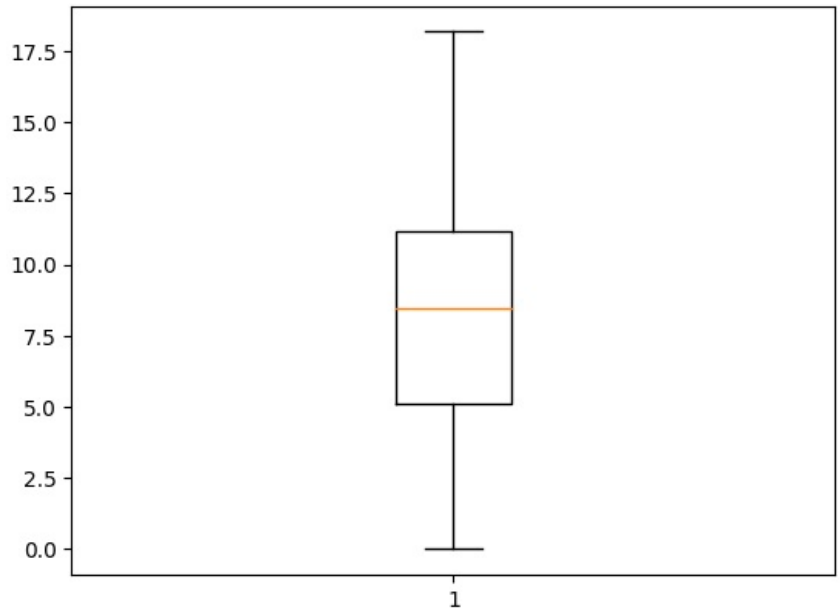
Out[25]:
```
App                object
Category           object
Rating            float64
Reviews             int32
Size               object
Installs           object
Type               object
Price              object
Content Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
Rating_category    object
dtype: object
```

In [26]:
```
plt.boxplot(df['Reviews']);
```



In [27]:
```
df['Reviews']= np.log(df['Reviews'])
```

In [28]:
```
plt.boxplot(df['Reviews']);
```



9. The column 'Size' contains alphanumeric values, treat the non numeric data and convert the column into suitable data type. (hint: Replace M with 1 million and K with 1 thousand, and drop the entries where size='Varies with device')

In [29]:
```
df['Size'].value_counts().sort_index()
```

Out[29]:
```
Size
1.0M                  4
1.1M                 25
1.2M                 30
1.3M                 27
1.4M                 25
                   ...
986k                  1
98M                  13
994k                  1
99M                  37
Varies with device  1468
Name: count, Length: 413, dtype: int64
```

In [30]:
```
df = df.drop(df[df['Size']=='Varies with device'].index)
```

In [31]:
```
df[df['Size']=='Varies with device']
```

Out[31]:

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

In [32]:
```
def convert(value):
    value= value.replace('M','*1000000')
    value= value.replace('k','*1000')
    return eval(value)
```

In [33]:
```
df['Size']=df['Size'].apply(convert)
```

In [34]:
```
df['Size']
```

Out[34]:
```
0        19000000.0
1        14000000.0
2         8700000.0
3        25000000.0
4         2800000.0
             ...
10833      619000.0
10834     2600000.0
10836    53000000.0
10837     3600000.0
10840    19000000.0
Name: Size, Length: 7418, dtype: float64
```

In [35]:
```
df.dtypes
```

Out[35]:
```
App                object
Category           object
Rating            float64
Reviews           float64
Size              float64
Installs           object
Type               object
Price              object
Content Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
Rating_category    object
dtype: object
```

10. Check the column 'Installs', treat the unwanted characters and convert the column into a suitable data type.

In [36]:
```
df['Installs']
```

Out[36]:
```
0            10,000+
1           500,000+
2         5,000,000+
3        50,000,000+
4           100,000+
             ...
10833         1,000+
10834           500+
10836         5,000+
10837           100+
10840    10,000,000+
Name: Installs, Length: 7418, dtype: object
```

In [37]:
```
df['Installs']= df['Installs'].str.replace(',','')
df['Installs']= df['Installs'].str.replace('+','')
```

In [38]:
```
df['Installs'].unique()
```

Out[38]:
```
array(['10000', '500000', '5000000', '50000000', '100000', '50000',
       '1000000', '10000000', '5000', '100000000', '1000', '500000000',
       '100', '500', '10', '1000000000', '5', '50', '1'], dtype=object)
```

In [39]:
```
df['Installs']=df['Installs'].astype(int)
```

In [40]:
```
df.dtypes
```

Out[40]:
```
App                object
Category           object
Rating            float64
Reviews           float64
Size              float64
Installs            int32
Type               object
Price              object
Content Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
Rating_category    object
dtype: object
```

11. Check the column 'Price' , remove the unwanted characters and convert the column into a suitable data type.

In [41]:
```
df['Price'].unique()
```

Out[41]:
```
array(['0', '$4.99', '$6.99', '$7.99', '$3.99', '$5.99', '$2.99', '$1.99',
       '$9.99', '$0.99', '$9.00', '$5.49', '$10.00', '$24.99', '$11.99',
       '$79.99', '$16.99', '$14.99', '$29.99', '$12.99', '$3.49',
       '$10.99', '$7.49', '$1.50', '$19.99', '$15.99', '$33.99', '$39.99',
       '$2.49', '$4.49', '$1.70', '$1.49', '$3.88', '$399.99', '$17.99',
       '$400.00', '$3.02', '$1.76', '$4.84', '$4.77', '$1.61', '$1.59',
       '$6.49', '$1.29', '$299.99', '$379.99', '$37.99', '$18.99',
       '$389.99', '$8.49', '$1.75', '$14.00', '$2.00', '$3.08', '$2.59',
       '$19.40', '$15.46', '$8.99', '$3.04', '$13.99', '$4.29', '$3.28',
       '$4.60', '$1.00', '$2.90', '$1.97', '$2.56', '$1.20'], dtype=object)
```

In [42]:
```
df['Price']= df['Price'].str.replace('$','').astype(float)
```

In [43]:
```
df['Price'].unique()
```

Out[43]:
```
array([  0.  ,   4.99,   6.99,   7.99,   3.99,   5.99,   2.99,   1.99,
         9.99,   0.99,   9.  ,   5.49,  10.  ,  24.99,  11.99,  79.99,
        16.99,  14.99,  29.99,  12.99,   3.49,  10.99,   7.49,   1.5 ,
        19.99,  15.99,  33.99,  39.99,   2.49,   4.49,   1.7 ,   1.49,
         3.88, 399.99,  17.99, 400.  ,   3.02,   1.76,   4.84,   4.77,
         1.61,   1.59,   6.49,   1.29, 299.99, 379.99,  37.99,  18.99,
       389.99,   8.49,   1.75,  14.  ,   2.  ,   3.08,   2.59,  19.4 ,
        15.46,   8.99,   3.04,  13.99,   4.29,   3.28,   4.6 ,   1.  ,
         2.9 ,   1.97,   2.56,   1.2 ])
```

12. Drop the columns which you think redundant for the analysis.(suggestion: drop column 'rating', since we created a new feature from it (i.e. rating_category) and the columns 'App', 'Rating' ,'Genres','Last Updated', 'Current Ver','Android Ver' columns since which are redundant for our analysis)

In [44]:
```
df.columns
```

Out[44]:
```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type',
       'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver',
       'Android Ver', 'Rating_category'],
      dtype='object')
```

In [45]:
```
df =df.drop(columns=['Rating','App', 'Rating' ,'Genres','Last Updated', 'Current Ver','Android Ver'])
```

In [46]:
```
df
```

Out[46]:

	Category	Reviews	Size	Installs	Type	Price	Content Rating	Rating_category
0	ART_AND_DESIGN	5.068904	19000000.0	10000	Free	0.0	Everyone	high
1	ART_AND_DESIGN	6.874198	14000000.0	500000	Free	0.0	Everyone	high
2	ART_AND_DESIGN	11.379508	8700000.0	5000000	Free	0.0	Everyone	high
3	ART_AND_DESIGN	12.281384	25000000.0	50000000	Free	0.0	Teen	high
4	ART_AND_DESIGN	6.874198	2800000.0	100000	Free	0.0	Everyone	high
...	...	...	...	...	...	...	...	...
10833	BOOKS_AND_REFERENCE	3.784190	619000.0	1000	Free	0.0	Everyone	high
10834	FAMILY	1.945910	2600000.0	500	Free	0.0	Everyone	high
10836	FAMILY	3.637586	53000000.0	5000	Free	0.0	Everyone	high
10837	FAMILY	1.386294	3600000.0	100	Free	0.0	Everyone	high
10840	LIFESTYLE	12.894978	19000000.0	10000000	Free	0.0	Everyone	high

7418 rows × 8 columns

In [47]:
```
df.dtypes
```

Out[47]:
```
Category            object
Reviews            float64
Size               float64
Installs             int32
Type                object
Price              float64
Content Rating      object
Rating_category     object
dtype: object
```

13. Encode the categorical columns.

In [48]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [49]:
```
df['Category']= le.fit_transform(df['Category'])
```

In [50]:
```
df['Type']= le.fit_transform(df['Type'])
df['Content Rating']= le.fit_transform(df['Content Rating'])
df['Rating_category']= le.fit_transform(df['Rating_category'])
```

In [51]:
```
df
```

Out[51]:

	Category	Reviews	Size	Installs	Type	Price	Content Rating	Rating_category
0	0	5.068904	19000000.0	10000	0	0.0	1	0
1	0	6.874198	14000000.0	500000	0	0.0	1	0
2	0	11.379508	8700000.0	5000000	0	0.0	1	0
3	0	12.281384	25000000.0	50000000	0	0.0	4	0
4	0	6.874198	2800000.0	100000	0	0.0	1	0
...	...	...	...	...	...	...	...	...
10833	3	3.784190	619000.0	1000	0	0.0	1	0
10834	11	1.945910	2600000.0	500	0	0.0	1	0
10836	11	3.637586	53000000.0	5000	0	0.0	1	0
10837	11	1.386294	3600000.0	100	0	0.0	1	0
10840	18	12.894978	19000000.0	10000000	0	0.0	1	0

7418 rows × 8 columns

In [52]:
```
df.dtypes
```

Out[52]:
```
Category             int32
Reviews            float64
Size               float64
Installs             int32
Type                 int32
Price              float64
Content Rating       int32
Rating_category      int32
dtype: object
```

In [53]:
```
df
```

Out[53]:

	Category	Reviews	Size	Installs	Type	Price	Content Rating	Rating_category
0	0	5.068904	19000000.0	10000	0	0.0	1	0
1	0	6.874198	14000000.0	500000	0	0.0	1	0
2	0	11.379508	8700000.0	5000000	0	0.0	1	0
3	0	12.281384	25000000.0	50000000	0	0.0	4	0
4	0	6.874198	2800000.0	100000	0	0.0	1	0
...	...	...	...	...	...	...	...	...
10833	3	3.784190	619000.0	1000	0	0.0	1	0
10834	11	1.945910	2600000.0	500	0	0.0	1	0
10836	11	3.637586	53000000.0	5000	0	0.0	1	0
10837	11	1.386294	3600000.0	100	0	0.0	1	0
10840	18	12.894978	19000000.0	10000000	0	0.0	1	0

7418 rows × 8 columns

14. Segregate the target and independent features (Hint: Use Rating_category as the target)

In [54]:
```
x= df.drop(df[['Rating_category']],axis= 1)
x
```

Out[54]:

	Category	Reviews	Size	Installs	Type	Price	Content Rating
0	0	5.068904	19000000.0	10000	0	0.0	1
1	0	6.874198	14000000.0	500000	0	0.0	1
2	0	11.379508	8700000.0	5000000	0	0.0	1
3	0	12.281384	25000000.0	50000000	0	0.0	4
4	0	6.874198	2800000.0	100000	0	0.0	1
...	...	...	...	...	...	...	...
10833	3	3.784190	619000.0	1000	0	0.0	1
10834	11	1.945910	2600000.0	500	0	0.0	1
10836	11	3.637586	53000000.0	5000	0	0.0	1
10837	11	1.386294	3600000.0	100	0	0.0	1
10840	18	12.894978	19000000.0	10000000	0	0.0	1

7418 rows × 7 columns

In [55]:
```
y = df[['Rating_category']]
y
```

Out[55]:

	Rating_category
0	0
1	0
2	0
3	0
4	0
...	...
10833	0
10834	0
10836	0
10837	0
10840	0

7418 rows × 1 columns

15. Split the dataset into train and test.

In [56]:
```
from sklearn.model_selection import train_test_split
```

In [57]:
```
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.25,random_state=555)
```

In [58]:
```
x_train
```

Out[58]:

	Category	Reviews	Size	Installs	Type	Price	Content Rating
503	7	12.314425	21000000.0	1000000	0	0.0	3
768	8	6.486161	556000.0	10000	0	0.0	1
2575	27	10.003242	18000000.0	1000000	0	0.0	1
1215	13	6.320768	23000000.0	100000	0	0.0	4
1395	15	9.827632	57000000.0	1000000	0	0.0	1
...	...	...	...	...	...	...	...
4200	11	11.543105	5600000.0	5000000	0	0.0	1
1724	14	13.944142	59000000.0	50000000	0	0.0	4
10423	11	11.200746	46000000.0	5000000	0	0.0	4
7152	14	10.849784	36000000.0	500000	0	0.0	4
9571	18	10.936619	15000000.0	1000000	0	0.0	1

5563 rows × 7 columns

In [59]:
```
x_test
```

Out[59]:

	Category	Reviews	Size	Installs	Type	Price	Content Rating
9072	29	3.401197	22000000.0	5000	0	0.0	1
10635	21	4.488636	5700000.0	5000	0	0.0	1
7987	11	12.571441	21000000.0	5000000	0	0.0	4
1597	18	8.345693	20000000.0	1000000	0	0.0	1
1533	17	7.855545	20000000.0	100000	0	0.0	1
...	...	...	...	...	...	...	...
7842	11	3.761200	3300000.0	1000	0	0.0	1
5026	11	6.322565	6700000.0	10000	0	0.0	1
7819	14	6.484635	14000000.0	50000	0	0.0	4
9007	29	5.043425	16000000.0	10000	0	0.0	1
154	3	6.756932	13000000.0	100000	0	0.0	1

1855 rows × 7 columns

In [60]:
```
y_train
```

Out[60]:

	Rating_category
503	0
768	0
2575	0
1215	0
1395	0
...	...
4200	0
1724	0
10423	0
7152	0
9571	0

5563 rows × 1 columns

In [61]:
```
y_test
```

Out[61]:

	Rating_category
9072	1
10635	0
7987	0
1597	0
1533	0
...	...
7842	0
5026	0
7819	1
9007	0
154	0

1855 rows × 1 columns

16. Standardize the data, so that the values are within a particular range.

In [62]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
```

In [63]:
```
df = ss.fit_transform(df)
```

In [64]:
```
df = pd.DataFrame(df)
df
```

Out[64]:

	0	1	2	3	4	5	6	7
0	-2.037910	-0.680066	-0.160461	-0.168829	-0.281595	-0.063065	-0.463448	-0.3535
1	-2.037910	-0.196758	-0.373793	-0.158250	-0.281595	-0.063065	-0.463448	-0.3535
2	-2.037910	1.009389	-0.599925	-0.061099	-0.281595	-0.063065	-0.463448	-0.3535
3	-2.037910	1.250836	0.095537	0.910421	-0.281595	-0.063065	2.516492	-0.3535
4	-2.037910	-0.196758	-0.851657	-0.166886	-0.281595	-0.063065	-0.463448	-0.3535
...	...	...	...	...	...	...	...	...
7413	-1.669278	-1.024006	-0.944713	-0.169024	-0.281595	-0.063065	-0.463448	-0.3535
7414	-0.686260	-1.516144	-0.860191	-0.169034	-0.281595	-0.063065	-0.463448	-0.3535
7415	-0.686260	-1.063254	1.290197	-0.168937	-0.281595	-0.063065	-0.463448	-0.3535
7416	-0.686260	-1.665963	-0.817524	-0.169043	-0.281595	-0.063065	-0.463448	-0.3535
7417	0.173880	1.415106	-0.160461	0.046848	-0.281595	-0.063065	-0.463448	-0.3535

7418 rows × 8 columns

In [ ]: