gs74zwjfd

May 18, 2024

# 1 Mid Term Project


- Syeda Aleeza Tahir

  FA22-Bai-038

- Muniba Manal

  FA22-Bai-032


# Dataset Google App store

# 2 1 DATASET SELECTION

We selected the Google App Store dataset for our project on applying classifiers to predict app ratings due to its relevance, accessibility, and diverse features. This dataset provides a real-world scenario with practical implications, offering information on app categories, sizes, installs, reviews, and more. By working with this dataset, we can gain insights into how machine learning models can be applied to improve app development, marketing strategies, and user experiences. Analyzing app ratings has significant implications for developers, marketers, and users, making the Google App Store dataset an ideal choice for exploring and predicting app ratings with classifiers.

# 3 2 DATA PROCESSING:

# 4 Importing necessary Libraries

[ ]:
```
# Import all the required libraries and modules
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import plotly
import plotly.graph_objects as go
```

Here we read a CSV file named "googleplaystore.csv" into a pandas DataFrame named df

[ ]:
```
import pandas as pd
```

```python
df = pd.read_csv('googleplaystore.csv')
```

[ ]:
```python
df
```

[ ]:
```
                                                    App             Category  \
0          Photo Editor & Candy Camera & Grid & ScrapBook        ART_AND_DESIGN
1                                   Coloring book moana        ART_AND_DESIGN
2      U Launcher Lite – FREE Live Cool Themes, Hide …       ART_AND_DESIGN
3                                 Sketch - Draw & Paint        ART_AND_DESIGN
4                  Pixel Draw - Number Art Coloring Book        ART_AND_DESIGN
…                                                   …                  …
10836                                  Sya9a Maroc - FR                FAMILY
10837                  Fr. Mike Schmitz Audio Teachings                FAMILY
10838                            Parkinson Exercices FR               MEDICAL
10839                     The SCP Foundation DB fr nn5n   BOOKS_AND_REFERENCE
10840       iHoroscope - 2018 Daily Horoscope & Astrology            LIFESTYLE

       Rating Reviews                Size     Installs  Type Price  \
0         4.1     159                 19M      10,000+  Free     0
1         3.9     967                 14M     500,000+  Free     0
2         4.7   87510                8.7M   5,000,000+  Free     0
3         4.5  215644                 25M  50,000,000+  Free     0
4         4.3     967                2.8M     100,000+  Free     0
…         …       …                  …          …     …     …
10836     4.5      38                 53M       5,000+  Free     0
10837     5.0       4                3.6M         100+  Free     0
10838     NaN       3                9.5M       1,000+  Free     0
10839     4.5     114  Varies with device       1,000+  Free     0
10840     4.5  398307                 19M  10,000,000+  Free     0

      Content Rating                     Genres      Last Updated  \
0           Everyone               Art & Design   January 7, 2018
1           Everyone  Art & Design;Pretend Play  January 15, 2018
2           Everyone               Art & Design    August 1, 2018
3               Teen               Art & Design      June 8, 2018
4           Everyone    Art & Design;Creativity     June 20, 2018
…                …                          …               …
10836       Everyone                  Education     July 25, 2017
10837       Everyone                  Education      July 6, 2018
10838       Everyone                    Medical  January 20, 2017
10839     Mature 17+          Books & Reference  January 19, 2015
10840       Everyone                  Lifestyle     July 25, 2018

              Current Ver         Android Ver
0                   1.0.0        4.0.3 and up
1                   2.0.0        4.0.3 and up
2                   1.2.4        4.0.3 and up
```

```
3      Varies with device          4.2 and up
4                     1.1          4.4 and up
…                      …                …
10836                1.48          4.1 and up
10837                 1.0          4.1 and up
10838                 1.0          2.2 and up
10839  Varies with device  Varies with device
10840  Varies with device  Varies with device

[10841 rows x 13 columns]
```

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

[ ]:
```
                                                 App        Category  Rating  \
0     Photo Editor & Candy Camera & Grid & ScrapBook  ART_AND_DESIGN     4.1
1                                Coloring book moana  ART_AND_DESIGN     3.9
2  U Launcher Lite – FREE Live Cool Themes, Hide …  ART_AND_DESIGN     4.7
3                              Sketch - Draw & Paint  ART_AND_DESIGN     4.5
4              Pixel Draw - Number Art Coloring Book  ART_AND_DESIGN     4.3

  Reviews  Size     Installs  Type Price Content Rating  \
0     159   19M      10,000+  Free     0       Everyone
1     967   14M     500,000+  Free     0       Everyone
2   87510  8.7M   5,000,000+  Free     0       Everyone
3  215644   25M  50,000,000+  Free     0           Teen
4     967  2.8M     100,000+  Free     0       Everyone

                      Genres      Last Updated         Current Ver  \
0               Art & Design   January 7, 2018               1.0.0
1  Art & Design;Pretend Play  January 15, 2018               2.0.0
2               Art & Design    August 1, 2018               1.2.4
3               Art & Design      June 8, 2018  Varies with device
4    Art & Design;Creativity     June 20, 2018                 1.1

   Android Ver
0  4.0.3 and up
1  4.0.3 and up
2  4.0.3 and up
3    4.2 and up
4    4.4 and up
```

[ ]:
```
df.tail()
```

[ ]:
```
                                                 App             Category  \
10836                               Sya9a Maroc - FR               FAMILY
10837                Fr. Mike Schmitz Audio Teachings               FAMILY
10838                          Parkinson Exercices FR              MEDICAL
```

```
10839                  The SCP Foundation DB fr nn5n  BOOKS_AND_REFERENCE
10840  iHoroscope - 2018 Daily Horoscope & Astrology            LIFESTYLE

       Rating Reviews                Size     Installs  Type Price  \
10836     4.5      38                 53M       5,000+  Free     0
10837     5.0       4                3.6M         100+  Free     0
10838     NaN       3                9.5M       1,000+  Free     0
10839     4.5     114  Varies with device       1,000+  Free     0
10840     4.5  398307                 19M  10,000,000+  Free     0

      Content Rating             Genres      Last Updated         Current Ver  \
10836       Everyone          Education     July 25, 2017                1.48
10837       Everyone          Education      July 6, 2018                 1.0
10838       Everyone            Medical  January 20, 2017                 1.0
10839     Mature 17+  Books & Reference  January 19, 2015  Varies with device
10840       Everyone          Lifestyle     July 25, 2018  Varies with device

              Android Ver
10836          4.1 and up
10837          4.1 and up
10838          2.2 and up
10839  Varies with device
10840  Varies with device
```

# 5 DATA CLEANING

#Checking the information in the data

[ ]:
```
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
```

#Dropping duplicated rows On play store, two apps may have same name, but all the size, installs, rating, reviews, price need not be same. so using these categories, we will drop the duplicates in the data

```
[ ]: for col in df.columns:
         print(f"Number of duplicates in {col} column are: {df[col].duplicated().
       ↪sum()}")
```

```
Number of duplicates in App column are: 1181
Number of duplicates in Category column are: 10807
Number of duplicates in Rating column are: 10800
Number of duplicates in Reviews column are: 4839
Number of duplicates in Size column are: 10379
Number of duplicates in Installs column are: 10819
Number of duplicates in Type column are: 10837
Number of duplicates in Price column are: 10748
Number of duplicates in Content Rating column are: 10834
Number of duplicates in Genres column are: 10721
Number of duplicates in Last Updated column are: 9463
Number of duplicates in Current Ver column are: 8008
Number of duplicates in Android Ver column are: 10807
```

```
[ ]: df.drop_duplicates(['App','Size','Installs','Reviews','Rating','Price','Android␣
      ↪Ver'], inplace=True)
     print(f'Number of rows after removing duplicate values: {df.shape[0]}')
```

```
Number of rows after removing duplicate values: 10350
```

#checking new data frame

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

```
<class 'pandas.core.frame.DataFrame'>
Index: 10350 entries, 0 to 10840
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   App             10350 non-null  object
 1   Category        10350 non-null  object
 2   Rating          8885 non-null   float64
 3   Reviews         10350 non-null  object
 4   Size            10350 non-null  object
 5   Installs        10350 non-null  object
 6   Type            10349 non-null  object
 7   Price           10350 non-null  object
```

```
 8   Content Rating  10349 non-null  object
 9   Genres          10350 non-null  object
 10  Last Updated    10350 non-null  object
 11  Current Ver     10342 non-null  object
 12  Android Ver     10347 non-null  object
dtypes: float64(1), object(12)
memory usage: 1.1+ MB
```

we can see that number of rows before deleting duplicates was 10841, now its 10350

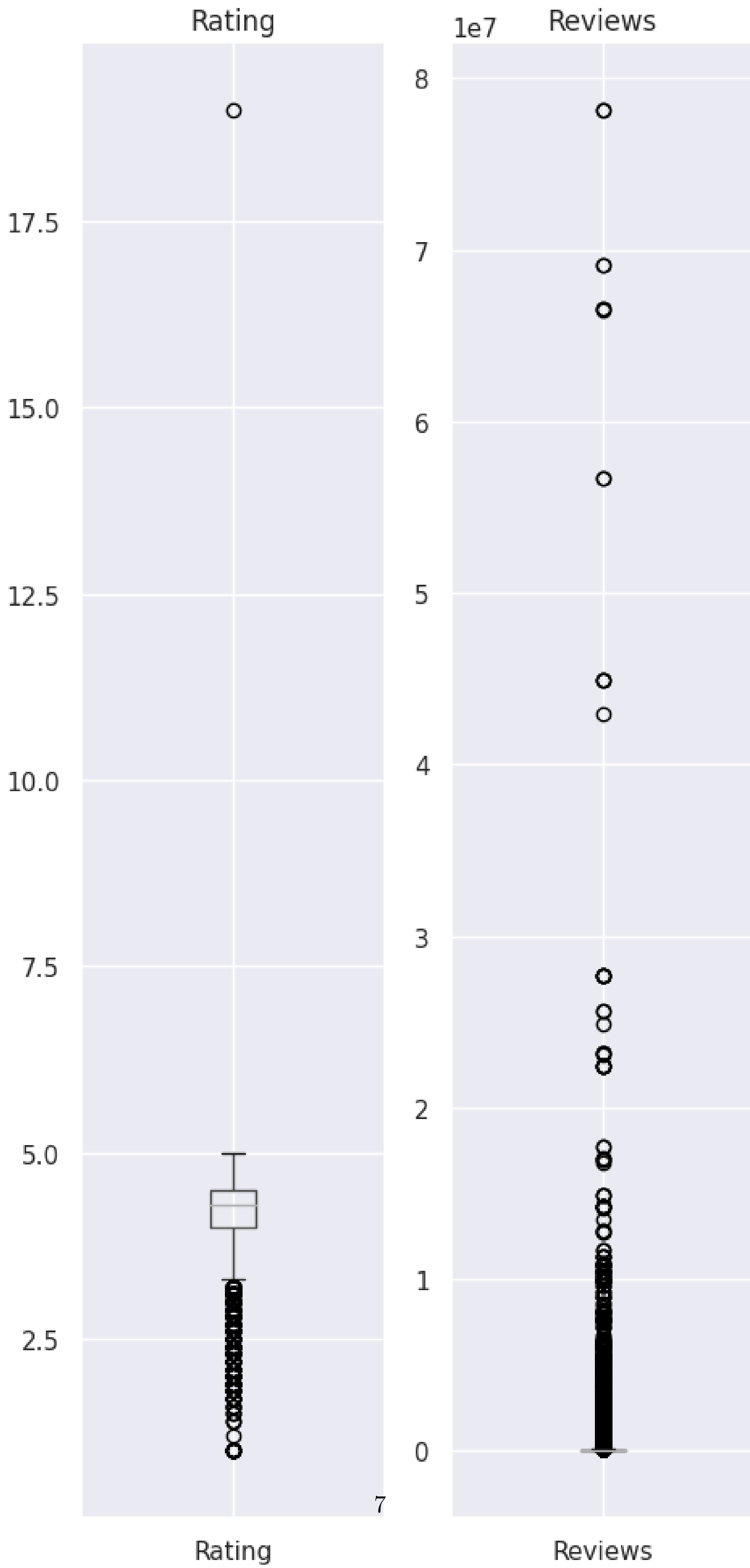
Here we are calculating the total number of columns in the DataFrame df that have missing values by first counting the number of missing values in each column (df.isnull().sum()) and then calculating the length of that result (len()).

[ ]:
```
len(df.isnull().sum())
```

[ ]:
```
13
```

## 6 Visualizing Outliers

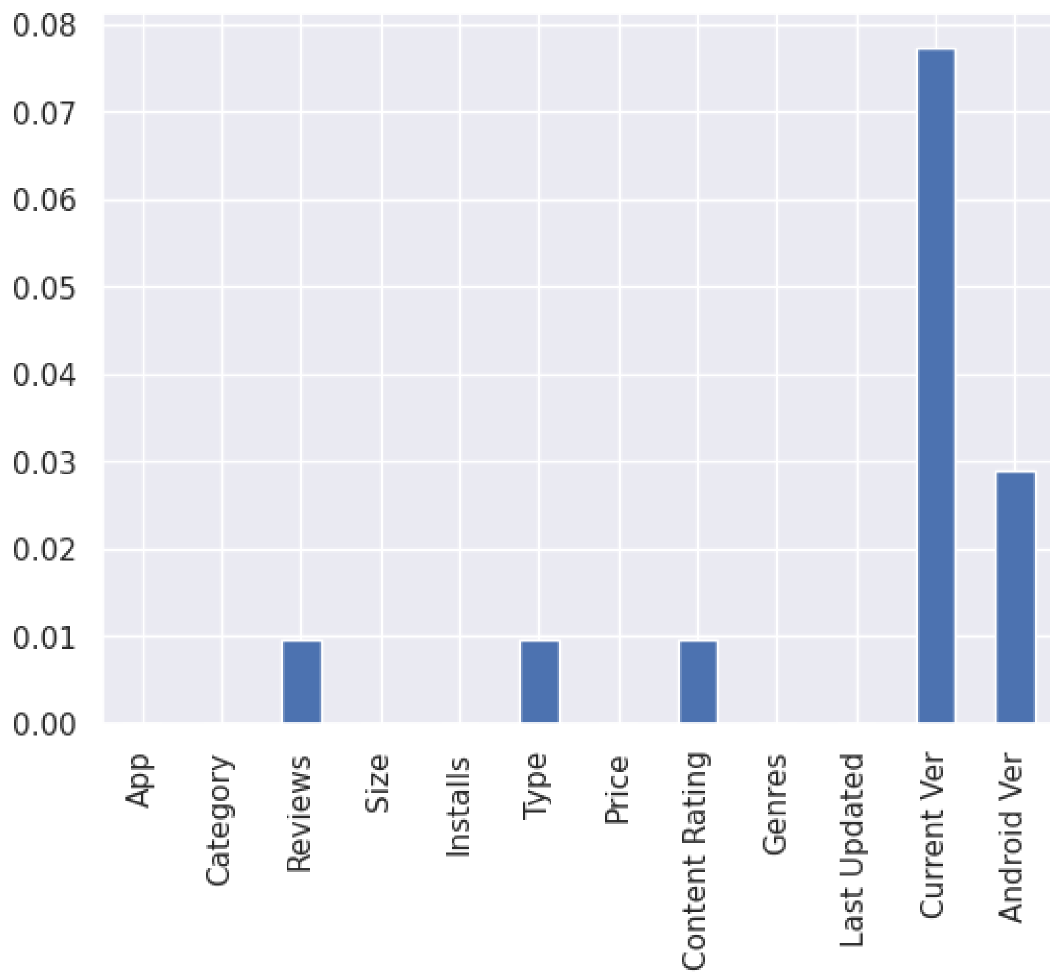
[ ]:
```
# Plotting
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(5, 10))
df['Reviews'] = pd.to_numeric(df['Reviews'], errors='coerce')
# Plot box plot for 'Rating' data
df.boxplot(column=['Rating'], ax=axes[0])
axes[0].set_title('Rating')

# Plot box plot for 'Reviews' data
df.boxplot(column=['Reviews'], ax=axes[1])
axes[1].set_title('Reviews')

plt.tight_layout()
plt.show()
```


Rating
17.5
15.0
12.5
10.0
7.5
5.0
2.5
Rating
Reviews
1e7
8
7
6
5
4
3
2
1
0
Reviews

### 6.1 Percentage of missing values

[ ]:
```
missing_percentage = df.isnull().sum()/len(df)*100
missing_percentage[missing_percentage<1].plot(kind = 'bar')
```

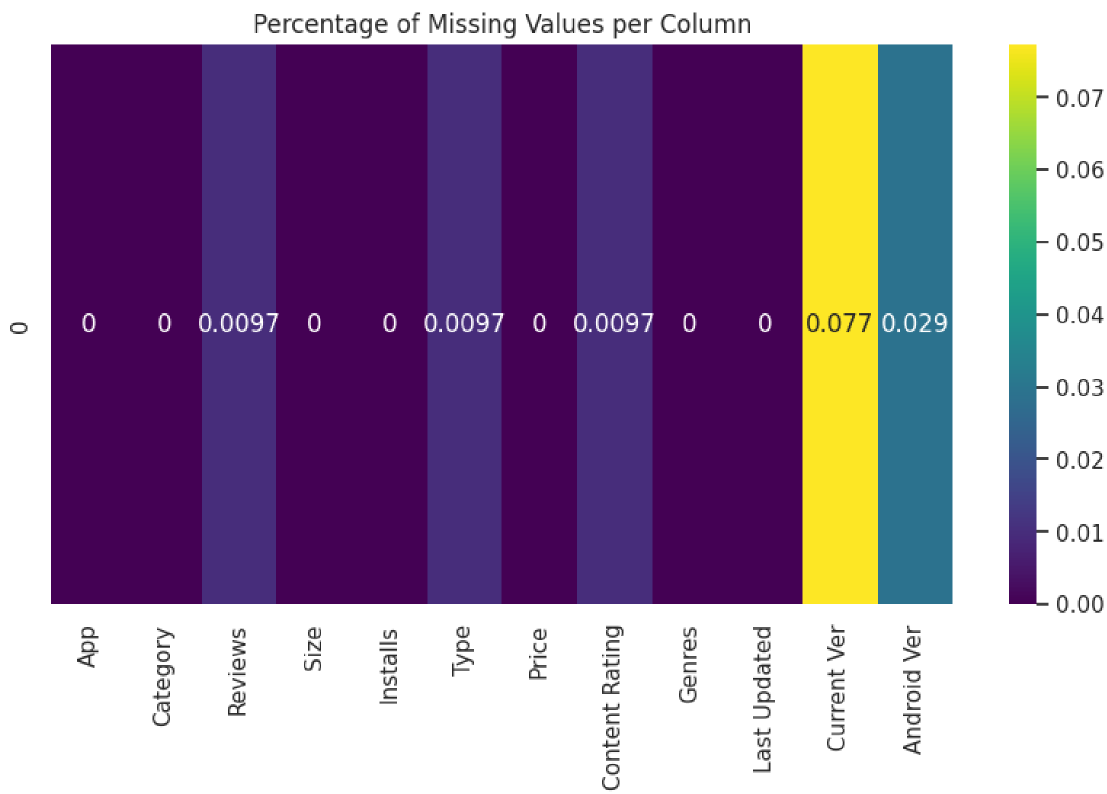
[ ]: <Axes: >




### 6.2 Percentage of missing values by heatmap

[ ]:
```
import pandas as pd
import seaborn as sns

# Calculate missing percentages
```

```
missing_percentage = df.isnull().sum()/len(df)*100

# Select columns with missing percentages less than 1%
missing_percentage = missing_percentage[missing_percentage<1]

# Create heatmap
plt.figure(figsize=(10, 5))
sns.heatmap(missing_percentage.to_frame().T, cmap='viridis', annot=True)
plt.title('Percentage of Missing Values per Column')
plt.show()
```




# Dropping the entries where there are missing values and checking each column in the data for null values

[ ]:
```
df=df.dropna()
df.isnull().sum()
```

[ ]:
```
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
```

[ ]:
```
df=df.dropna()
df.Reviews= df.Reviews.astype(float)
```

#converting into numeric values

[ ]:
```
df.Installs= df["Installs"].str.replace(",", "")
df.Installs= df["Installs"].str.replace("+", "")
df["Installs"] = pd.to_numeric(df["Installs"])
```

[ ]:
```
df["Price"]= df["Price"].str.replace("$", "")
```

[ ]:
```
# Convert the data in "Price" to float
df["Price"]= df.Price.astype(float)
```

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

[ ]:

| | Rating | Reviews | Installs | Price |
|---|---|---|---|---|
| count | 8878.000000 | 8.878000e+03 | 8.878000e+03 | 8878.000000 |
| mean | 4.187745 | 4.729619e+05 | 1.649903e+07 | 0.963719 |
| std | 0.522572 | 2.906987e+06 | 8.643798e+07 | 16.201978 |
| min | 1.000000 | 1.000000e+00 | 1.000000e+00 | 0.000000 |
| 25% | 4.000000 | 1.640000e+02 | 1.000000e+04 | 0.000000 |
| 50% | 4.300000 | 4.708000e+03 | 5.000000e+05 | 0.000000 |
| 75% | 4.500000 | 7.119725e+04 | 5.000000e+06 | 0.000000 |
| max | 5.000000 | 7.815831e+07 | 1.000000e+09 | 400.000000 |

[ ]:
```
#Ensuring there are no longer missing values
df.isnull().any()

# False for every category means that there are no longer missing values
```

[ ]:
```
App               False
Category          False
Rating            False
Reviews           False
Size              False
Installs          False
```

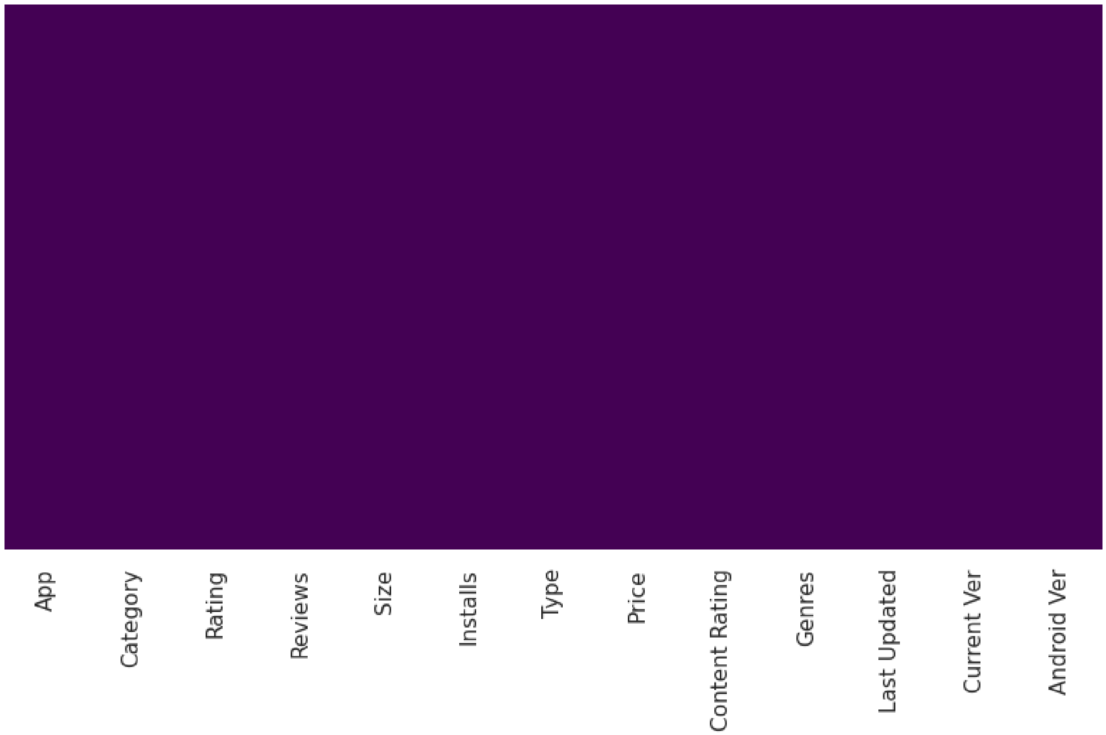
```
Type                False
Price               False
Content Rating      False
Genres              False
Last Updated        False
Current Ver         False
Android Ver         False
dtype: bool
```

```
[ ]: # procedure for converting the column "Size" to float
     # there are sizes counted in mb, kb, in numbers without measurement unit and␣
      ↪with "varies with device"
     df.Size.unique()
     # removing the "m" which is the mb for the size

     df.Size= df["Size"].str.replace("M", "")
     df.Size= df['Size'].str.replace("Varies with device","-1")
     #Here we replace k and change the unit to Mb
     df['Size']=df['Size'].apply(lambda x: str(round((float(x.rstrip('k'))/1024),1)␣
      ↪)if x[-1]=='k' else x)
```

# 7 Displaying missing values if-any left

```
[ ]: plt.figure(figsize=(10,5))
     sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='viridis')
```

```
[ ]: <Axes: >
```




**Hence, the graph showed no missing values remained in the dataset. which shows that the data is cleaned properly and missing values are handled properly. So lets further prooced to Exploratory data analysis.**

[ ]:
```
df.Size.unique()
```

[ ]:
```
array(['19', '14', '8.7', '25', '2.8', '5.6', '29', '33', '3.1', '28',
       '12', '20', '21', '37', '5.5', '17', '39', '31', '4.2', '23',
       '6.0', '6.1', '4.6', '9.2', '5.2', '11', '24', '-1', '9.4', '15',
       '10', '1.2', '26', '8.0', '7.9', '56', '57', '35', '54', '0.2',
       '3.6', '5.7', '8.6', '2.4', '27', '2.7', '2.5', '7.0', '16', '3.4',
       '8.9', '3.9', '2.9', '38', '32', '5.4', '18', '1.1', '2.2', '4.5',
       '9.8', '52', '9.0', '6.7', '30', '2.6', '7.1', '22', '6.4', '3.2',
       '8.2', '4.9', '9.5', '5.0', '5.9', '13', '73', '6.8', '3.5', '4.0',
       '2.3', '2.1', '42', '9.1', '55', '0.0', '7.3', '6.5', '1.5', '7.5',
       '51', '41', '48', '8.5', '46', '8.3', '4.3', '4.7', '3.3', '40',
       '7.8', '8.8', '6.6', '5.1', '61', '66', '0.1', '8.4', '3.7', '44',
       '0.7', '1.6', '6.2', '53', '1.4', '3.0', '7.2', '5.8', '3.8',
       '9.6', '45', '63', '49', '77', '4.4', '70', '9.3', '8.1', '36',
       '6.9', '7.4', '84', '97', '2.0', '1.9', '1.8', '5.3', '47', '0.5',
       '76', '7.6', '59', '9.7', '78', '72', '43', '7.7', '6.3', '0.3',
       '93', '65', '79', '100', '58', '50', '68', '64', '34', '67', '60',
       '94', '9.9', '99', '0.6', '95', '80', '1.7', '10.0', '74', '62',
```

```
'69', '75', '98', '85', '82', '96', '87', '71', '86', '91', '81',
'92', '83', '88', '0.8', '0.9', '0.4', '4.8', '1.3', '1.0', '4.1',
'89', '90'], dtype=object)
```

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

```
<class 'pandas.core.frame.DataFrame'>
Index: 8878 entries, 0 to 10840
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   App             8878 non-null   object
 1   Category        8878 non-null   object
 2   Rating          8878 non-null   float64
 3   Reviews         8878 non-null   float64
 4   Size            8878 non-null   object
 5   Installs        8878 non-null   int64
 6   Type            8878 non-null   object
 7   Price           8878 non-null   float64
 8   Content Rating  8878 non-null   object
 9   Genres          8878 non-null   object
 10  Last Updated    8878 non-null   object
 11  Current Ver     8878 non-null   object
 12  Android Ver     8878 non-null   object
dtypes: float64(3), int64(1), object(9)
memory usage: 971.0+ KB
```

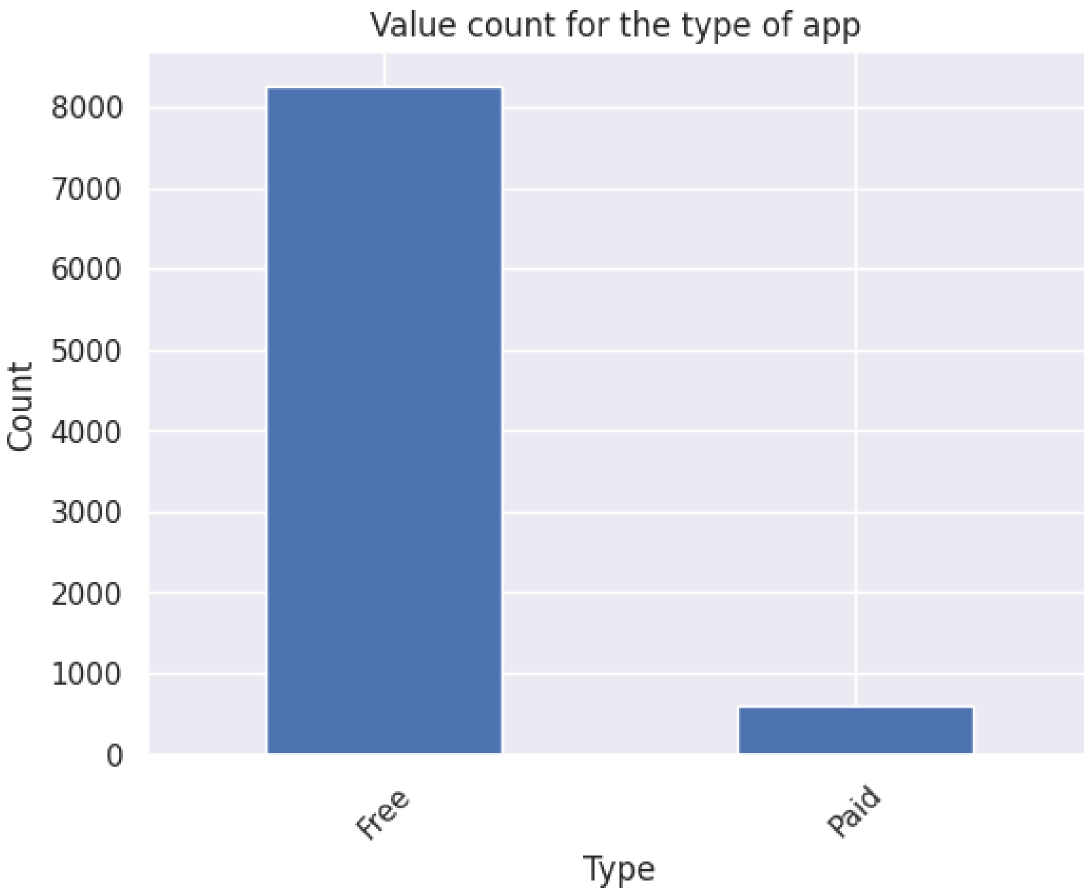
Rows reduced to 8878 after cleaning process

# 8 3 Exploitory Data Analysis (EDA):

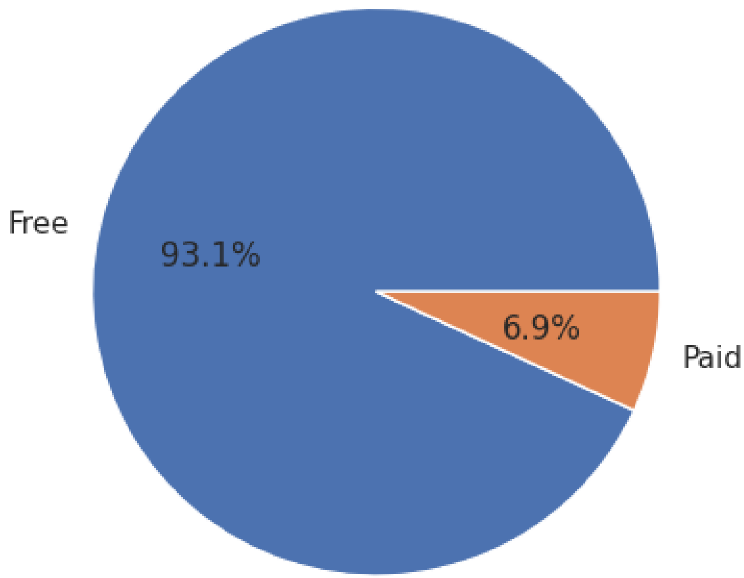
[ ]:
```
#Checking unique categories in the Type of the Apps
df["Type"].value_counts()
```

[ ]:
```
Type
Free    8268
Paid     610
Name: count, dtype: int64
```

[ ]:
```
df["Type"].value_counts().plot.bar()
plt.ylabel("Count")
plt.xlabel("Type")
plt.title("Value count for the type of app ")
plt.xticks(rotation=45)

plt.show()
```




[ ]:
```
total = df['Type'].value_counts()
plt.pie(total,labels = ['Free','Paid'], autopct= "%1.1f%%")
plt.show()
```




[ ]:
```
# Converting the data in the column "Reviews" to float to that we can apply␣
↪statistics

df.Reviews= df.Reviews.astype(float)
```

As the columns of Genres is same as the column for category, we drop the column for Genres.Also, the column of android version, current version, Last updated are not of our use, so we drop these columns.

[ ]:
```
df.drop(['Genres','Last Updated','Current Ver',"Android␣
↪Ver"],axis=1,inplace=True)
```

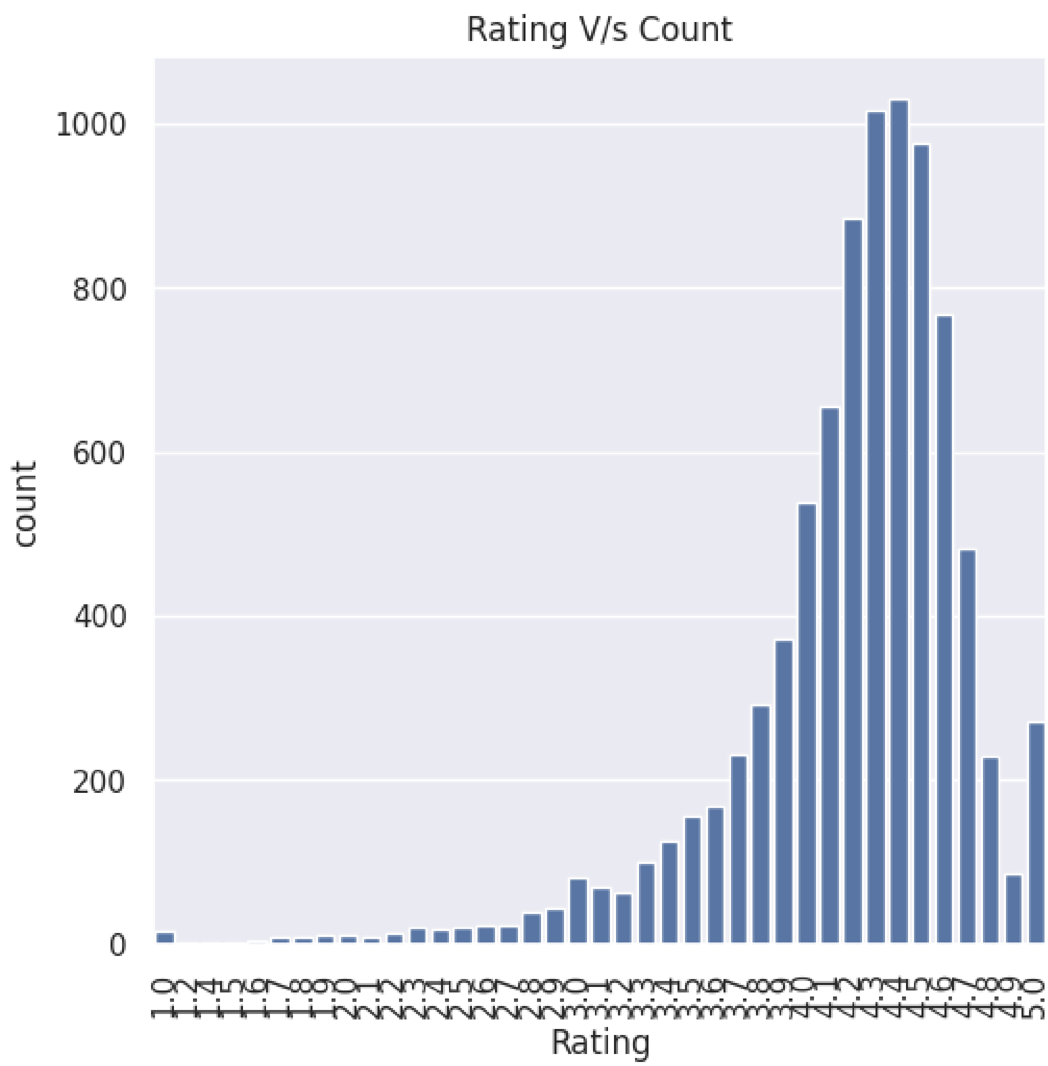
[ ]:
```
df.Rating.value_counts()
```

[ ]:
```
Rating
4.4    1030
4.3    1016
4.5     975
4.2     885
4.6     767
4.1     655
4.0     538
4.7     482
3.9     372
```

```
3.8    293
5.0    271
3.7    231
4.8    228
3.6    169
3.5    157
3.4    127
3.3    101
4.9     87
3.0     82
3.1     69
3.2     63
2.9     45
2.8     40
2.6     24
2.7     23
2.5     20
2.3     20
2.4     19
1.0     16
2.2     14
1.9     12
2.0     12
1.7      8
1.8      8
2.1      8
1.6      4
1.4      3
1.5      3
1.2      1
Name: count, dtype: int64
```

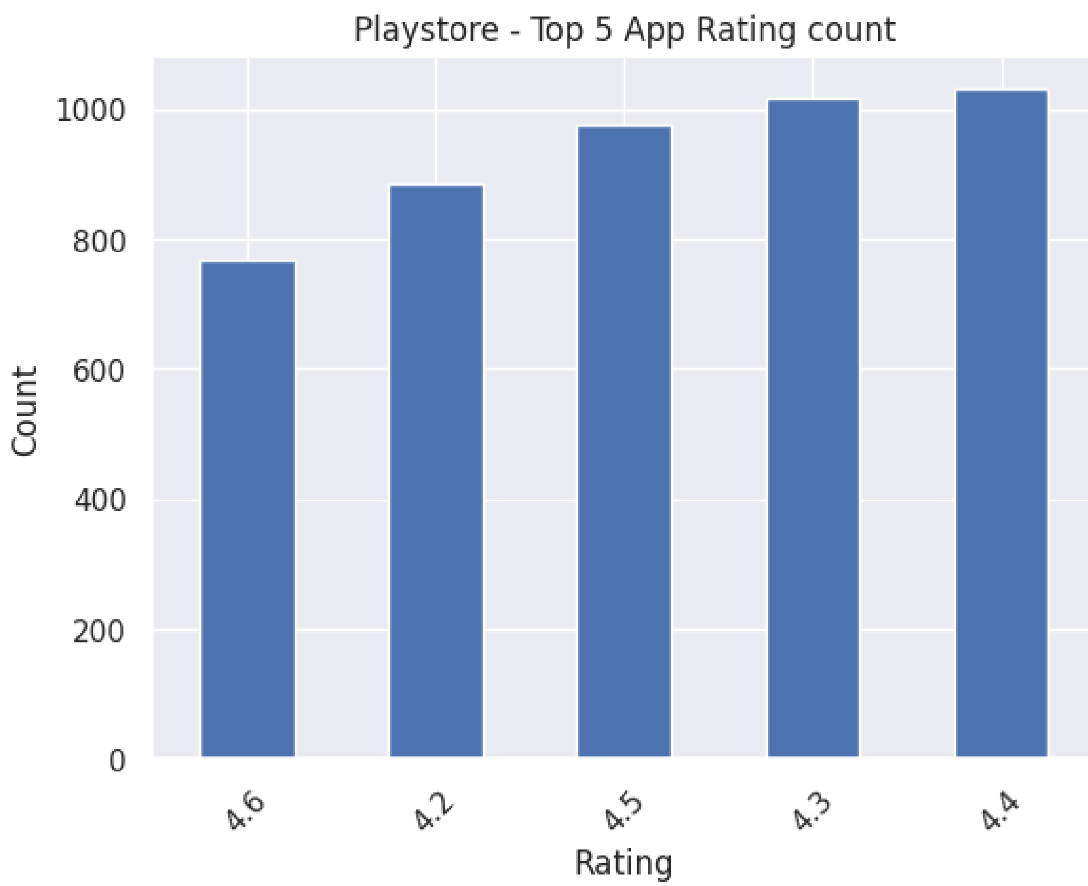
# 9 Count plot for rating

[ ]:
```
plt.figure(figsize=(6,6))
#sns.set_theme(style="darkgrid")
plt.xticks(rotation=90)
plt.title("Rating V/s Count")
ax = sns.countplot(x="Rating", data=df)
```



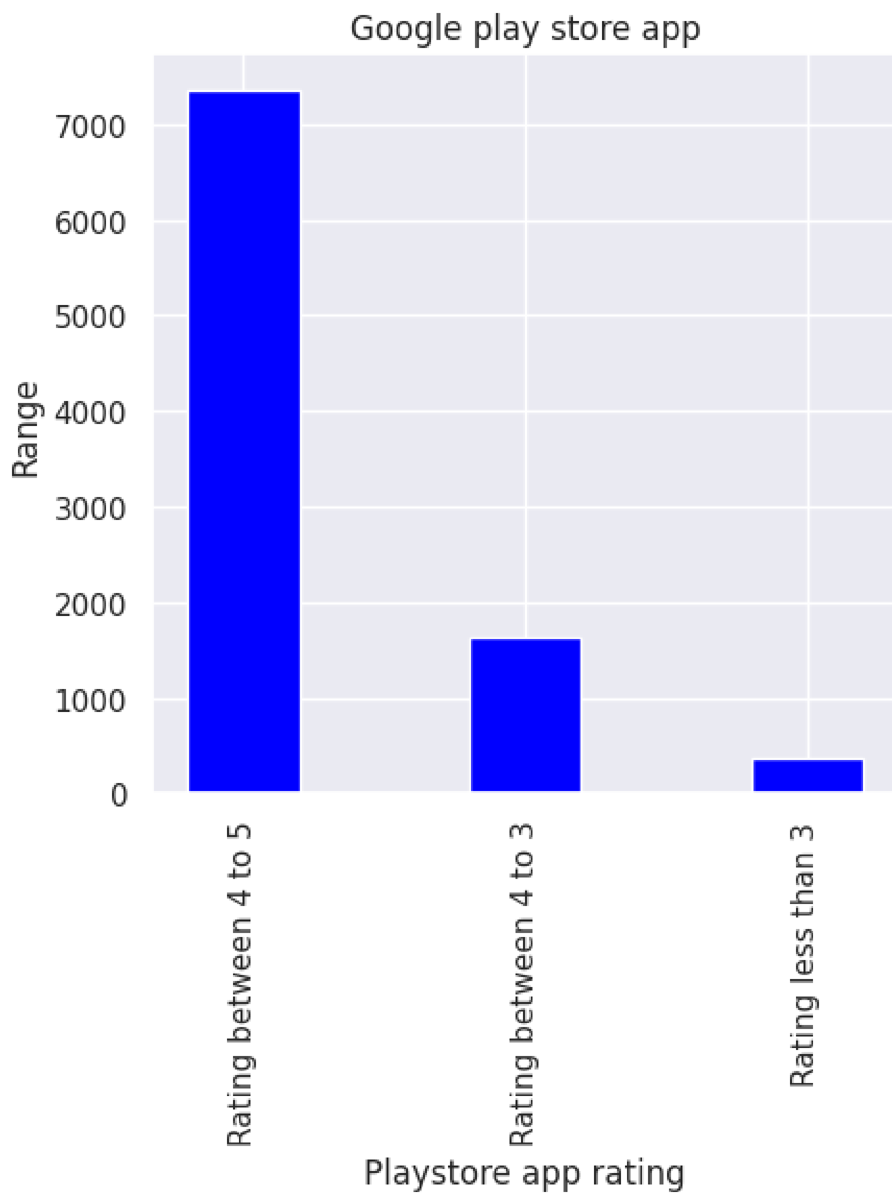

#Looking at the top five value counts for rating

```
[ ]: df["Rating"].value_counts().nlargest(5).sort_values(ascending=True).plot.bar()
plt.ylabel("Count")
plt.xlabel("Rating")
plt.title("Playstore - Top 5 App Rating count")
plt.xticks(rotation=45)

plt.show()
```




```
[ ]: # creating the dataset
     # Plot show there are more than 7000 apps having rating more than 4

     data = {'Rating between 4 to 5':7363, 'Rating between 4 to 3':1627, 'Rating␣
      ↪less than 3':370,
                     }
     courses = list(data.keys())
     values = list(data.values())

     fig = plt.figure(figsize = (5, 5))

     # creating the bar plot
     plt.bar(courses, values, color ='blue',
                     width =0.4)

     plt.xlabel("Playstore app rating")
     plt.ylabel("Range")
     plt.title("Google play store app")
     plt.xticks(rotation=90)
```

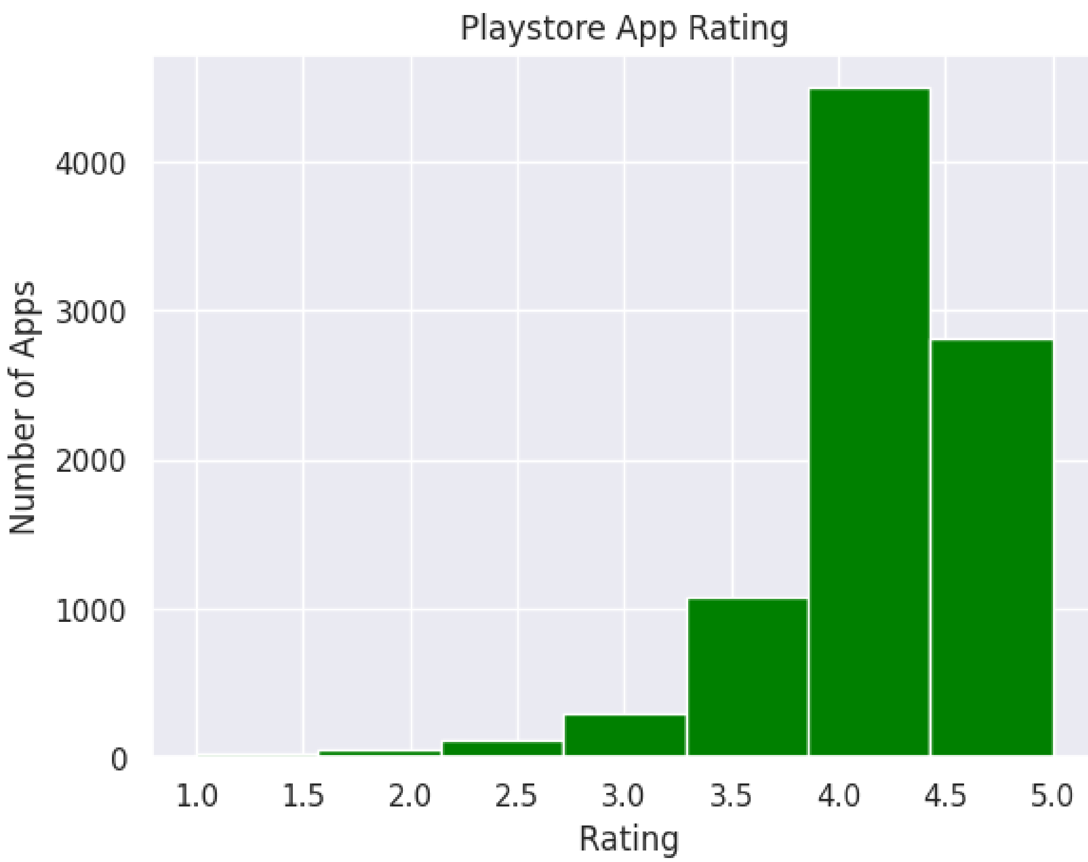
```
plt.show()
```




#Hist plot for the app rating

[ ]:
```
app_rating= df["Rating"]
num_bins=7
plt.hist(app_rating, num_bins, facecolor="green", alpha = 1)
plt.title('Playstore App Rating')
plt.xlabel(" Rating")
plt.ylabel("Number of Apps")
plt.show
```

[ ]: <function matplotlib.pyplot.show(close=None, block=None)>




#sorting app with rating 5 corresponding to installs and reviews

[ ]:
```
df1=df[df['Rating']==5]
df1 =df1.sort_values(by=['Installs','Reviews'], ascending=False)
df1
```

[ ]:
```
                                                  App   Category  Rating  \
9511                             Ek Bander Ne Kholi Dukan     FAMILY     5.0
8058                                          Oración CX  LIFESTYLE     5.0
8260        Superheroes, Marvel, DC, Comics, TV, Movies News     COMICS     5.0
7514                  CL Keyboard - Myanmar Keyboard (No Ads)      TOOLS     5.0
10357                                         Ríos de Fe  LIFESTYLE     5.0
…                                                        …        …      …
2459     Anatomy & Physiology Vocabulary Exam Review App    MEDICAL     5.0
9218                                 EB Cash Collections   BUSINESS     5.0
2454                                 KBA-EZ Health Guide    MEDICAL     5.0
5917                                           Ra Ga Ba       GAME     5.0
10697                                           Mu.F.O.       GAME     5.0
```

```
       Reviews Size  Installs  Type  Price Content Rating
9511      10.0  3.0     10000  Free   0.00       Everyone
8058     103.0  3.8      5000  Free   0.00       Everyone
8260      34.0   12      5000  Free   0.00       Everyone
7514      24.0  3.2      5000  Free   0.00       Everyone
10357    141.0   15      1000  Free   0.00       Everyone
…          …   …        …    …     …            …
2459       1.0  4.6         5  Free   0.00       Everyone
9218       1.0  4.3         5  Free   0.00       Everyone
2454       4.0   25         1  Free   0.00       Everyone
5917       2.0   20         1  Paid   1.49       Everyone
10697      2.0   16         1  Paid   0.99       Everyone

[271 rows x 9 columns]
```

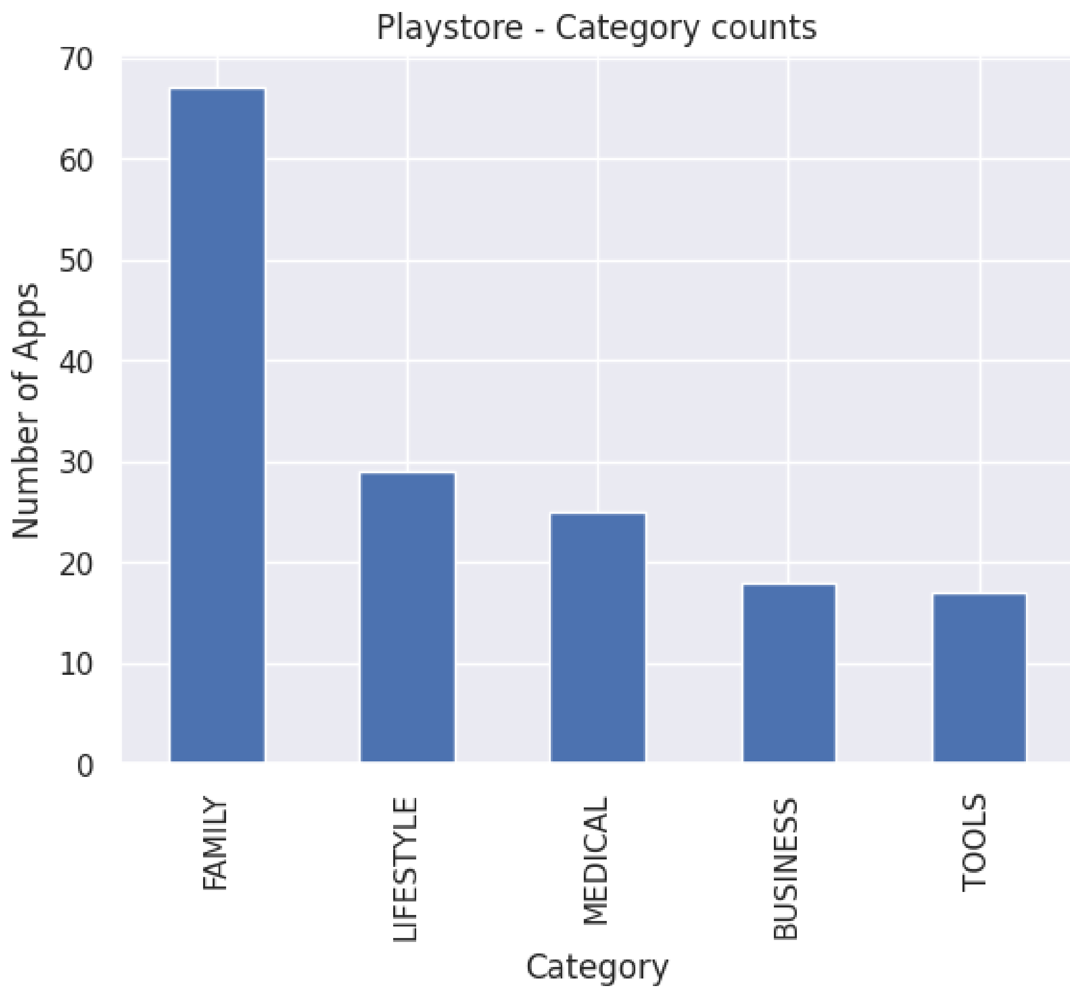
#Value count of each category for the apps with rating 5

[ ]:
```
df1.Category.value_counts(ascending=False)
```

[ ]:
```
Category
FAMILY                 67
LIFESTYLE              29
MEDICAL                25
BUSINESS               18
TOOLS                  17
GAME                   12
HEALTH_AND_FITNESS     12
PERSONALIZATION        10
SOCIAL                  8
PRODUCTIVITY            8
FINANCE                 8
NEWS_AND_MAGAZINES      7
BOOKS_AND_REFERENCE     6
SHOPPING                6
DATING                  6
EVENTS                  6
PHOTOGRAPHY             6
COMMUNICATION           5
SPORTS                  4
TRAVEL_AND_LOCAL        3
LIBRARIES_AND_DEMO      2
COMICS                  2
FOOD_AND_DRINK          2
ART_AND_DESIGN          1
PARENTING               1
Name: count, dtype: int64
```

#Top five categories with respect to value count of app (rating=5)

```
[ ]: df1.Category.value_counts(ascending=False).nlargest(5).
      ↪sort_values(ascending=False).plot.bar()
     plt.ylabel("Number of Apps")
     plt.xlabel("Category")
     plt.title("Playstore - Category counts")
     plt.show()
```




#Reading the top five categories in the apps

```
[ ]: df["Category"].value_counts().nlargest(5).sort_values(ascending=False)
```

```
[ ]: Category
     FAMILY          1711
     GAME            1074
     TOOLS            732
```

```
PRODUCTIVITY      334
FINANCE           317
Name: count, dtype: int64
```

#Category vs Count

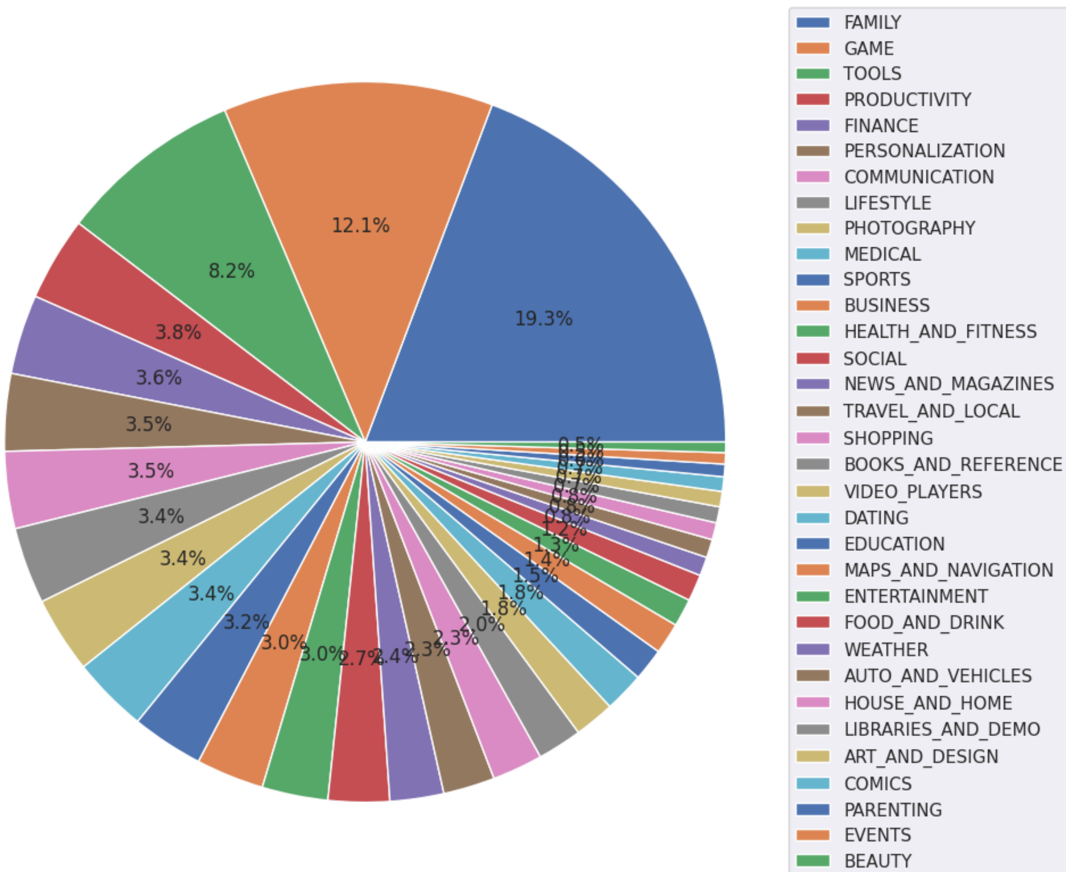
[ ]:
```
#Which category has highest no. of installs
top_category_by_installs  = df.groupby('Category').agg({'Installs' : 'sum'})
top_category_by_installs.sort_values(by='Installs', ascending=False)
```

[ ]:

| Category | Installs |
|---|---|
| GAME | 31543862717 |
| COMMUNICATION | 24152241530 |
| SOCIAL | 12513841475 |
| PRODUCTIVITY | 12463070180 |
| TOOLS | 11440224500 |
| FAMILY | 9915079590 |
| PHOTOGRAPHY | 9721243130 |
| TRAVEL_AND_LOCAL | 6361859300 |
| VIDEO_PLAYERS | 6221897200 |
| NEWS_AND_MAGAZINES | 5393110650 |
| SHOPPING | 2563331540 |
| ENTERTAINMENT | 2455660000 |
| PERSONALIZATION | 2074341930 |
| BOOKS_AND_REFERENCE | 1916291655 |
| SPORTS | 1528531465 |
| HEALTH_AND_FITNESS | 1361006220 |
| BUSINESS | 863518120 |
| FINANCE | 770312400 |
| MAPS_AND_NAVIGATION | 724267560 |
| LIFESTYLE | 534741120 |
| EDUCATION | 533852000 |
| WEATHER | 426096500 |
| FOOD_AND_DRINK | 257777750 |
| DATING | 206522410 |
| HOUSE_AND_HOME | 125082000 |
| ART_AND_DESIGN | 124228100 |
| LIBRARIES_AND_DEMO | 61083000 |
| COMICS | 56036100 |
| AUTO_AND_VEHICLES | 53129800 |
| MEDICAL | 42162676 |
| PARENTING | 31116110 |
| BEAUTY | 26916200 |
| EVENTS | 15949410 |

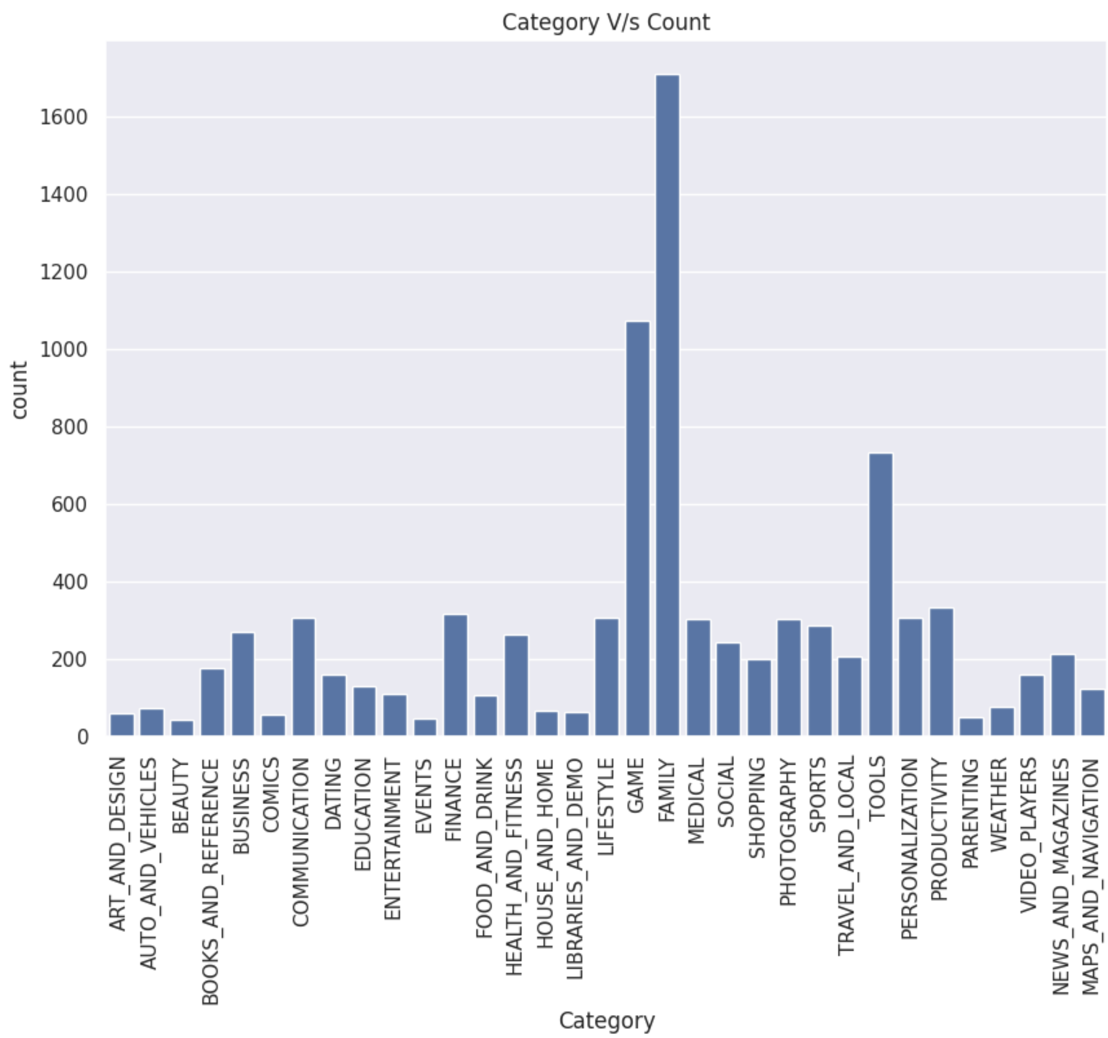
[ ]:
```python
#Basic pie chart to view distribution of apps across various categories
fig, ax = plt.subplots(figsize=(9, 9), subplot_kw=dict(aspect="equal"))
number_of_apps = df["Category"].value_counts()
labels = number_of_apps.index
sizes = number_of_apps.values
ax.pie(sizes,labeldistance=2,autopct='%1.1f%%')
ax.legend(labels=labels,loc="right",bbox_to_anchor=(0.9, 0, 0.5, 1))
ax.axis("equal")
```

[ ]:
```
(-1.0999999999163619,
 1.0999999999960173,
 -1.0999999998369518,
 1.09999999687039)
```




[ ]:
```python
plt.figure(figsize=(10,7))
sns.set_theme(style="darkgrid")
#dat = sns.load_dataset("df")
plt.xticks(rotation=90)
plt.title("Category V/s Count")
```

```
ax = sns.countplot(x="Category", data=df)
```



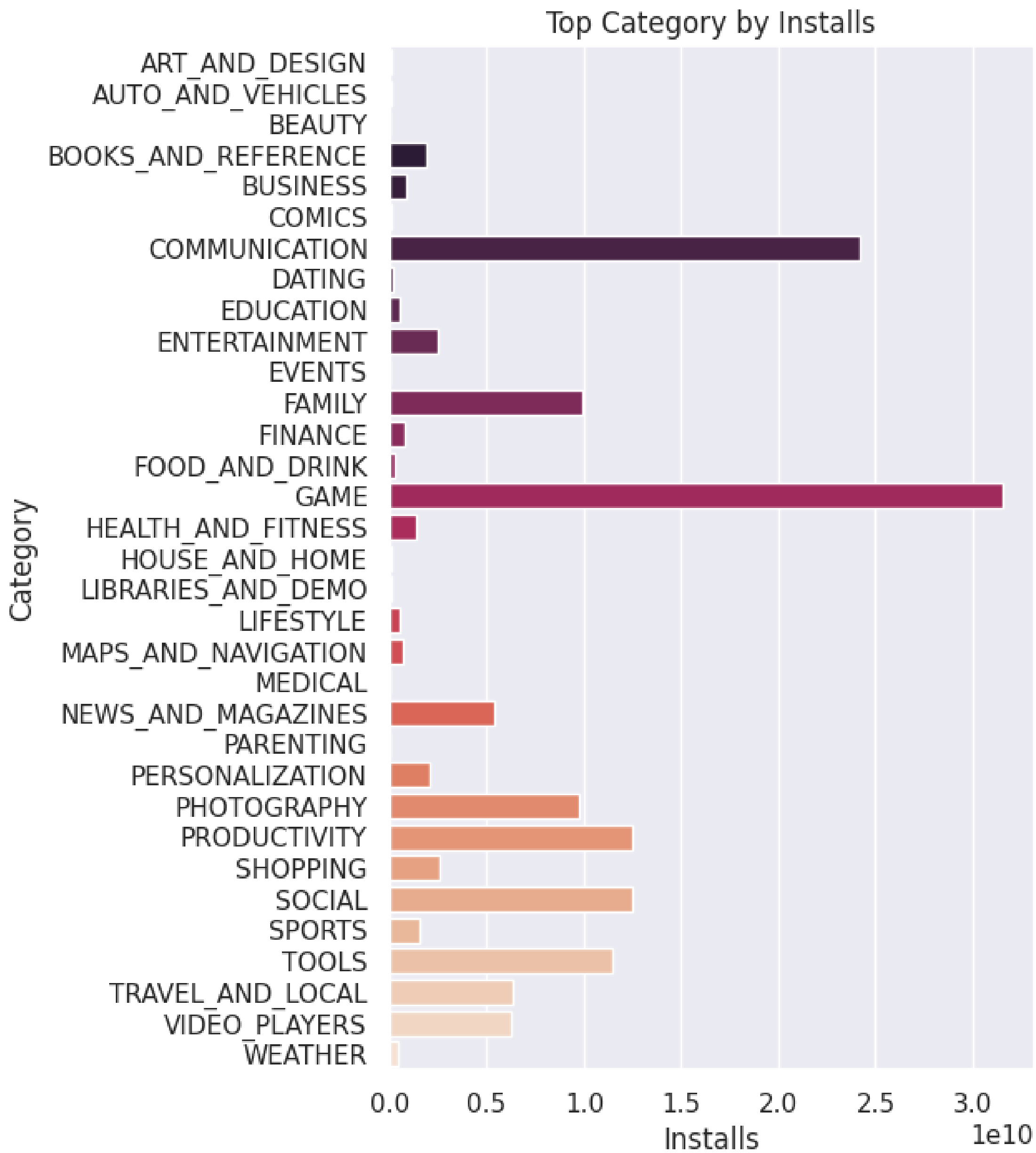

#Top category by installs

[ ]:
```
plt.figure(figsize=(5,8))
plt.title("Top Category by Installs")
palette = sns.color_palette("rocket", len(top_category_by_installs))
sns.barplot(y = top_category_by_installs.index, x =␣
↪top_category_by_installs['Installs'], palette=palette)
```

```
<ipython-input-147-42561806eebb>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.
```

```
sns.barplot(y = top_category_by_installs.index, x =
top_category_by_installs['Installs'], palette=palette)
```

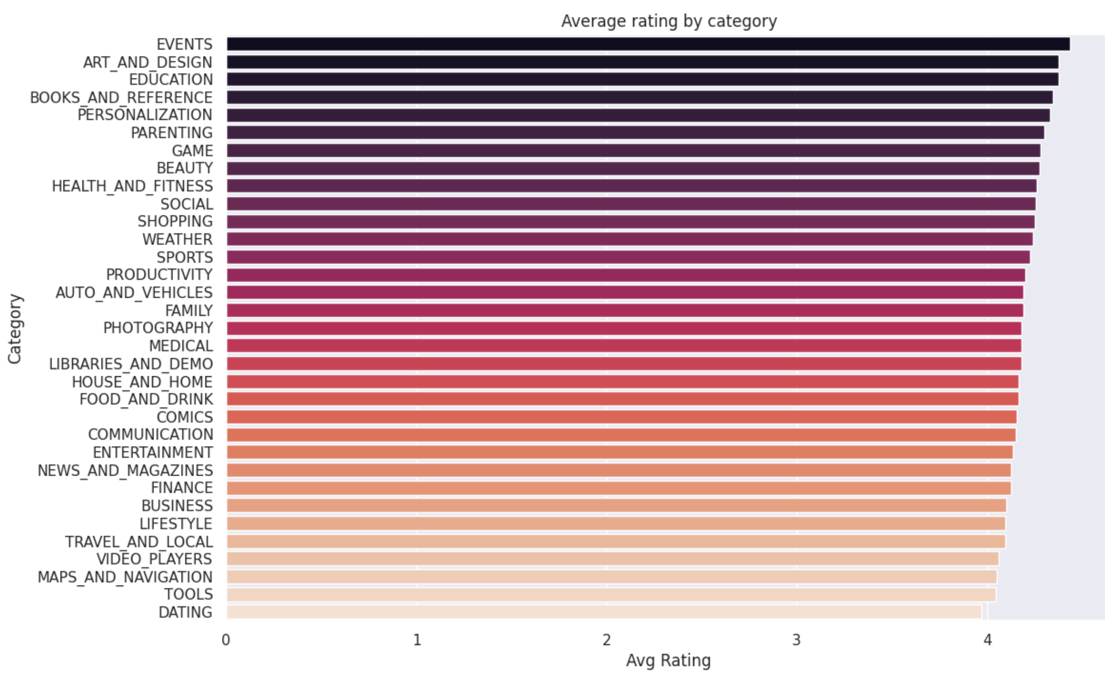
[ ]: <Axes: title={'center': 'Top Category by Installs'}, xlabel='Installs', ylabel='Category'>



# 10 Average Rating by Category

```
[ ]: avg_cat_mean = df.groupby('Category')['Rating'].mean().sort_values(ascending =␣
     ↪False)
     #avg_cat_mean
     plt.figure(figsize=(12,8))
     sns.barplot(x = avg_cat_mean.values, y = avg_cat_mean.index, palette="rocket")
     plt.xlabel('Avg Rating')
     plt.ylabel('Category')
     plt.title('Average rating by category')
     plt.show()
```

```
<ipython-input-148-5a979f60cf87>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x = avg_cat_mean.values, y = avg_cat_mean.index, palette="rocket")
```



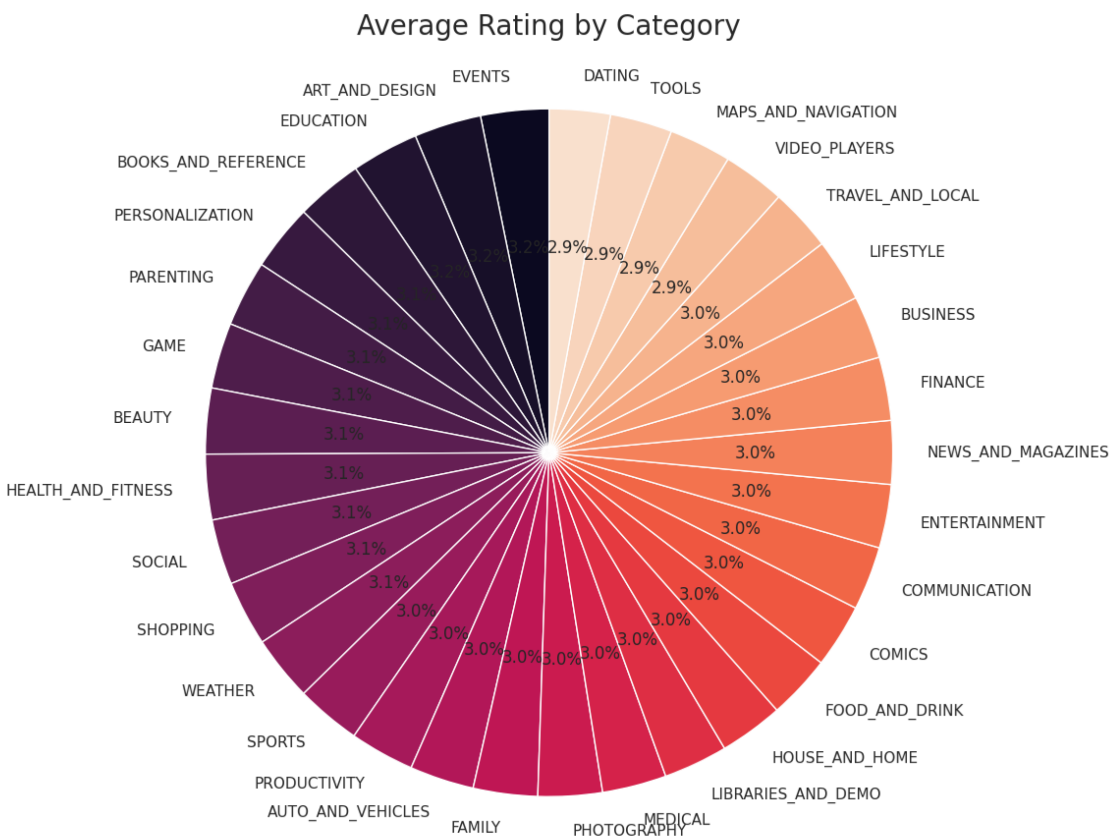

```
[ ]: #Which category has the highest average rating?
     top_rated_category = df.groupby('Category').agg({'Rating' : 'mean'})
     top_rated_category.sort_values(by='Rating', ascending=False)
```

[ ]:

```
                        Rating
Category
EVENTS                4.435556
ART_AND_DESIGN        4.377049
EDUCATION             4.375969
BOOKS_AND_REFERENCE   4.347458
PERSONALIZATION       4.333117
PARENTING             4.300000
GAME                  4.281285
BEAUTY                4.278571
HEALTH_AND_FITNESS    4.261450
SOCIAL                4.254918
SHOPPING              4.252239
WEATHER               4.244000
SPORTS                4.225175
PRODUCTIVITY          4.201796
AUTO_AND_VEHICLES     4.190411
FAMILY                4.190123
PHOTOGRAPHY           4.182895
MEDICAL               4.182450
LIBRARIES_AND_DEMO    4.179688
HOUSE_AND_HOME        4.164706
FOOD_AND_DRINK        4.164151
COMICS                4.155172
COMMUNICATION         4.151466
ENTERTAINMENT         4.136036
NEWS_AND_MAGAZINES    4.128505
FINANCE               4.127445
BUSINESS              4.102593
LIFESTYLE             4.096066
TRAVEL_AND_LOCAL      4.094146
VIDEO_PLAYERS         4.063750
MAPS_AND_NAVIGATION   4.051613
TOOLS                 4.046995
DATING                3.971698
```

[ ]:

```
avg_cat_mean = df.groupby('Category')['Rating'].mean().sort_values(ascending =
    False)
colors = sns.color_palette("rocket", len(avg_cat_mean))
plt.figure(figsize=(14, 10))
plt.pie(avg_cat_mean.values, labels=avg_cat_mean.index, autopct="%1.1f%%",
    colors=colors, startangle=90, wedgeprops={'edgecolor': 'white', 'linewidth':
    1})
plt.axis('equal')  # Equal aspect ratio ensures a circular pie chart
plt.title('\nAverage Rating by Category\n', fontsize=20)
plt.show()
```




# 11 Sorting (descending sorting) the dataframe by number of installs

[ ]:
```python
df.sort_values(by="Installs", ascending= False).head()
```

[ ]:
```
                          App           Category  Rating     Reviews Size  \
386                  Hangouts      COMMUNICATION     4.0   3419433.0   -1
3736              Google News  NEWS_AND_MAGAZINES     3.9    877635.0   13
4098  Maps - Navigate & Explore   TRAVEL_AND_LOCAL     4.3   9231613.0   -1
3909                Instagram             SOCIAL     4.5  66509917.0   -1
2604                Instagram             SOCIAL     4.5  66577446.0   -1

        Installs  Type  Price Content Rating
386   1000000000  Free    0.0       Everyone
3736  1000000000  Free    0.0           Teen
4098  1000000000  Free    0.0       Everyone
3909  1000000000  Free    0.0           Teen
2604  1000000000  Free    0.0           Teen
```

#Making another dataframe that gives most installed and most rated apps

[ ]:
```
top_installed_and_rated_apps = df.sort_values(by=["Installs", "Rating"],␣
 ↪ascending=True)
top_installed_and_rated_apps.head() # main top apps
```

[ ]:
```
                         App Category  Rating  Reviews Size  Installs  Type  \
2454   KBA-EZ Health Guide  MEDICAL     5.0      4.0   25         1  Free
5917             Ra Ga Ba      GAME     5.0      2.0   20         1  Paid
10697             Mu.F.O.      GAME     5.0      2.0   16         1  Paid
10562         FK Atlantas    SPORTS     1.5      2.0   26         5  Free
2450      Tablet Reminder   MEDICAL     5.0      4.0  2.5         5  Free

       Price Content Rating
2454    0.00       Everyone
5917    1.49       Everyone
10697   0.99       Everyone
10562   0.00       Everyone
2450    0.00       Everyone
```

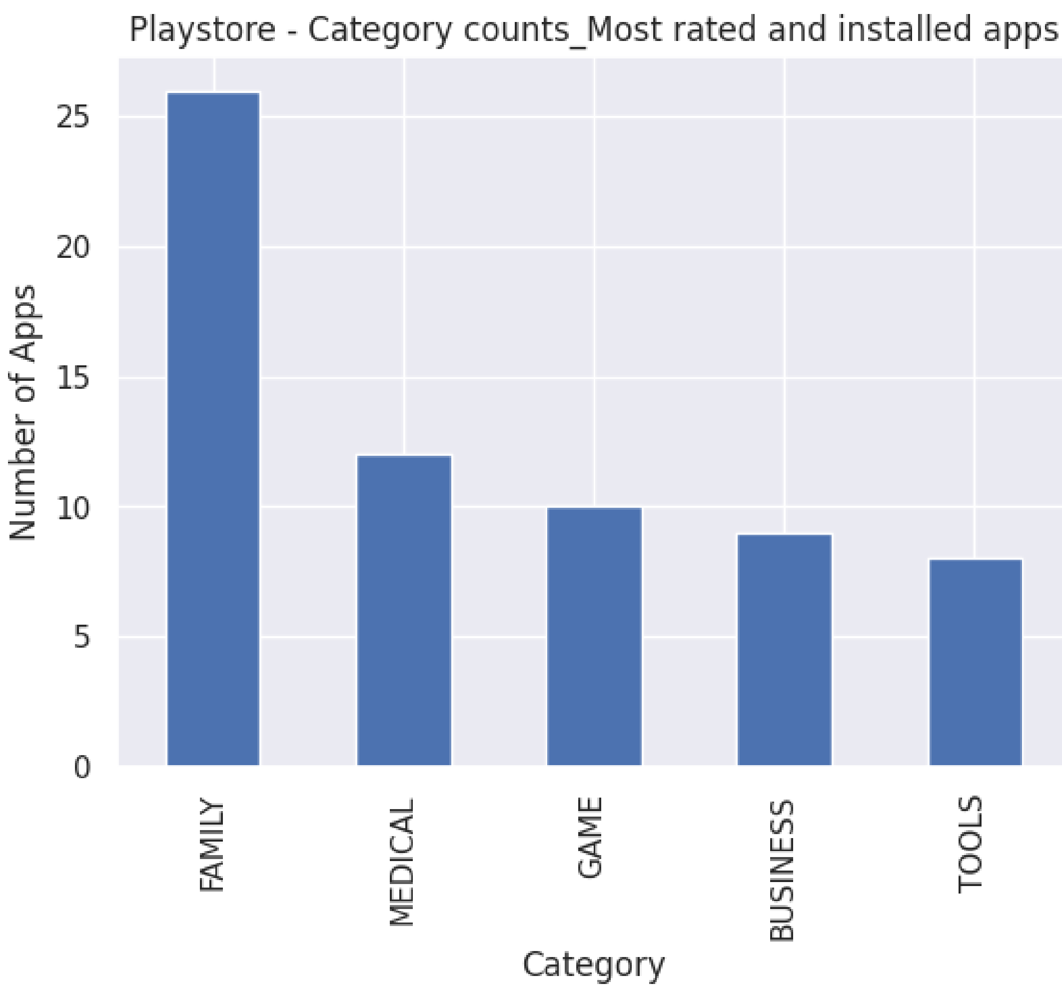
**-1 indicate a missing or unknown value in the dataset, and it's a common practice to use specific placeholders like -1 or NaN (Not a Number) to represent missing data.**

[ ]:
```
#Now, we make dataframe for most installed and reviewed apps
top_installed_and_reviewed_apps = df.sort_values(by=["Installs", "Reviews"],␣
 ↪ascending=False)
top_installed_and_reviewed_apps.head()
```

[ ]:
```
                     App       Category  Rating     Reviews Size    Installs  \
2544            Facebook         SOCIAL     4.1  78158306.0   -1  1000000000
3943            Facebook         SOCIAL     4.1  78128208.0   -1  1000000000
336   WhatsApp Messenger  COMMUNICATION     4.4  69119316.0   -1  1000000000
3904  WhatsApp Messenger  COMMUNICATION     4.4  69109672.0   -1  1000000000
2604           Instagram         SOCIAL     4.5  66577446.0   -1  1000000000

      Type  Price Content Rating
2544  Free    0.0           Teen
3943  Free    0.0           Teen
336   Free    0.0       Everyone
3904  Free    0.0       Everyone
2604  Free    0.0           Teen
```

## 12 Top installed and rated apps

[ ]:
```
#Here We use the barplot to see the value counts of category in the top␣
↪installed and rated apps
top_installed_and_rated_apps["Category"].head(100).value_counts().nlargest(5).
↪sort_values(ascending=False).plot.bar()
plt.ylabel("Number of Apps")
plt.xlabel("Category")
plt.title("Playstore - Category counts_Most rated and installed apps")
plt.show()
```



## 12.1 Top 10 reviewed apps

[ ]:
```
# top 10 reviewed apps
df.nlargest(10, 'Reviews')[['App', 'Reviews']]
```

[ ]:

| | App | Reviews |
|---|---|---|
| 2544 | Facebook | 78158306.0 |
| 3943 | Facebook | 78128208.0 |
| 336 | WhatsApp Messenger | 69119316.0 |
| 3904 | WhatsApp Messenger | 69109672.0 |
| 2604 | Instagram | 66577446.0 |
| 2545 | Instagram | 66577313.0 |
| 3909 | Instagram | 66509917.0 |
| 382 | Messenger – Text and Video Chat for Free | 56646578.0 |
| 335 | Messenger – Text and Video Chat for Free | 56642847.0 |
| 1879 | Clash of Clans | 44893888.0 |

#Count for content rating

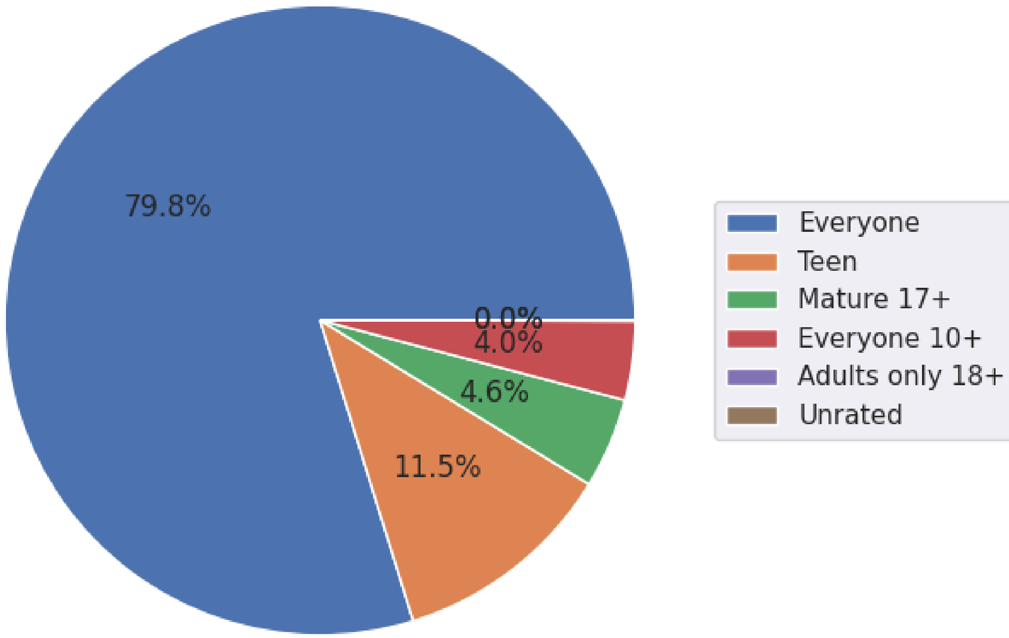
[ ]:
```
df["Content Rating"].value_counts(ascending=False)
```

[ ]:
```
Content Rating
Everyone           7083
Teen               1021
Mature 17+          411
Everyone 10+        359
Adults only 18+       3
Unrated               1
Name: count, dtype: int64
```

#Basic pie chart to view distribution of apps across various content rating

[ ]:
```
fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(aspect="equal"))
number_of_apps = df["Content Rating"].value_counts()
labels = number_of_apps.index
sizes = number_of_apps.values
ax.pie(sizes,labeldistance=2,autopct='%1.1f%%')
ax.legend(labels=labels,loc="right",bbox_to_anchor=(0.9, 0, 0.5, 1))
```

[ ]: <matplotlib.legend.Legend at 0x7bcdb234c460>



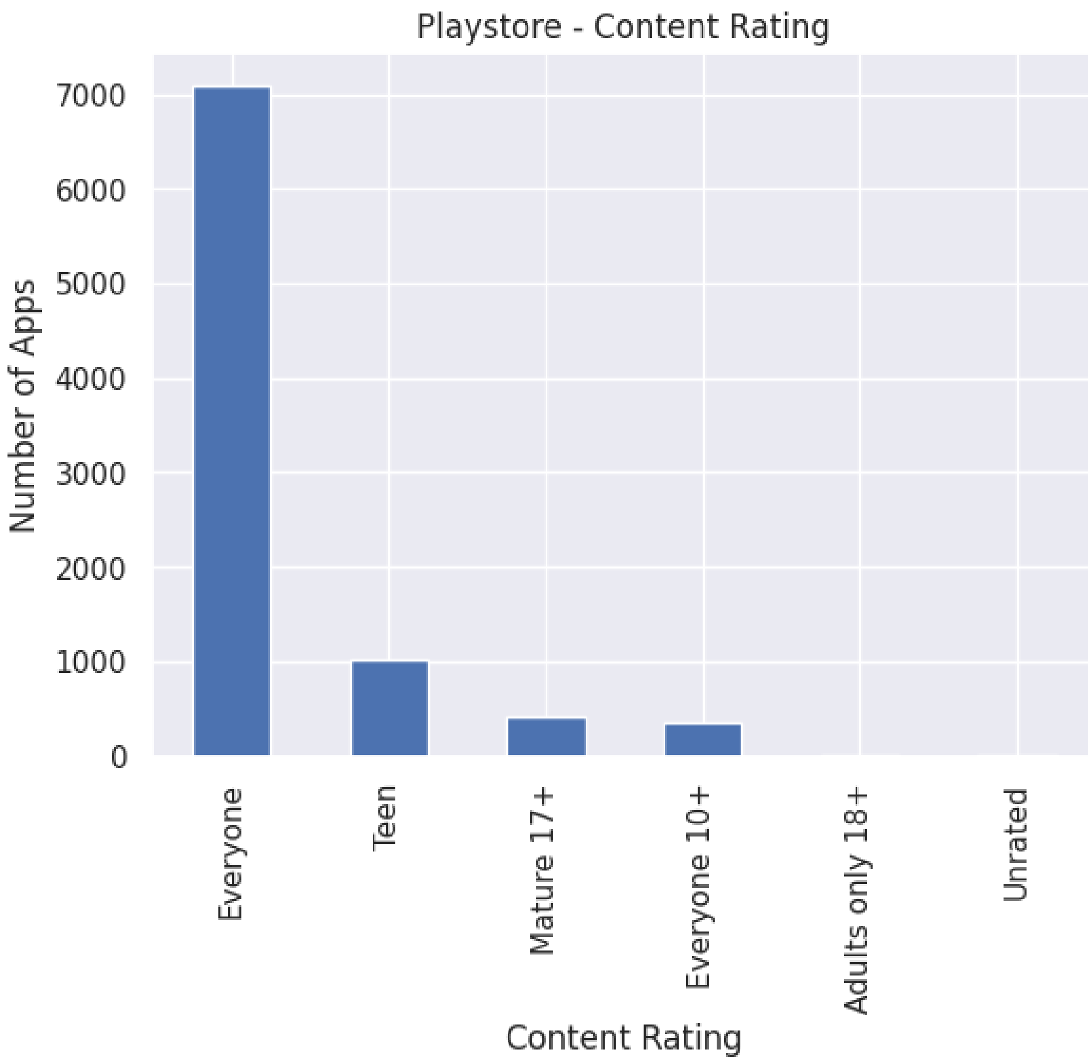

#Reading top apps sorted by reviews

[ ]:
```
df.sort_values(by="Reviews", ascending= False).head()
```

[ ]:
```
                   App       Category  Rating     Reviews Size    Installs  \
2544              Facebook         SOCIAL     4.1  78158306.0   -1  1000000000
3943              Facebook         SOCIAL     4.1  78128208.0   -1  1000000000
336   WhatsApp Messenger  COMMUNICATION     4.4  69119316.0   -1  1000000000
3904  WhatsApp Messenger  COMMUNICATION     4.4  69109672.0   -1  1000000000
2604             Instagram         SOCIAL     4.5  66577446.0   -1  1000000000

      Type  Price Content Rating
2544  Free    0.0           Teen
3943  Free    0.0           Teen
336   Free    0.0       Everyone
3904  Free    0.0       Everyone
2604  Free    0.0           Teen
```

#Plot for content rating This plot shows that more than 7000 apps allows the content reviews to everyone

```
[ ]: df["Content Rating"].value_counts().sort_values(ascending=False).plot.bar()
     plt.ylabel("Number of Apps")
     plt.xlabel("Content Rating")
     plt.title("Playstore - Content Rating")
     plt.show()
```




### 12.2 Summary

**Reviews column has higher correlation with Installs column. Number of Reviews increases with the number of Installs, which is obvious.**

**The majority of apps are distributed around a rating of 4.The 'Family' category has the most apps in this data. 'Game' category is next with the second highest number.**

**In this data, Games have the most installations. Communication and social apps come next in terms of how many people have them.Majority of apps are free (about 92.63%) as compared to paid apps.**

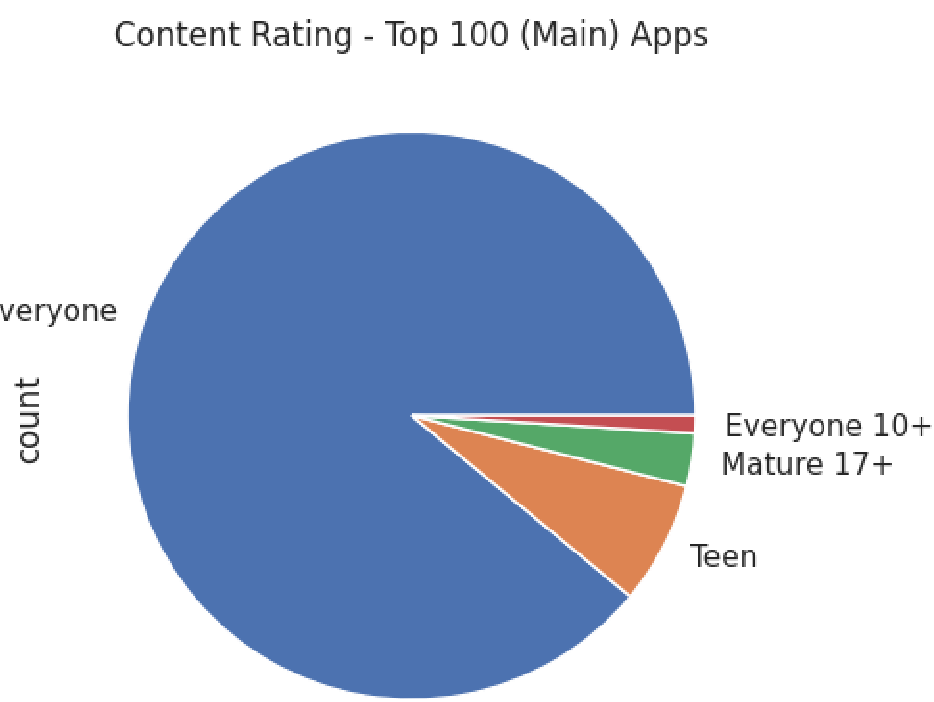
**The 'Events' category has the hisghest average rating while the 'Dating' category receives lowest average rating.**

**In this dataset, Many apps worked with Android version 4.1 and newer.The paid apps have the highest average rating as compared to free apps.**

**Most apps have a content rating of 'Everyone', but there are only two apps without any rating**

### 12.3 Content Rating of top 100 Main apps

[ ]:
```
#Here we look for the content rating distribution using pie chart
app1=top_installed_and_rated_apps.head(100)
app1["Content Rating"].value_counts().plot.pie()
plt.title("Content Rating - Top 100 (Main) Apps")
plt.show()
```




#sorting apps according to price

[ ]:
```
# Prices of the apps
df["Price"].value_counts().sort_values(ascending=False).head(10)
```

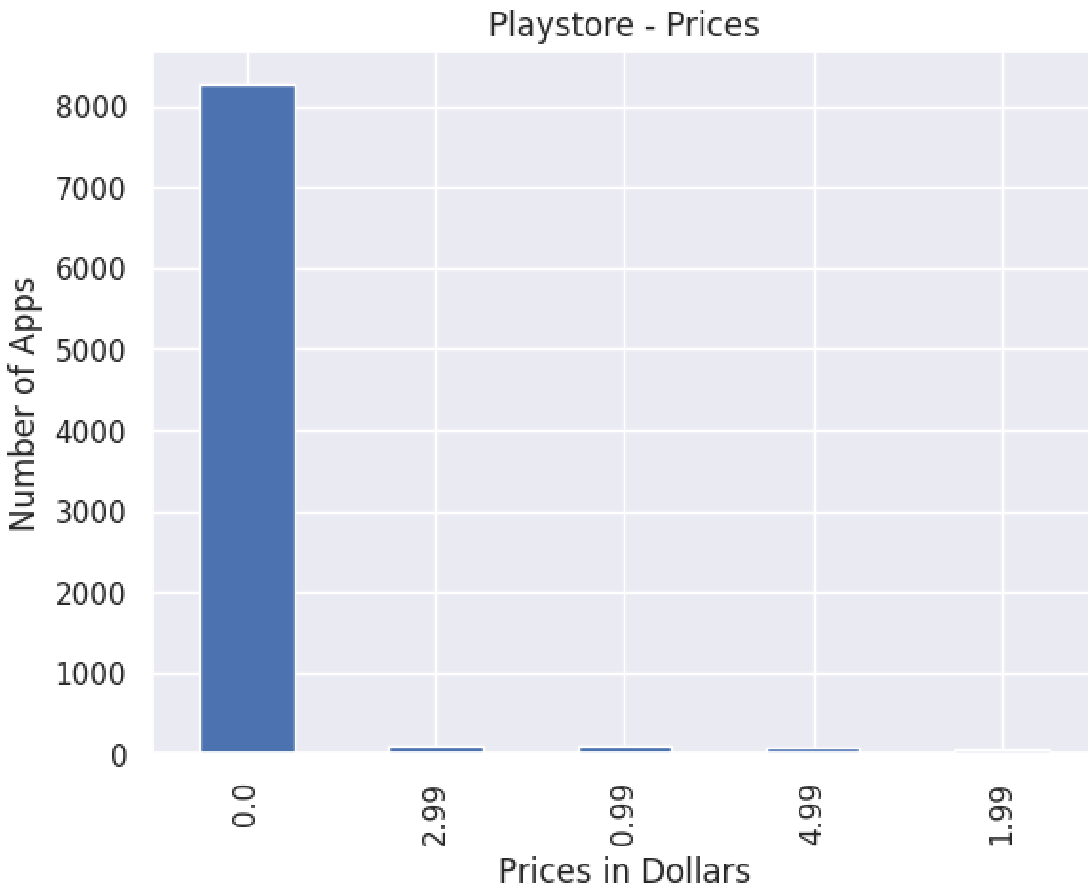
[ ]:
```
Price
0.00    8268
2.99     110
0.99     104
4.99      68
1.99      59
3.99      55
1.49      30
2.49      20
5.99      14
9.99      14
Name: count, dtype: int64
```

[ ]:
```
df.Type.value_counts()
```

[ ]:
```
Type
Free    8268
Paid     610
Name: count, dtype: int64
```

#Price vs app

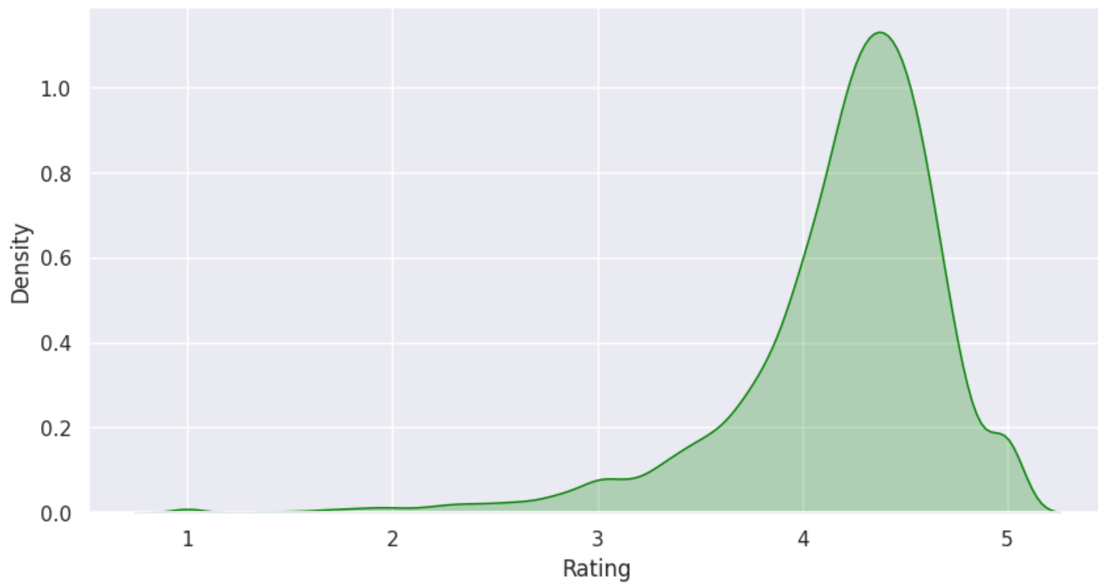
[ ]:
```
#Now we visualize it by using bar plot
#This plot shows that data has more than 90% apps that are free
df["Price"].value_counts().nlargest(5).sort_values(ascending=False).plot.bar()
plt.ylabel("Number of Apps")
plt.xlabel("Prices in Dollars")
plt.title("Playstore - Prices")
plt.show()
```




```
[ ]: # How are the ratings distributed across different apps?
     plt.figure(figsize=(10, 5))
     sns.kdeplot(df['Rating'], color="green", shade=True)
```

```
<ipython-input-164-3b88650a8b3a>:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df['Rating'], color="green", shade=True)
```

```
[ ]: <Axes: xlabel='Rating', ylabel='Density'>
```




[ ]:
```
# Which category has the highest no. of apps?
top_category = df['Category'].value_counts().head(10)
top_category
```

[ ]:
```
Category
FAMILY             1711
GAME               1074
TOOLS               732
PRODUCTIVITY        334
FINANCE             317
PERSONALIZATION     308
COMMUNICATION       307
LIFESTYLE           305
PHOTOGRAPHY         304
MEDICAL             302
Name: count, dtype: int64
```

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

[ ]:

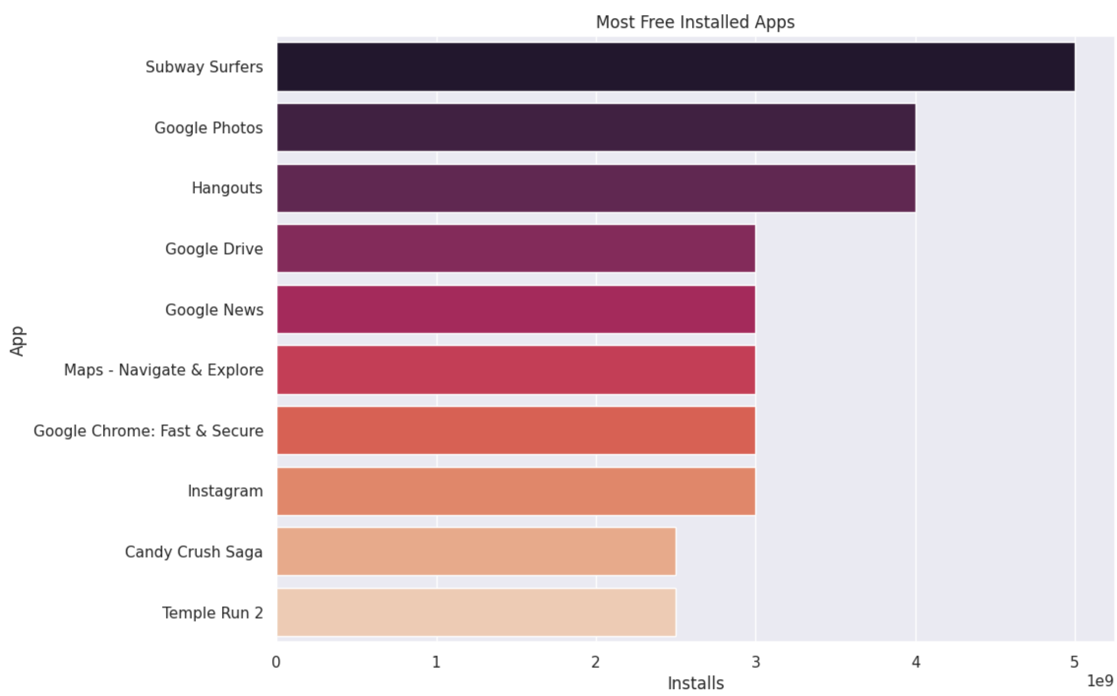
| | Rating | Reviews | Installs | Price |
|---|---|---|---|---|
| count | 8878.000000 | 8.878000e+03 | 8.878000e+03 | 8878.000000 |
| mean | 4.187745 | 4.729619e+05 | 1.649903e+07 | 0.963719 |
| std | 0.522572 | 2.906987e+06 | 8.643798e+07 | 16.201978 |
| min | 1.000000 | 1.000000e+00 | 1.000000e+00 | 0.000000 |
| 25% | 4.000000 | 1.640000e+02 | 1.000000e+04 | 0.000000 |
| 50% | 4.300000 | 4.708000e+03 | 5.000000e+05 | 0.000000 |
| 75% | 4.500000 | 7.119725e+04 | 5.000000e+06 | 0.000000 |

```
max        5.000000  7.815831e+07  1.000000e+09   400.000000
```

# 13 Top 10 Free install apps

[ ]:
```
top_10_free_apps = df[df['Type'] == 'Free'].groupby('App')['Installs'].sum().
  ↪sort_values(ascending = False).head(10)
plt.figure(figsize=(11,8))
sns.barplot(x = top_10_free_apps.values, y = top_10_free_apps.index,␣
  ↪palette="rocket")
plt.xlabel('Installs')
plt.ylabel('App')
plt.title('Most Free Installed Apps')
plt.show()
```

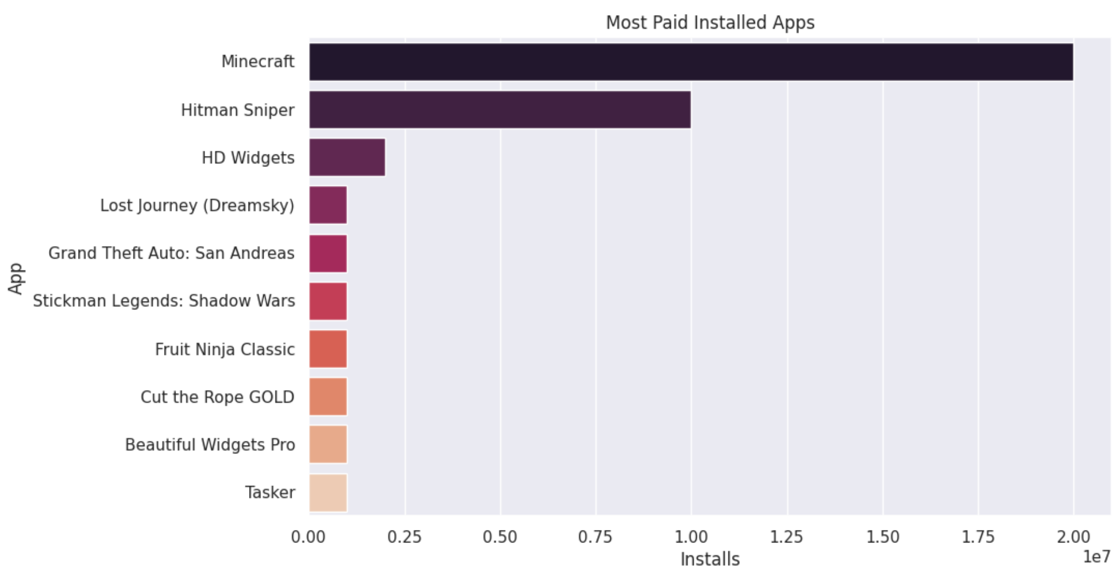
```
<ipython-input-167-b9841db9e15c>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x = top_10_free_apps.values, y = top_10_free_apps.index,
palette="rocket")
```



## 14 Most Paid Installed Apps

[ ]:
```
top_10_paid_apps = df[df['Type'] == 'Paid'].groupby('App')['Installs'].sum().
    ↪sort_values(ascending = False).head(10)
plt.figure(figsize=(10,6))
sns.barplot(x = top_10_paid_apps.values, y = top_10_paid_apps.index,␣
    ↪palette="rocket")
plt.xlabel('Installs')
plt.ylabel('App')
plt.title('Most Paid Installed Apps')
plt.show()
```

```
<ipython-input-168-37f173fab1c1>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x = top_10_paid_apps.values, y = top_10_paid_apps.index,
palette="rocket")
```




### 14.1 Top 20 most expensive apps

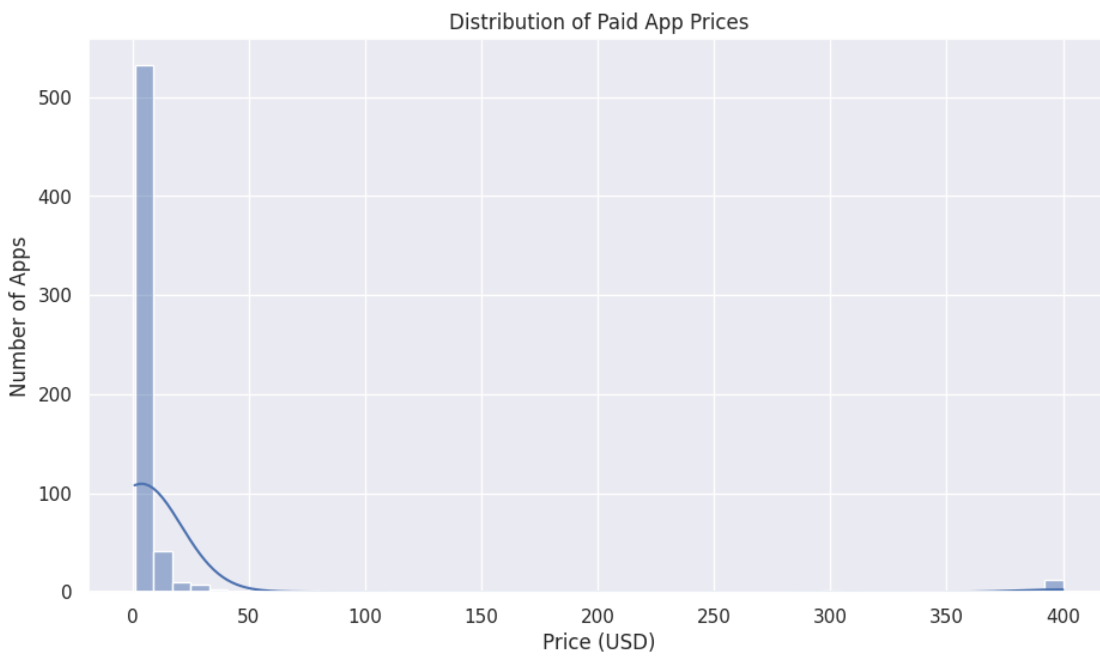
[ ]:
```
# Top 20 most expensive apps
df.nlargest(20, 'Price')[['App', 'Price']]
```

[ ]:
```
                                  App   Price
4367         I'm Rich - Trump Edition  400.00
```

```
4197          most expensive app (H)  399.99
4362                        I'm rich  399.99
5351                       I am rich  399.99
5354                  I am Rich Plus  399.99
5356               I Am Rich Premium  399.99
5358                      I am Rich!  399.99
5359                I am rich(premium)  399.99
5362                  I Am Rich Pro  399.99
5364  I am rich (Most expensive app)  399.99
5369                       I am Rich  399.99
5373               I AM RICH PRO PLUS  399.99
5366                       I Am Rich  389.99
5357              I am extremely Rich  379.99
5355                   I am rich VIP  299.99
2253        Vargo Anesthesia Mega App   79.99
2414                    LTC AS Legal   39.99
5360                 I am Rich Person   37.99
2301          A Manual of Acupuncture   33.99
2266                        EMT PASS   29.99
```

### 14.2 Distribution of Paid App Prices

[ ]:
```
plt.figure(figsize=(11,6))
paid = df[df['Price'] > 0]
sns.histplot(paid['Price'], bins=50, kde=True)
plt.xlabel('Price (USD)')
plt.ylabel('Number of Apps')
plt.title('Distribution of Paid App Prices')
plt.show()
```




## 15 Corelation Analysis

```
[ ]: numerical_df = df[['Rating','Reviews','Installs','Price']]
     numerical_df
```

```
[ ]:        Rating   Reviews  Installs  Price
     0         4.1     159.0     10000    0.0
     1         3.9     967.0    500000    0.0
     2         4.7   87510.0   5000000    0.0
     3         4.5  215644.0  50000000    0.0
     4         4.3     967.0    100000    0.0
     …          …         …         …      …
     10834     4.0       7.0       500    0.0
     10836     4.5      38.0      5000    0.0
     10837     5.0       4.0       100    0.0
     10839     4.5     114.0      1000    0.0
     10840     4.5  398307.0  10000000    0.0

     [8878 rows x 4 columns]
```

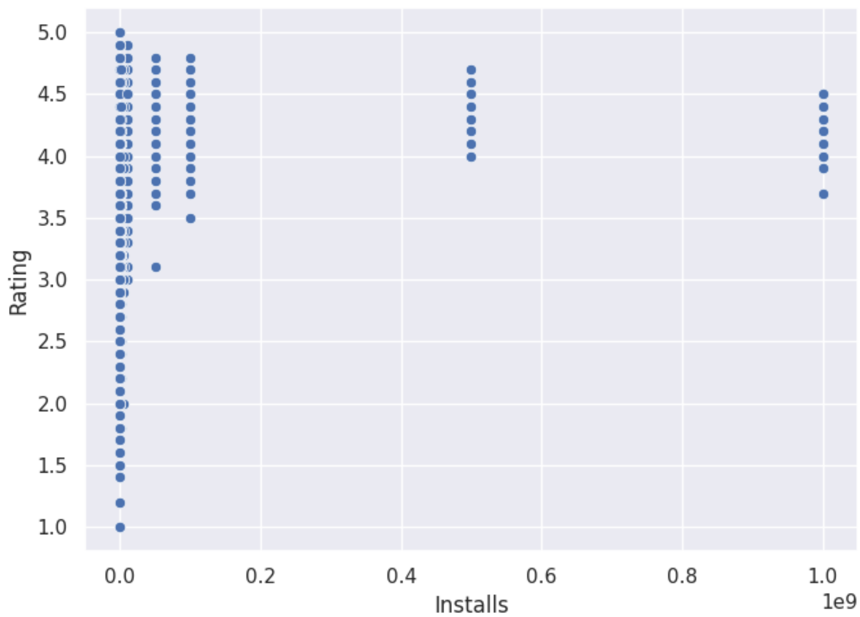
### 15.1 Exploring relationship between numerical_df columns

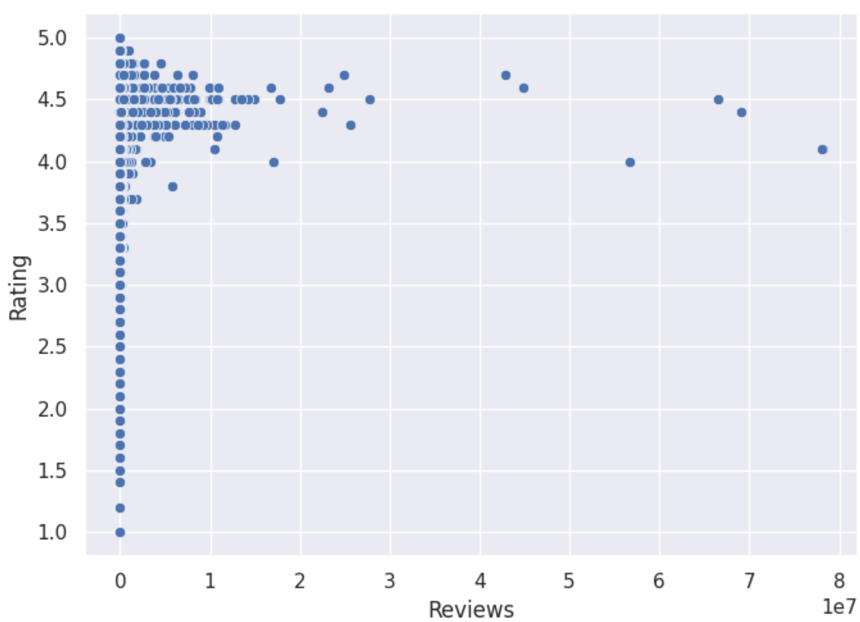
```
[ ]: sns.heatmap(numerical_df.corr(), annot=True)
```

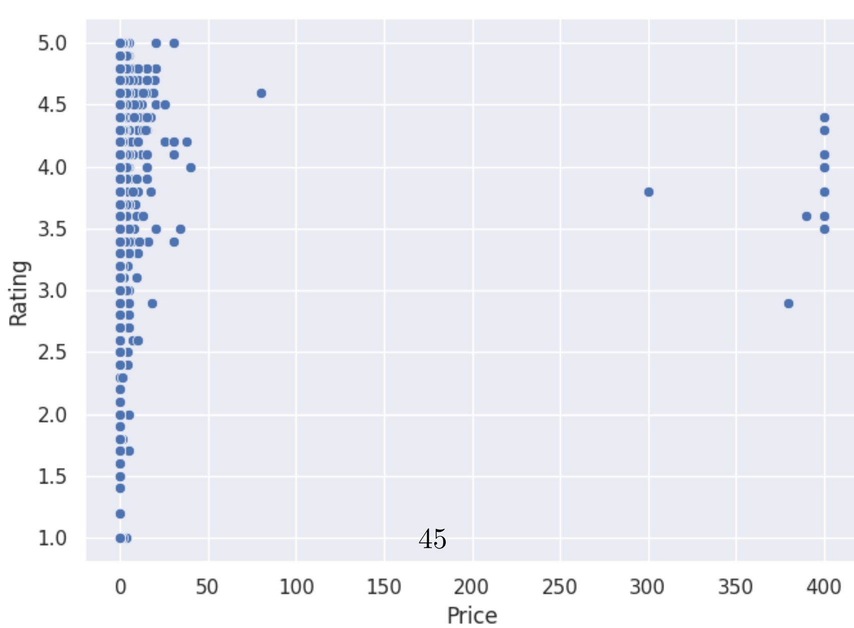
[ ]: <Axes: >


[ ]:
```python
# Import necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Define features and target values
features = ['Installs', 'Reviews', 'Price']
target = 'Rating'

# Create subplots
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(7, 15))

# Create scatter plot for each feature against the target
for i, feature in enumerate(features):
    sns.scatterplot(x=feature, y=target, data=df, ax=axes[i])
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel(target)

# Adjust layout and display the plot
plt.tight_layout()
```

```
plt.show()
```


5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Rating
0.0
0.2
0.4
0.6
0.8
1.0
Installs
1e9


5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Rating
0
1
2
3
4
5
6
7
8
Reviews
1e7


5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Rating
0
50
100
150
200
250
300
350
400
Price

## 15.2 Summary

**Reviews column has higher correlation with Installs column. Number of Reviews increases with the number of Installs, which is obvious.**

**The majority of apps are distributed around a rating of 4.The 'Family' category has the most apps in this data. 'Game' category is next with the second highest number.**

**In this data, Games have the most installations. Communication and social apps come next in terms of how many people have them.Majority of apps are free (about 92.63%) as compared to paid apps.**

**The 'Events' category has the hisghest average rating while the 'Dating' category receives lowest average rating.**

**In this dataset, Many apps worked with Android version 4.1 and newer.The paid apps have the highest average rating as compared to free apps.**

**Most apps have a content rating of 'Everyone', but there are only two apps without any rating**

# 16 4 Model building and evaluation

#Linear Classification With library

Following code performs binary classification on a dataset of mobile apps from the Google Play Store. Here's a breakdown of what each part of the code does:

**Data Preprocessing:**

The dataset is loaded from a CSV file. Non-numeric values in the 'Size' column are converted to numeric format. Non-numeric values in the 'Installs' column are converted to numeric format. Rows with missing values are dropped. The 'Rating' column is converted to a binary classification task by categorizing ratings greater than or equal to 4 as 'Good' and ratings less than 4 as 'Bad'.

**Feature Selection:**

Features (X) are selected, including 'Reviews', 'Size', 'Installs', and 'Price'. The target variable (y) is set as the binary 'Rating' column. Train-Test Split:

The dataset is split into training and testing sets using a 80-20 split ratio. Model Building:

A Logistic Regression model is instantiated.

**Model Trainng**:

The Logistic Regression model is trained on the training data (X_train, y_train). Model Evaluation:

The trained model is used to make predictions on the test data (X_test). The accuracy of the model is calculated using the predicted labels and the actual labels (y_test). A classification report is generated, which includes precision, recall, F1-score, and support for each class ('Good' and 'Bad'). Output:

The accuracy score and the classification report are printed to evaluate the performance of the model. Overall, this code demonstrates how to perform binary classification to predict whether a mobile app has a 'Good' or 'Bad' rating based on its features, using a Logistic Regression model. Let me know if you need further clarification or assistance with any part of the code!

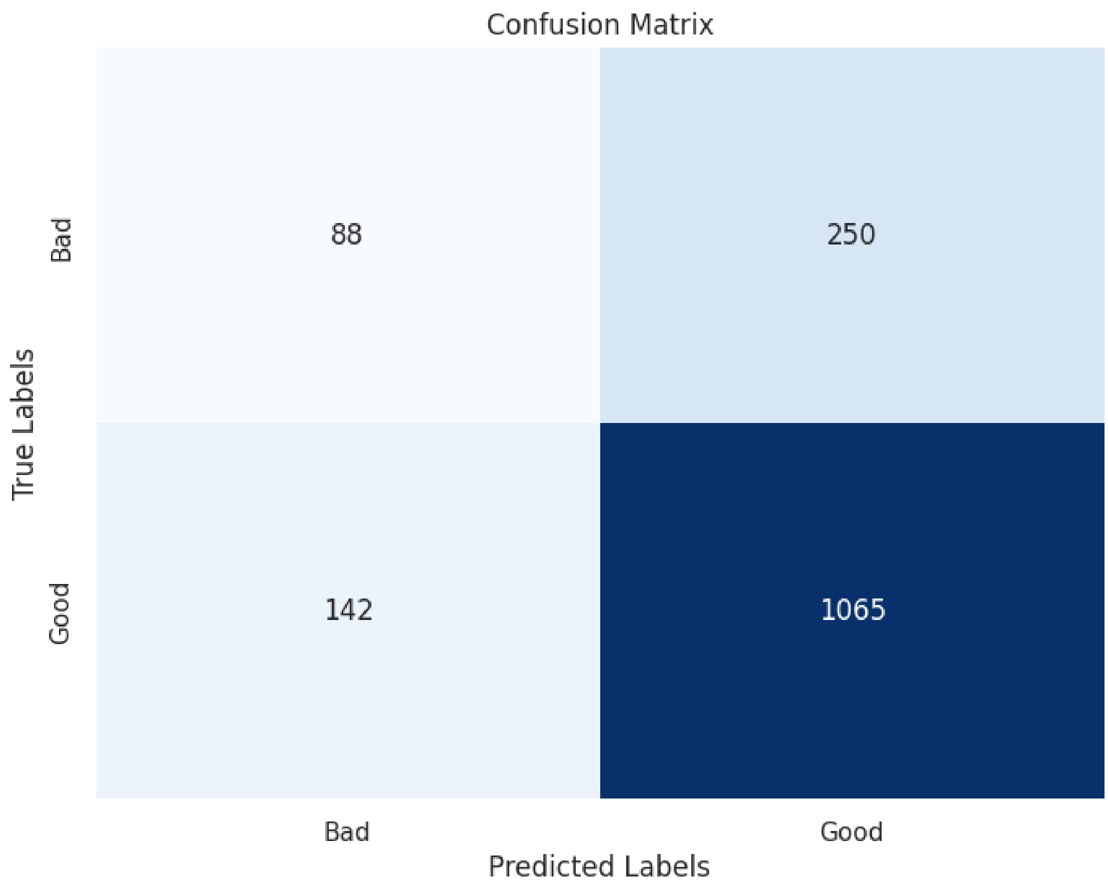
[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
df = pd.read_csv('/content/googleplaystore.csv')

# Handle non-numeric values in 'Size' column
df['Size'] = df['Size'].apply(lambda x: float(x.replace('M', '')) if 'M' in x␣
 ↪else float(x.replace('k', '')) / 1000 if 'k' in x else None)

# Handle non-numeric values in 'Installs' column
df['Installs'] = df['Installs'].apply(lambda x: float(x.replace('+', '').
 ↪replace(',', '')) if '+' in x or ',' in x else None)

# Drop rows with missing values
df.dropna(inplace=True)

df["Price"]= df["Price"].str.replace("$", "")
# Convert 'Rating' to a binary classification task
df['Rating'] = df['Rating'].apply(lambda x: 'Good' if x >= 4 else 'Bad')

# Define features (X) and target variable (y)
X = df[['Reviews', 'Size', 'Installs', 'Price']]
y = df['Rating']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Create a linear classification model (Perceptron)
linear_model = Perceptron()

# Fit the model to the training data
linear_model.fit(X_train, y_train)

# Make predictions on the test data
```

```
y_pred = linear_model.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.7462783171521036
Classification Report:
              precision    recall  f1-score   support

         Bad       0.38      0.26      0.31       338
        Good       0.81      0.88      0.84      1207

    accuracy                           0.75      1545
   macro avg       0.60      0.57      0.58      1545
weighted avg       0.72      0.75      0.73      1545
```

#Evaluation This code generates a confusion matrix for a classification model predicting app ratings as either "Good" or "Bad". The matrix showcases the model's performance by comparing its predictions against the actual ratings in the test dataset. Each row represents the actual ratings, while each column signifies the predicted ratings. The values within the matrix denote the frequency of instances where a specific rating was predicted compared to its actual occurrence. Annotations within the cells provide a numerical representation of these frequencies. The heatmap visualization aids in interpreting the confusion matrix, with color intensity indicating higher or lower frequencies. This concise and informative representation offers insights into the classification model's accuracy and its ability to correctly classify app ratings.

[ ]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=linear_model.classes_, yticklabels=linear_model.
 ↪classes_)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```




#Manually linear classification

### 16.1 Basic and reference implementation:

#### 16.1.1 Linear Classification Process:

**Initialization: Initialize the weights and bias to arbitrary values.**

**Training:** Given a training dataset with labeled examples, adjust the weights and bias iteratively to minimize the classification error. The training process typically involves an optimization algorithm such as gradient descent, which updates the weights and bias in the direction that reduces the classification error. #### Decision Making:

Once the model is trained, it can be used to predict the class labels of new, unseen data points. #### For a given input data point, compute the linear combination of features using the learned weights and bias. Apply a threshold function (e.g., step function) to the linear combination to determine the predicted class label.

```
import numpy as np

class Perceptron:
```

```
    def __init__(self, learning_rate=0.01, n_iterations=1000):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.weights = None
        self.bias = None

    def train(self, X, y):
        n_samples, n_features = X.shape
        # Initialize weights and bias
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iterations):
            for i in range(n_samples):
                # Compute prediction
                y_pred = self.predict(X[i])
                # Update weights and bias based on prediction error
                self.weights += self.learning_rate * (y[i] - y_pred) * X[i]
                self.bias += self.learning_rate * (y[i] - y_pred)

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        # Apply step function for binary classification
        return np.where(linear_output >= 0, 1, 0)

# Example usage:
# Define training data (X) and labels (y)
X_train = np.array([[2, 3], [1, 2], [3, 4], [5, 6]])
y_train = np.array([1, 0, 1, 0])

# Initialize and train the Perceptron model
model = Perceptron()
model.train(X_train, y_train)

# Define test data
X_test = np.array([[4, 5], [1, 1]])

# Make predictions
predictions = model.predict(X_test)
print("Predictions:", predictions)
```

#### 16.1.2 Example:

**Consider a binary classification problem where we want to predict whether an email is spam (1) or not spam (0) based on two features: the number of words related to finance and the number of words related to health.** The decision boundary could be a straight line in the feature space, separating emails about finance from those about health. The weights and bias determine the orientation and position of this line, respectively. ### Limitations:

#### Linear classification assumes that the classes are linearly separable, which may not be true for complex datasets. #### It may not capture nonlinear relationships between features and target variables effectively.

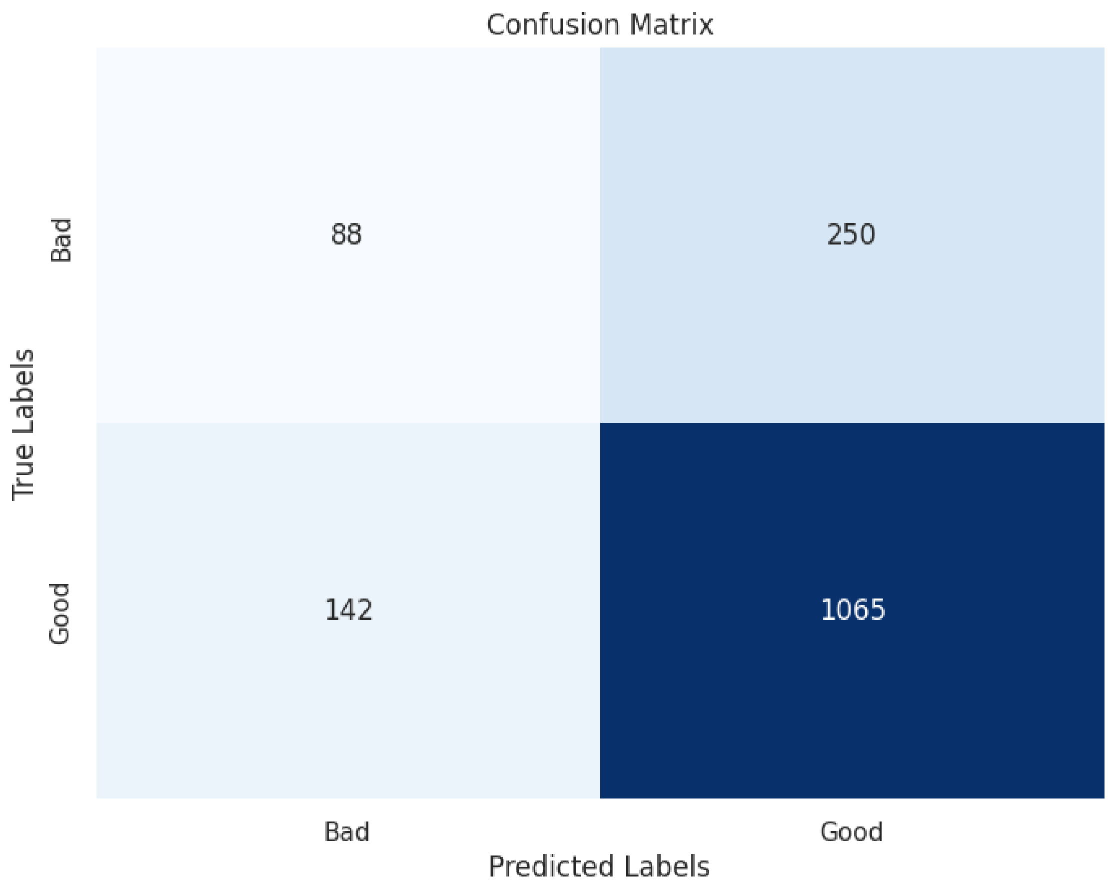
[ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Define the LinearClassifier class
class LinearClassifier:
    def __init__(self, learning_rate=0.01, n_iterations=1000):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iterations):
            for i in range(n_samples):
                linear_model = np.dot(X[i], self.weights) + self.bias
                activation = 1 if linear_model >= 0 else 0
                update = self.learning_rate * (y[i] - activation)
                self.weights += update * X[i]
                self.bias += update

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        activation = np.where(linear_model >= 0, 1, 0)
        return activation

# Load the dataset
df = pd.read_csv('/content/googleplaystore.csv')

# Handle non-numeric values in 'Size' column
df['Size'] = df['Size'].apply(lambda x: float(x.replace('M', '')) if 'M' in x␣
↪else float(x.replace('k', '')) / 1000 if 'k' in x else None)

# Handle non-numeric values in 'Installs' column
df['Installs'] = df['Installs'].apply(lambda x: float(x.replace('+', '').
↪replace(',', '')) if '+' in x or ',' in x else None)

# Handle non-numeric values and invalid values in 'Price' column
df['Price'] = df['Price'].apply(lambda x: float(x.replace('$', '')) if '$' in x␣
↪else None)
df = df.dropna(subset=['Price'])  # Drop rows with missing or invalid prices
```

```
# Convert 'Rating' to a binary classification task
df['Rating'] = df['Rating'].apply(lambda x: 1 if x >= 4 else 0)

# Define features (X) and target variable (y)
X = df[['Reviews', 'Size', 'Installs', 'Price']].values.astype(float)
y = df['Rating'].values.astype(int)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Create an instance of the LinearClassifier class
linear_classifier = LinearClassifier(learning_rate=0.01, n_iterations=1000)

# Train the model
linear_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = linear_classifier.predict(X_test)

# Calculate accuracy
accuracy = np.mean(y_pred == y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.34375
```

## 17 Evaluation

[ ]:
```
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues', cbar=False,
            xticklabels=['Bad', 'Good'], yticklabels=['Bad', 'Good'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```




#Logistic Regression

Here we are building a logistic regression model to predict the rating of mobile applications in the Google Play Store dataset. We first preprocess the data, converting non-numeric values in the 'Size' and 'Installs' columns to numerical format, and convert the 'Price' column to remove the dollar sign. We then transform the 'Rating' column into a binary classification task, labeling **ratings above or equal to 4 as 'Good' and the rest as 'Bad'**. Features such as 'Reviews', 'Size', 'Installs', and 'Price' are used to train the model. After splitting the data into training and testing sets, we fit the logistic regression model to the training data and evaluate its performance using accuracy, a confusion matrix, and classification metrics like precision, recall, and F1 score. Finally, we visualize the confusion matrix to gain insights into the model's predictive capabilities regarding the ratings of the mobile applications.

#using libraraies

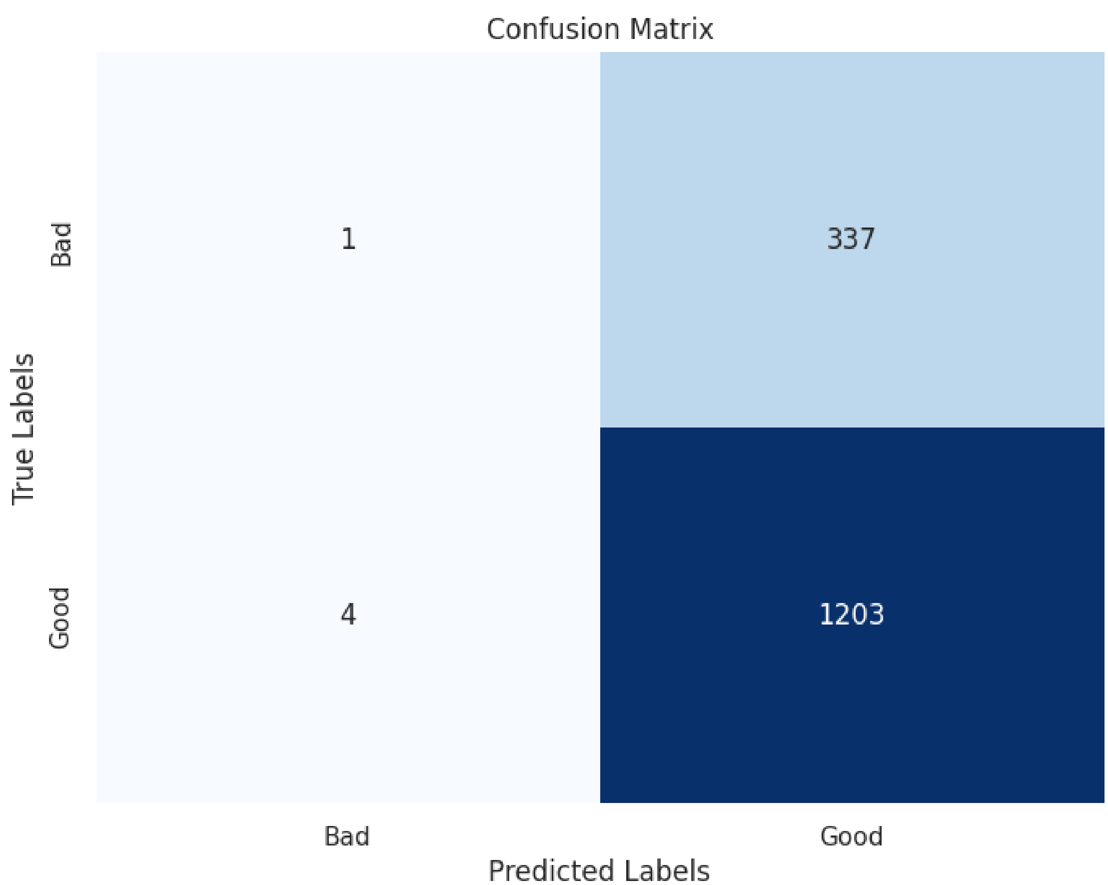
[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import pandas as pd
import seaborn as sns
```

```python
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('/content/googleplaystore.csv')

# Handle non-numeric values in 'Size' column
df['Size'] = df['Size'].apply(lambda x: float(x.replace('M', '')) if 'M' in x␣
↪else float(x.replace('k', '')) / 1000 if 'k' in x else None)

# Handle non-numeric values in 'Installs' column
df['Installs'] = df['Installs'].apply(lambda x: float(x.replace('+', '').
↪replace(',', '')) if '+' in x or ',' in x else None)

# Drop rows with missing values
df.dropna(inplace=True)

df["Price"]= df["Price"].str.replace("$", "")
# Convert 'Rating' to a binary classification task
df['Rating'] = df['Rating'].apply(lambda x: 'Good' if x >= 4 else 'Bad')

# Define features (X) and target variable (y)
X = df[['Reviews', 'Size', 'Installs', 'Price']]
y = df['Rating']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Initialize the logistic regression model
model = LogisticRegression(max_iter=200)  # Increase max_iter further if needed

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

# Predict the target labels for the test set
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7792880258899676
```

## 18 Evaluation

[ ]:
```
# Plot confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=model.classes_, yticklabels=model.classes_)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

print(classification_report(y_test, y_pred))
```




```
              precision    recall  f1-score   support

         Bad       0.20      0.00      0.01       338
        Good       0.78      1.00      0.88      1207

    accuracy                           0.78      1545
```

```
   macro avg       0.49      0.50      0.44      1545
weighted avg       0.65      0.78      0.69      1545
```

#Manually

This code implements a logistic regression model from scratch to predict whether mobile applications in the Google Play Store dataset have a 'Good' or 'Bad' rating based on features such as the number of reviews, size, installs, and price. After preprocessing the data and normalizing the features, gradient descent optimization is used to train the model. The trained model is then evaluated on a test set, and its accuracy is calculated. Additionally, a confusion matrix is constructed to visualize the model's performance, showing the counts of true positive, true negative, false positive, and false negative predictions. Finally, a heatmap is generated to display the confusion matrix, providing insights into the model's predictive capabilities regarding the ratings of mobile applications.

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

# Load the dataset
df = pd.read_csv('/content/googleplaystore.csv')

# Handle non-numeric values in 'Size' column
df['Size'] = df['Size'].apply(lambda x: float(x.replace('M', '')) if 'M' in x␣
↪else float(x.replace('k', '')) / 1000 if 'k' in x else None)

# Handle non-numeric values in 'Installs' column
df['Installs'] = df['Installs'].apply(lambda x: float(x.replace('+', '').
↪replace(',', '')) if '+' in x or ',' in x else None)

# Drop rows with missing values
df.dropna(inplace=True)

df["Price"]= df["Price"].str.replace("$", "")
# Convert 'Rating' to a binary classification task
df['Rating'] = df['Rating'].apply(lambda x: 1 if x >= 4 else 0)

# Define features (X) and target variable (y)
X = df[['Reviews', 'Size', 'Installs', 'Price']].values
y = df['Rating'].values.reshape(-1, 1)

# Feature scaling
X_numeric = df[['Reviews', 'Size', 'Installs', 'Price']].astype(float)
X_scaled = (X_numeric - X_numeric.mean()) / X_numeric.std()
X = X_scaled.values
```

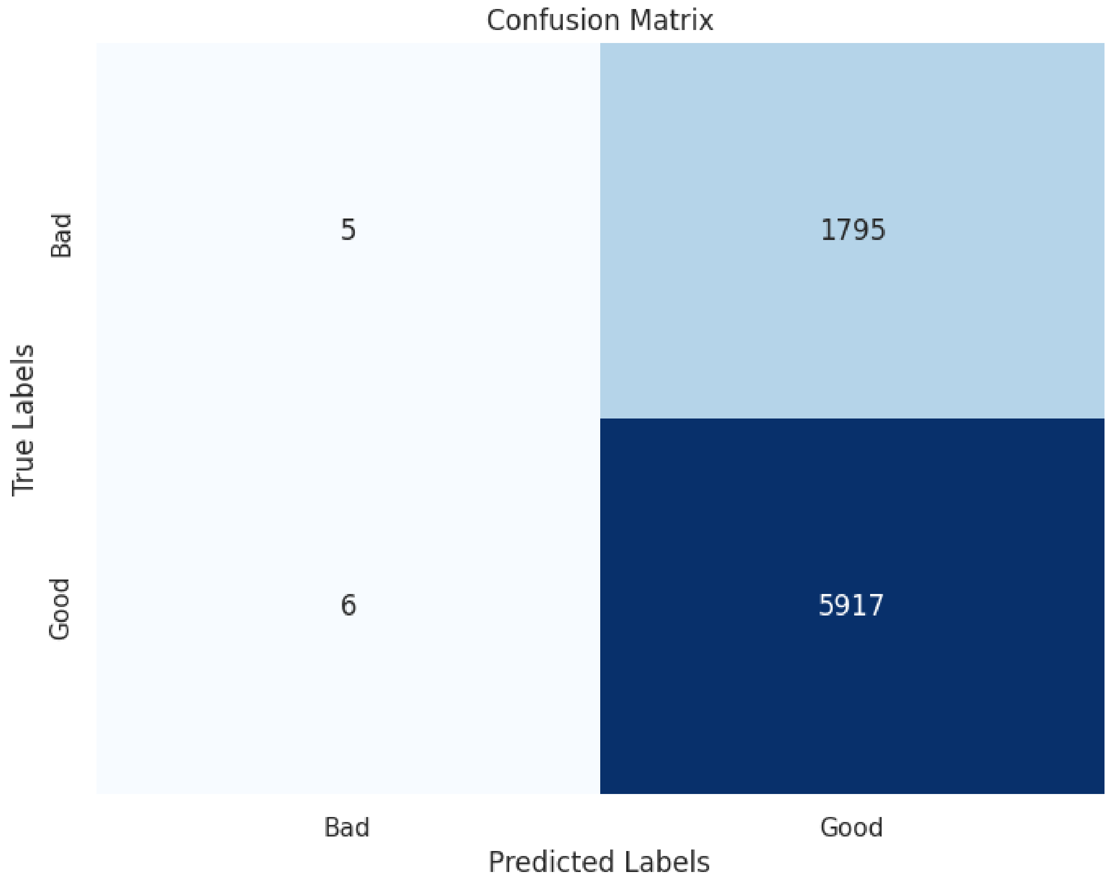
```python
# Add intercept term to X
intercept = np.ones((X.shape[0], 1))
X = np.concatenate((intercept, X), axis=1)

# Define sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Define cost function
def compute_cost(X, y, theta):
    m = len(y)
    h = sigmoid(np.dot(X, theta))
    epsilon = 1e-5
    cost = (1/m) * np.sum(-y * np.log(h + epsilon) - (1 - y) * np.log(1 - h + epsilon))
    return cost

# Define gradient descent function
def gradient_descent(X, y, theta, alpha, iterations):
    m = len(y)
    cost_history = []
    for _ in range(iterations):
        h = sigmoid(np.dot(X, theta))
        gradient = np.dot(X.T, (h - y)) / m
        theta -= alpha * gradient
        cost = compute_cost(X, y, theta)
        cost_history.append(cost)
    return theta, cost_history

# Initialize parameters
theta = np.zeros((X.shape[1], 1))

# Set hyperparameters
alpha = 0.01
iterations = 1000

# Run gradient descent
theta_optimized, cost_history = gradient_descent(X, y, theta, alpha, iterations)

# Predict function
def predict(X, theta):
    return sigmoid(np.dot(X, theta))

# Make predictions
y_pred_proba = predict(X, theta_optimized)
y_pred_class = (y_pred_proba >= 0.5).astype(int)
```

```
# Calculate accuracy
accuracy = np.mean(y_pred_class == y.flatten()) * 100
print("Accuracy:", accuracy)

# Compute confusion matrix
conf_matrix = np.zeros((2, 2))
for true_label, pred_label in zip(y.flatten(), y_pred_class.flatten()):
    conf_matrix[true_label, pred_label] += 1

print("Confusion Matrix:\n", conf_matrix)

# Define labels for the confusion matrix
labels = ['Bad', 'Good']
```

```
Accuracy: 76.61695637849239
Confusion Matrix:
 [[5.000e+00 1.795e+03]
 [6.000e+00 5.917e+03]]
```

# 19 Evaluation

[ ]:
```
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='.0f', cmap='Blues', cbar=False,
            xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```




## 20 Comparison of Linear Classification and Logistic Regression Models:

**Linear Classification:**

- Simpler model, often used as a baseline for classification tasks.
- Easy to understand and interpret.
- Computationally efficient.

**Logistic Regression:**

- More complex model that incorporates the sigmoid function to produce probabilistic outputs.
- Capable of handling non-linear relationships between features and the target variable.
- Provides additional insights through coefficients and the ability to calculate probabilities.

**Comparison:**

- **Accuracy:** Logistic regression typically achieves higher accuracy compared to linear classification, especially when the data exhibits non-linear patterns.
- **Interpretability:** Linear classification is easier to interpret due to its simplicity.
- **Robustness:** Logistic regression is generally more robust to outliers and noise in the data.

- **Computational Efficiency:** Linear classification is computationally faster than logistic regression.

**Recommendation:**

- If the goal is to quickly build a simple and interpretable model, linear classification can be a suitable choice.
- However, if higher accuracy and the ability to handle non-linear relationships are paramount, logistic regression is the preferred choice.
- The choice between the two models should be based on the specific characteristics of the data and the desired outcome of the classification task.

# 21 End