📝 Project Title : Google-Play-Store Analysis with Machine Learning

---

👤 **Name**: [Kumpatla Pavankumar]
🔢 **Reg No**: [12206445]
📚 **Section**: [K22UN]
🎓 **Roll No**: [38]
📘 **Course Name**: [Machine Learning]
💻 **Course Code**: [CSM354] 👨‍🏫 **Submitted to**: [Himanshu Gajanan Tikle]

✅ PROBLEM STATEMENT

- To analyze and predict the performance of Android applications on the Google Play Store using key app features such as category, size, number of installs, price, content rating, and more. The goal is to build a machine learning model that can predict the user rating of an app or classify apps based on their success factors.
- To identify the key factors that drive the popularity of an app on the Google Play Store, by analyzing attributes such as app category, size, content rating, price, and more. The objective is to build a classification model to predict whether an app will cross a specific threshold of installs (e.g., 1 million+ installs), helping developers optimize their app strategy.

✅ Justification for Solving the Problem

In today’s digital economy, mobile applications play a crucial role in engaging users and generating revenue. With over 2.5 million apps available on the Google Play Store, it becomes increasingly important for developers and stakeholders to understand what makes an app **popular** and widely downloaded.

Predicting an app's popularity based on its features—such as category, size, content rating, and pricing—can provide **valuable insights for developers, marketers, and investors**. By identifying patterns that lead to higher install rates, stakeholders can:

- **Optimize app development** strategies (e.g., choosing the right category, pricing, and app size).

- **Increase user engagement and downloads**, leading to better visibility and potential revenue.
- **Support data-driven decisions** for app launches, updates, and monetization models.
- **Reduce market entry risk** by aligning app characteristics with user demand and trends.

From a machine learning perspective, this is a relevant, real-world classification task involving both categorical and numerical data, requiring thoughtful preprocessing, feature engineering, and model selection. It’s also a great case study for building **explainable AI systems** that can justify their predictions based on key app features.

📄 Column Descriptions - Google Play Store Dataset

Column Name	Description
App	Name of the application.
Category	Category under which the app is listed on the Play Store.
Rating	Average user rating of the app (out of 5).
Reviews	Total number of user reviews for the app.
Size	Size of the app (in MB or KB, sometimes 'Varies with device').
Installs	Number of times the app has been installed.
Type	Indicates whether the app is Free or Paid.
Price	Price of the app (if it is paid).
ContentRating	Target age group for the app (e.g., Everyone, Teen).
Genres	Genre(s) of the app (sometimes multiple genres).
LastUpdated	The date when the app was last updated on the Play Store.
CurrentVer	The current version of the app available.
AndroidVer	Minimum required Android version to run the app.

📦 Python Libraries Used (Up to Machine Learning Phase)

🧮 Data Handling

- `pandas` – For loading and manipulating structured data.
- `numpy` – For numerical operations and handling arrays.

📊 Data Visualization

- `matplotlib.pyplot` – For creating static visualizations and plots.
- `seaborn` – For advanced and visually appealing statistical plots.

📅 Date Handling

- `datetime` – For working with date and time features.

⚠️ Miscellaneous

- `warnings` – To suppress warning messages during development.

In [ ]:

```

```

- 📌 Importing Libraries

In [532…

```
import pandas as pd
import numpy as np
```

- 📌 Importing Dataset

In [535…

```
inp0=pd.read_csv("Excels\googleplaystore.csv")
inp0.head(10)
```

Out[535…

	googleplaystore	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unna
0	Unnamed: 1	NaN	NaN	Unnamed:4	Unnamed: 5	
1	App	Category	Rating	Reviews	Size	I
2	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10
3	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500
4	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000
5	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000
6	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100
7	Paper flowers instructions	ART_AND_DESIGN	4.4	167	5.6M	50
8	Smoke Effect Photo Maker - Smoke Editor	ART_AND_DESIGN	3.8	178	19M	50
9	Infinite Painter	ART_AND_DESIGN	4.1	36815	29M	1,000

- 📌 Dimensions of the Google Play Store Dataset

In [538…

```
inp0.shape
```

Out[538…

```
(10843, 13)
```

🧹🧼 Data Cleaning Process ✨

- 📌 Loading Google Play Store Data (Skipping First Two Rows)

In [542…

```
inp0=pd.read_csv("Excels\googleplaystore.csv",skiprows=2)
inp0.head(10)
```

Out[542…

	App	Category	Rating	Reviews	Size	Installs	Type	Pric
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+	Free	
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000,000+	Free	
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100,000+	Free	
5	Paper flowers instructions	ART_AND_DESIGN	4.4	167	5.6M	50,000+	Free	
6	Smoke Effect Photo Maker - Smoke Editor	ART_AND_DESIGN	3.8	178	19M	50,000+	Free	
7	Infinite Painter	ART_AND_DESIGN	4.1	36815	29M	1,000,000+	Free	
8	Garden Coloring Book	ART_AND_DESIGN	4.4	13791	33M	1,000,000+	Free	
9	Kids Paint Free - Drawing Fun	ART_AND_DESIGN	4.7	121	3.1M	10,000+	Free	

- 📌 To get missing in each columns

In [545…

```
inp0.isnull().sum()
```

Out[545…

```
App                  0
Category             0
Rating            1474
Reviews              0
Size                 0
Installs             0
Type                 1
Price                0
ContentRating        1
Genres               0
LastUpdated          0
CurrentVer           8
AndroidVer           3
dtype: int64
```

- 📌 To get more information about the dataset

In [548…

```
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   ContentRating  10840 non-null  object 
 9   Genres         10841 non-null  object 
 10  LastUpdated    10841 non-null  object 
 11  CurrentVer     10833 non-null  object 
 12  AndroidVer     10838 non-null  object 
dtypes: float64(1), object(12)
memory usage: 1.1+ MB
```

- 📌 Summary Statistics of Google Play Store Data

In [551…

```
inp0.describe()
```

Out[551…

	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

- 📌 Calculate Null Values Percentage

In [554…

```
# Assuming df is your DataFrame
null_percentage = (inp0.isnull().sum() / len(inp0)) * 100
# Display the percentage of missing values for each column
print(null_percentage)
```

```
App                0.000000
Category           0.000000
Rating            13.596532
Reviews            0.000000
Size               0.000000
Installs           0.000000
Type               0.009224
Price              0.000000
ContentRating      0.009224
Genres             0.000000
LastUpdated        0.000000
CurrentVer         0.073794
AndroidVer         0.027673
dtype: float64
```

🔄📊 Dividing Columns for More Data 💡

- 📌 Dividing LastUpdated Column into LastUpdated_Month_date and LastUpdated_Year

- 📌 Extracting Month and Date from the "LastUpdated" Column

In [559…

```
inp0["LastUpdated_Month_date"]=inp0.LastUpdated.apply(lambda x: x.split(",")
inp0.head()
```

Out[559…

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

- 📌 Handling Missing Data in 'LastUpdated' by Replacing NaN with Empty String

In [562…

```
inp0["LastUpdated"] = inp0["LastUpdated"].fillna('')
```

- 📌 Extracting the Year from the 'LastUpdated' Column

In [565…

```
inp0['LastUpdated_Year'] = inp0['LastUpdated'].str.split(',').str[1].str.str
inp0.head()
```

Out[565…

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

- 📌 Dropping the 'LastUpdated' Column from the Dataset

In [568…

```
inp0.drop("LastUpdated",axis=1,inplace=True)
inp0.head()
```

Out[568…

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

In [ ]:

In [ ]:

📉 Handling Missing Values for `Rating` Column

🎯 Target Variable: `Rating`

The `Rating` column is the target variable for our machine learning model. It contains some missing values that need to be addressed before training.

🧹 Strategy: Drop Missing Records

Since `Rating` is our target variable and we cannot train or evaluate models on records without a label, we will **drop all rows where the `Rating` is missing**.

> ✅ This ensures that our dataset only includes labeled data, which is essential for supervised learning tasks.

- 📌 Creating Dataset Without Missing Ratings

In [574…
```
inp1=inp0[~inp0.Rating.isnull()]
inp1.shape
```

Out[574…
```
(9367, 14)
```

- 📌 Checking for Null Values in the 'Rating' Column

In [577…
```
#Because we drop the null columns
inp1.Rating.isnull().sum()
```

Out[577…
```
0
```

- 📌 Identifying Null Values in Each Column of the DataFrame

In [580…
```
inp1.isnull().sum()
```

Out[580…
```
App                      0
Category                 0
Rating                   0
Reviews                  0
Size                     0
Installs                 0
Type                     0
Price                    0
ContentRating            1
Genres                   0
CurrentVer               4
AndroidVer               3
LastUpdated_Month_date   0
LastUpdated_Year         1
dtype: int64
```

- ## 📌Explore/understand the Null Values for the column Android version 📌

- 📌 Extracting Rows with Missing 'AndroidVer' Values

In [ ]:
```

```

🚮 **Dropping** 🗑️

- 📌 Retrieving Data for the App at Index 10472

In [587…
```
#to get that column
inp1.loc[10472,:]
```

Out[587…

```
App                          Life Made WI-Fi Touchscreen Photo Frame
Category                                                         1.9
Rating                                                          19.0
Reviews                                                         3.0M
Size                                                          1,000+
Installs                                                        Free
Type                                                               0
Price                                                       Everyone
ContentRating                                                    NaN
Genres                                             February 11, 2018
CurrentVer                                                4.0 and up
AndroidVer                                                       NaN
LastUpdated_Month_date                                        1.0.19
LastUpdated_Year                                                 NaN
Name: 10472, dtype: object
```

- 📌 Extracting Entries with Missing 'AndroidVer' and Category 1.9

In [590…

```
#To get row where the androidver is null and category==1.9
inp1[(inp1['AndroidVer'].isnull() & (inp1.Category=="1.9"))]
```

Out[590…

	App	Category	Rating	Reviews	Size	Installs	Type	Pric
10472	Life Made WI-Fi Touchscreen Photo Frame	1.9	19.0	3.0M	1,000+	Free	0	Everyon

- 📌 Dropping Rows with Missing 'AndroidVer' for Category 1.9

In [593…

```
# we drop that column
inp1=inp1[~(inp1['AndroidVer'].isnull() & (inp1.Category=="1.9"))]
```

- 📌 Extracting Rows Where 'AndroidVer' is Null

In [596…

```
inp1[inp1['AndroidVer'].isnull()]
```

Out[596…

	App	Category	Rating	Reviews	Size	Installs	Type	Pr
4453	[substratum] Vacuum: P	PERSONALIZATION	4.4	230	11M	1,000+	Paid	$1
4490	Pi Dark [substratum]	PERSONALIZATION	4.5	189	2.1M	10,000+	Free	

- 📌 Calculating the Most Frequent Android Version

In [599…

```
# to get the Mode of the column
inp1['AndroidVer'].mode()[0]
```

Out[599… '4.1 and up'

- 📌 Filling the NANs with this values

In [602…
```python
# Filling the missing values with the mode values
inp1['AndroidVer']=inp1['AndroidVer'].fillna(inp1['AndroidVer'].mode()[0])
```

- 📌 Verifying That All Missing 'AndroidVer' Entries Have Been Handled

In [605…
```python
#We get 0 beacause we handle the missing values in AndroidVer column
inp1['AndroidVer'].isnull().sum()
```

Out[605… 0

- 📌 Identifying Null Values in Each Column of the DataFrame

In [608…
```python
inp1.isnull().sum()
```

Out[608…
```
App                     0
Category                0
Rating                  0
Reviews                 0
Size                    0
Installs                0
Type                    0
Price                   0
ContentRating           0
Genres                  0
CurrentVer              4
AndroidVer              0
LastUpdated_Month_date  0
LastUpdated_Year        0
dtype: int64
```

In [ ]:

In [ ]:

🛠️🔍 Handling Missing Values in 'CURRENT VERSION' 🧩

In [613…
```python
inp1['CurrentVer'].value_counts()
```

Out[613…

```
CurrentVer
Varies with device    1415
1                      476
1.1                    206
1.2                    133
2                      129
                      ...
1.3.A.2.9                1
9.9.1.1910               1
7.1.34.28                1
5.9.7                    1
0.3.4                    1
Name: count, Length: 2594, dtype: int64
```

- 📌 Replace the missing valueswith "Varies with devices"

- 📌 Getting Rows with Non-Missing 'CurrentVer' Values

In [617…

```
#to get the Non-null values rows in currentversion
inp1[~inp1['CurrentVer'].isnull()]
```

Out[617…

	App	Category	Rating	Reviews	Size	Install
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000,000+
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100,000+
...	...	...	...	...	...	..
10834	FR Calculator	FAMILY	4.0	7	2.6M	500+
10836	Sya9a Maroc - FR	FAMILY	4.5	38	53M	5,000+
10837	Fr. Mike Schmitz Audio Teachings	FAMILY	5.0	4	3.6M	100+
10839	The SCP Foundation DB fr nn5n	BOOKS_AND_REFERENCE	4.5	114	Varies with device	1,000+
10840	iHoroscope - 2018 Daily Horoscope & Astrology	LIFESTYLE	4.5	398307	19M	10,000,000+

9362 rows × 14 columns

- 📌 Finding the Mode (Most Common Value) in 'CurrentVer'

In [620…

```
# to get the mode in current version
inp1['CurrentVer'].mode()[0]
```

Out[620… 'Varies with device'

- 📌Imputing Null 'CurrentVer' Entries with the Most Frequent Value

In [623…
```
#We set the mode in null Columns
inp1['CurrentVer']=inp1['CurrentVer'].fillna(inp1['CurrentVer'].mode()[0])
```

- 📌Counting Missing Values in 'CurrentVer' After Imputation

In [626…
```
inp1['CurrentVer'].isnull().sum()
```

Out[626… 0

- ### 📌Identifying Null Values in Each Column of the DataFrame

In [629…
```
inp1.isnull().sum()
```

Out[629…
```
App                        0
Category                   0
Rating                     0
Reviews                    0
Size                       0
Installs                   0
Type                       0
Price                      0
ContentRating              0
Genres                     0
CurrentVer                 0
AndroidVer                 0
LastUpdated_Month_date     0
LastUpdated_Year           0
dtype: int64
```

In [ ]:

In [ ]:

🔄🧑‍💻 Change the Variables to the Correct Data Types 🧑‍💻

- 📌Checking Column Data Types in inp1

In [635…
```
inp1.dtypes
```

Out[635…
```
App                          object
Category                     object
Rating                      float64
Reviews                      object
Size                         object
Installs                     object
Type                         object
Price                        object
ContentRating                object
Genres                       object
CurrentVer                   object
AndroidVer                   object
LastUpdated_Month_date       object
LastUpdated_Year             object
dtype: object
```

In [637…
```
inp1.head()
```

Out[637…

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

💰💻 Handling Price Column Data Type 🏷️

- 📌Converting 'Price' Column to Numeric Type After Removing "$" Symbol

In [641…
```
inp1.Price=inp1.Price.apply(lambda x:0 if x=="0" else float(x[1:]))
```

In [643…
```
inp1.Price.dtypes
```

Out[643…
```
dtype('float64')
```

📝🔢 Handle the Reviews Column Data Types 🔧

- 📌Converting 'Reviews' Column to Integer Type

In [647…
```
inp1.Reviews=inp1.Reviews.astype("int32")
```

In [649…
```
inp1.dtypes
```

Out[649…
```
App                          object
Category                     object
Rating                      float64
Reviews                       int32
Size                         object
Installs                     object
Type                         object
Price                       float64
ContentRating                object
Genres                       object
CurrentVer                   object
AndroidVer                   object
LastUpdated_Month_date       object
LastUpdated_Year             object
dtype: object
```

In [ ]:
```

```

🧹📉 Clean Up or Handle the INSTALLS Column 🔧

In [653…
```
inp1.Installs.describe
```

Out[653…
```
<bound method NDFrame.describe of 0            10,000+
1           500,000+
2         5,000,000+
3        50,000,000+
4           100,000+
            ...
10834           500+
10836         5,000+
10837           100+
10839         1,000+
10840    10,000,000+
Name: Installs, Length: 9366, dtype: object>
```

- 📌 Function to Clean 'Installs' Values by Removing Commas and Plus Signs

In [656…
```
def clean_installs(val):
```

```
    return int(val.replace(",","").replace("+",""))
```

- 📌 Cleaning and Converting 'Installs' Value to Integer

In [659…
```
clean_installs("3,000+")
```
Out[659…
```
3000
```

- 📌 Transforming 'Installs' String with Comma and Plus to Integer

In [662…
```
type(clean_installs("300,00+"))
```
Out[662…
```
int
```

- 📌 Data Cleaning for 'Installs' Column with clean_installs Function

In [665…
```
inp1.Installs=inp1.Installs.apply(clean_installs)
```

- 📌 Summary Statistics for the 'Installs' Data

In [668…
```
inp1.Installs.describe
```
Out[668…
```
<bound method NDFrame.describe of 0           10000
1          500000
2         5000000
3        50000000
4          100000
           ...
10834         500
10836        5000
10837         100
10839        1000
10840    10000000
Name: Installs, Length: 9366, dtype: int64>
```

In [ ]:
```

```
In [ ]:
```

```

## 🧽📏 **Cleaning Size Column** 🧹

- 📌 Cleaning 'Size' Column by Converting 'M' and 'K' to Numeric Values

In [674…
```
def clean_size(val):
    if 'M' in val:
        return float(float(val.replace("M", "")) * 1000)
    elif 'K' in val:
```

```
        return float(val.replace("K", ""))
    else:
        return float()
```

- 📌 Converting 'Size' from Megabytes to Kilobytes

In [677…
```
clean_size("19M")
```

Out[677…
```
19000.0
```

- 📌 Applying clean_size Function to the 'Size' Column

In [680…
```
inp1.Size=inp1.Size.apply(clean_size)
```

In [ ]:

In [ ]:

🚫❌ Removing Rows with 'Unrated' and 'Adults Only 18+' Content Ratings 🔞

In [685…
```
inp1 = inp1[~inp1['ContentRating'].isin(['Unrated', 'Adults only 18+'])]
```

In [ ]:

In [ ]:

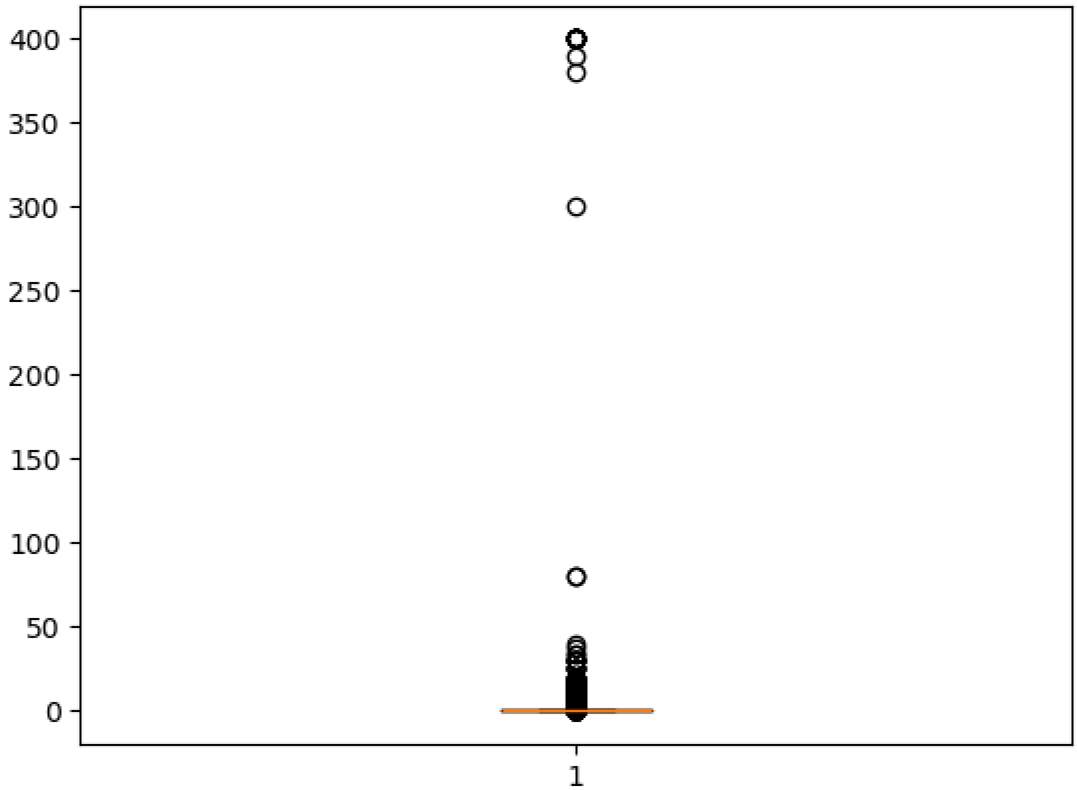
🔍📊 Identifying and Handling Outliers/Extreme Values ⚖️

- 📌 Importing libraries for visulization

In [691…
```
import matplotlib.pyplot as plt
%matplotlib inline
```

- 📌 Visualizing the Distribution of 'Price' Column Using a Boxplot

In [694…
```
plt.boxplot(inp1.Price)
plt.show()
```



- 📌 Identifying Apps with Prices Above 200

In [697…

```
inp1[inp1.Price>200]
```

Out[697…

	App	Category	Rating	Reviews	Size	Installs	Type	Price
4197	most expensive app (H)	FAMILY	4.3	6	1500.0	100	Paid	399.9
4362	💎 I'm rich	LIFESTYLE	3.8	718	26000.0	10000	Paid	399.9
4367	I'm Rich - Trump Edition	LIFESTYLE	3.6	275	7300.0	10000	Paid	400.0
5351	I am rich	LIFESTYLE	3.8	3547	1800.0	100000	Paid	399.9
5354	I am Rich Plus	FAMILY	4.0	856	8700.0	10000	Paid	399.9
5355	I am rich VIP	LIFESTYLE	3.8	411	2600.0	10000	Paid	299.9
5356	I Am Rich Premium	FINANCE	4.1	1867	4700.0	50000	Paid	399.9
5357	I am extremely Rich	LIFESTYLE	2.9	41	2900.0	1000	Paid	379.9
5358	I am Rich!	FINANCE	3.8	93	22000.0	1000	Paid	399.9
5359	I am rich(premium)	FINANCE	3.5	472	0.0	5000	Paid	399.9
5362	I Am Rich Pro	FAMILY	4.4	201	2700.0	5000	Paid	399.9
5364	I am rich (Most expensive app)	FINANCE	4.1	129	2700.0	1000	Paid	399.9
5366	I Am Rich	FAMILY	3.6	217	4900.0	10000	Paid	389.9
5369	I am Rich	FINANCE	4.3	180	3800.0	5000	Paid	399.9
5373	I AM RICH PRO PLUS	FINANCE	4.0	36	41000.0	1000	Paid	399.9

- 📌 Filtering Out Apps Priced Above 200

In [700…

```
inp1=inp1[~(inp1.Price>200)]
```

- 📌 Identifying Apps with Price Greater Than 200

In [703…

```
inp1[inp1.Price>200]
```

Out[703…

App	Category	Rating	Reviews	Size	Installs	Type	Price	ContentRating

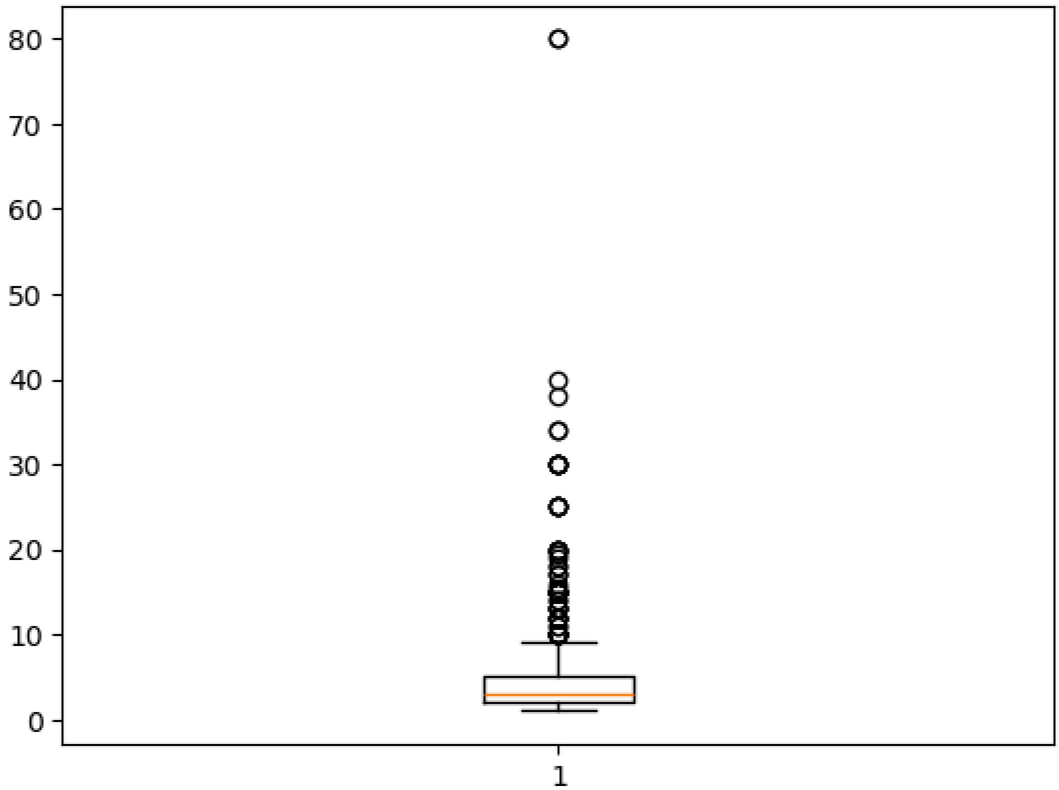
In [ ]:

```

```

Visualizing the Distribution of Positive 'Price' Values

In [707…

```
plt.boxplot(inp1[inp1.Price>0].Price)
plt.show()
```



- Filtering Apps with Price Greater Than 30

In [710…

```
inp1[inp1.Price>30]
```

Out[710…

	App	Category	Rating	Reviews	Size	Installs	Type	Price
2253	Vargo Anesthesia Mega App	MEDICAL	4.6	92	32000.0	1000	Paid	79.99
2301	A Manual of Acupuncture	MEDICAL	3.5	214	68000.0	1000	Paid	33.99
2365	Vargo Anesthesia Mega App	MEDICAL	4.6	92	32000.0	1000	Paid	79.99
2402	A Manual of Acupuncture	MEDICAL	3.5	214	68000.0	1000	Paid	33.99
2414	LTC AS Legal	MEDICAL	4.0	6	1300.0	100	Paid	39.99
5360	I am Rich Person	LIFESTYLE	4.2	134	1800.0	1000	Paid	37.99

- Removing Rows with Price Greater Than 30

In [713…

```
inp1=inp1[~(inp1.Price>30)]
```

- Shape of Dataset for Apps Priced Above 30

In [716…

```
inp1[inp1.Price>30].shape
```

Out[716…

```
(0, 14)
```

In [ ]:

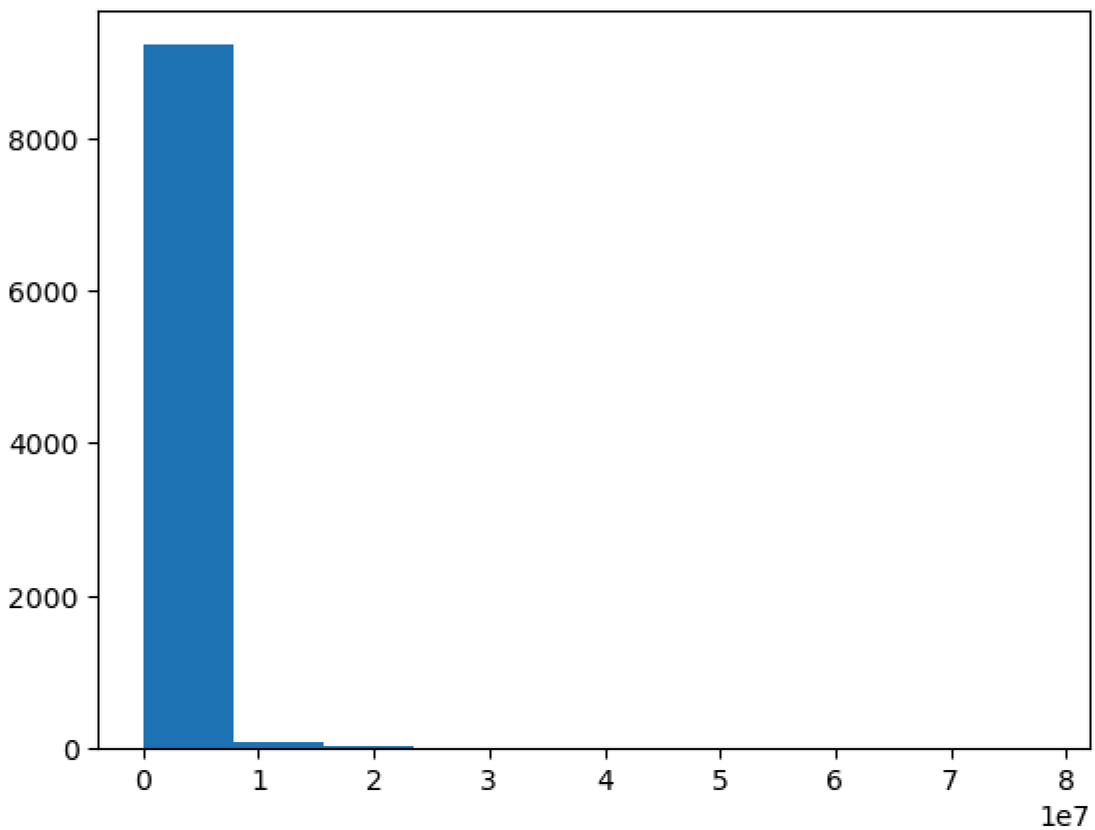
```

```

📊📈 VISUALIZATION 🔍

- Visualizing the Distribution of 'Reviews' with a Histogram

In [721…

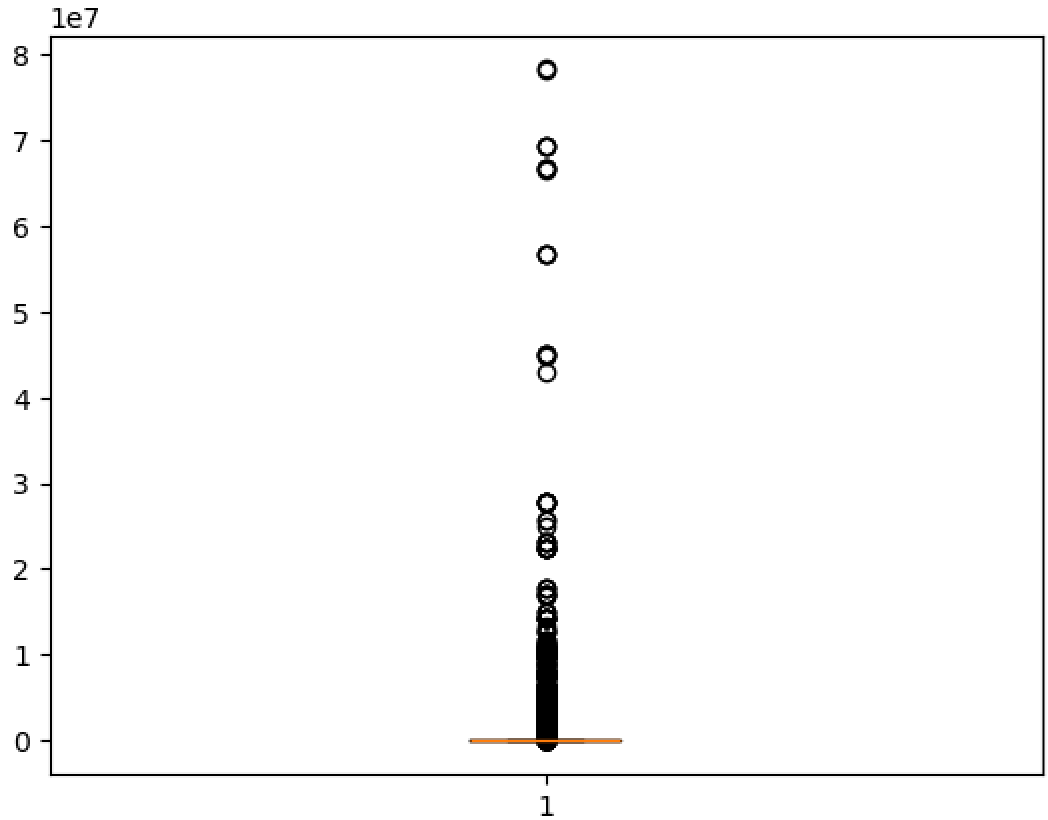
```
plt.hist(inp1.Reviews)
plt.show()
```



- Visualizing the Spread of 'Reviews' Using a Boxplot

In [724…

```
plt.boxplot(inp1.Reviews)
plt.show()
```



- Identifying Apps with Over 10 Million Reviews

In [727…

```
inp1[inp1.Reviews>=10000000]
```

Out[727…

	App	Category	Rating	Reviews	Size	Installs	Type
335	Messenger – Text and Video Chat for Free	COMMUNICATION	4.0	56642847	0.0	1000000000	Free
336	WhatsApp Messenger	COMMUNICATION	4.4	69119316	0.0	1000000000	Free
342	Viber Messenger	COMMUNICATION	4.3	11334799	0.0	500000000	Free
378	UC Browser - Fast Download Private & Secure	COMMUNICATION	4.5	17712922	40000.0	500000000	Free
381	WhatsApp Messenger	COMMUNICATION	4.4	69119316	0.0	1000000000	Free
...	...	...	...	...	...	...	...
6449	BBM - Free Calls & Messages	COMMUNICATION	4.3	12843436	0.0	100000000	Free
7536	Security Master - Antivirus, VPN, AppLock, Boo...	TOOLS	4.7	24900999	0.0	500000000	Free
7937	Shadow Fight 2	GAME	4.6	10981850	88000.0	100000000	Free
8894	Cache Cleaner-DU Speed Booster (booster & clea...	TOOLS	4.5	12759815	15000.0	100000000	Free
8896	DU Battery Saver - Battery Charger & Battery Life	TOOLS	4.5	13479633	14000.0	100000000	Free

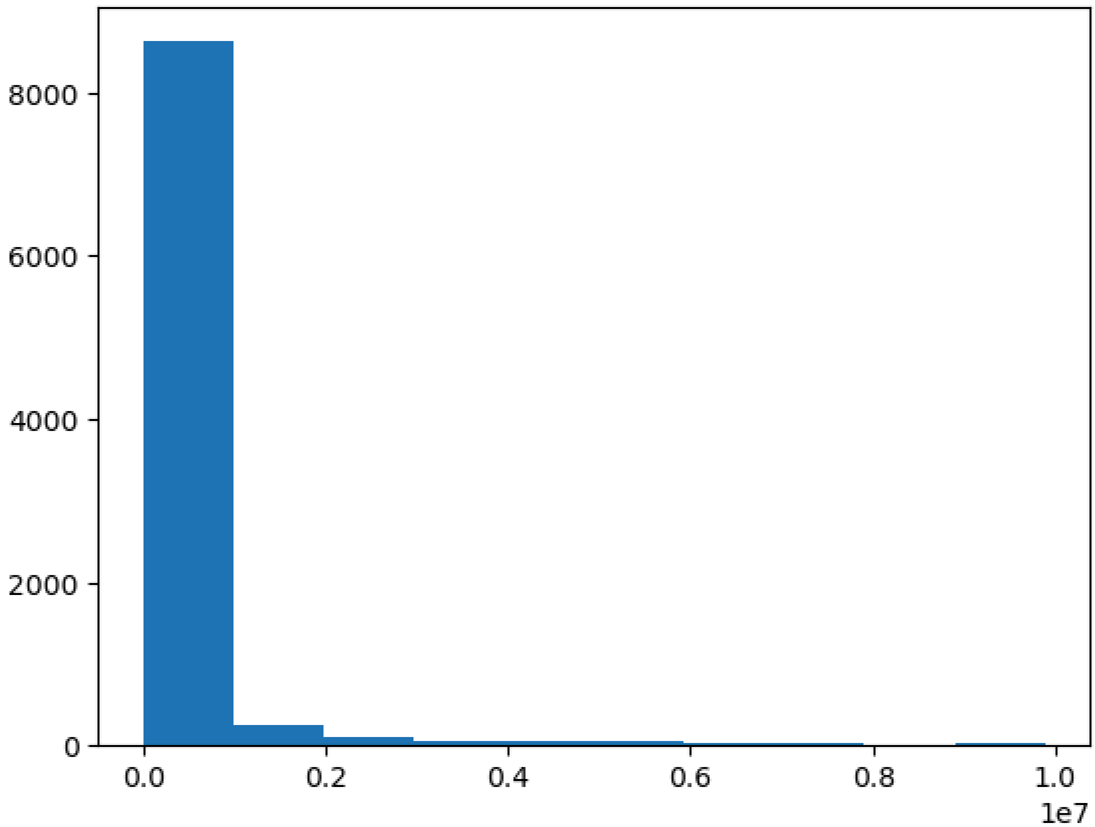
92 rows × 14 columns

- Cleaning Data by Removing Apps with More Than 10 Million Reviews

In [730…
```
inp1=inp1[~(inp1.Reviews>10000000)]
```

- Displaying the Frequency of Review Counts in a Histogram after cleaning

In [733…
```
plt.hist(inp1.Reviews)
plt.show()
```

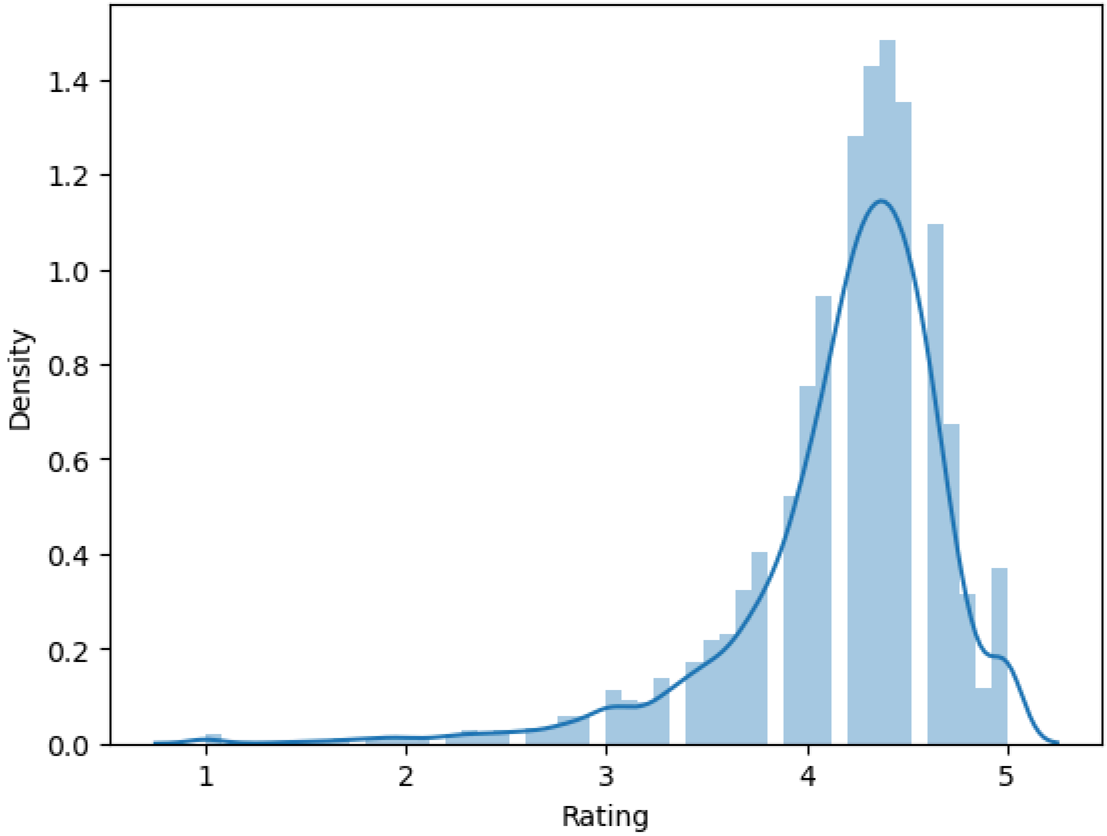


In [734…
```
import warnings
warnings.filterwarnings("ignore")
```

In [737…
```
import seaborn as sns
```

- Understanding the Rating Distribution of Apps Using Seaborn's Distplot

In [740…
```
sns.distplot(inp1.Rating)
plt.show()
```

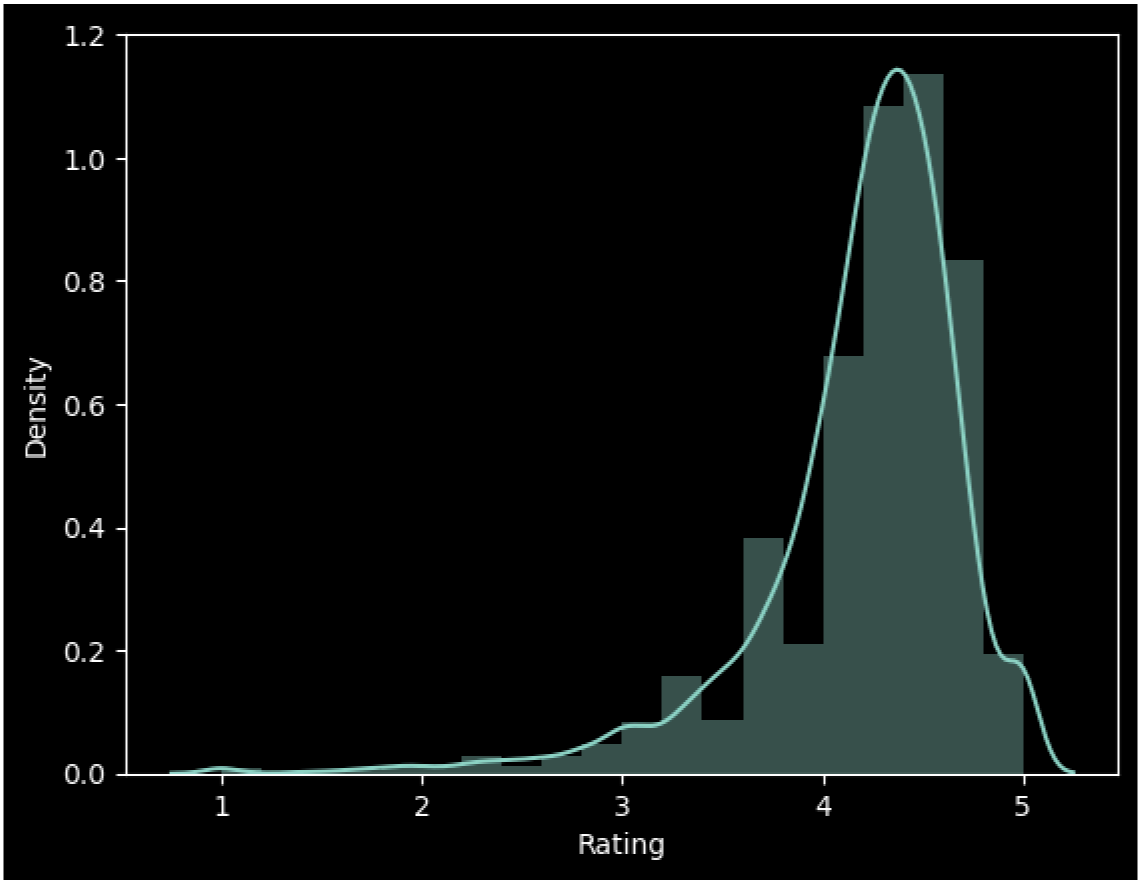


In [741…

```
plt.style.use("dark_background")
```

In [744…

```
sns.distplot(inp1.Rating,bins=20)
plt.show()
```



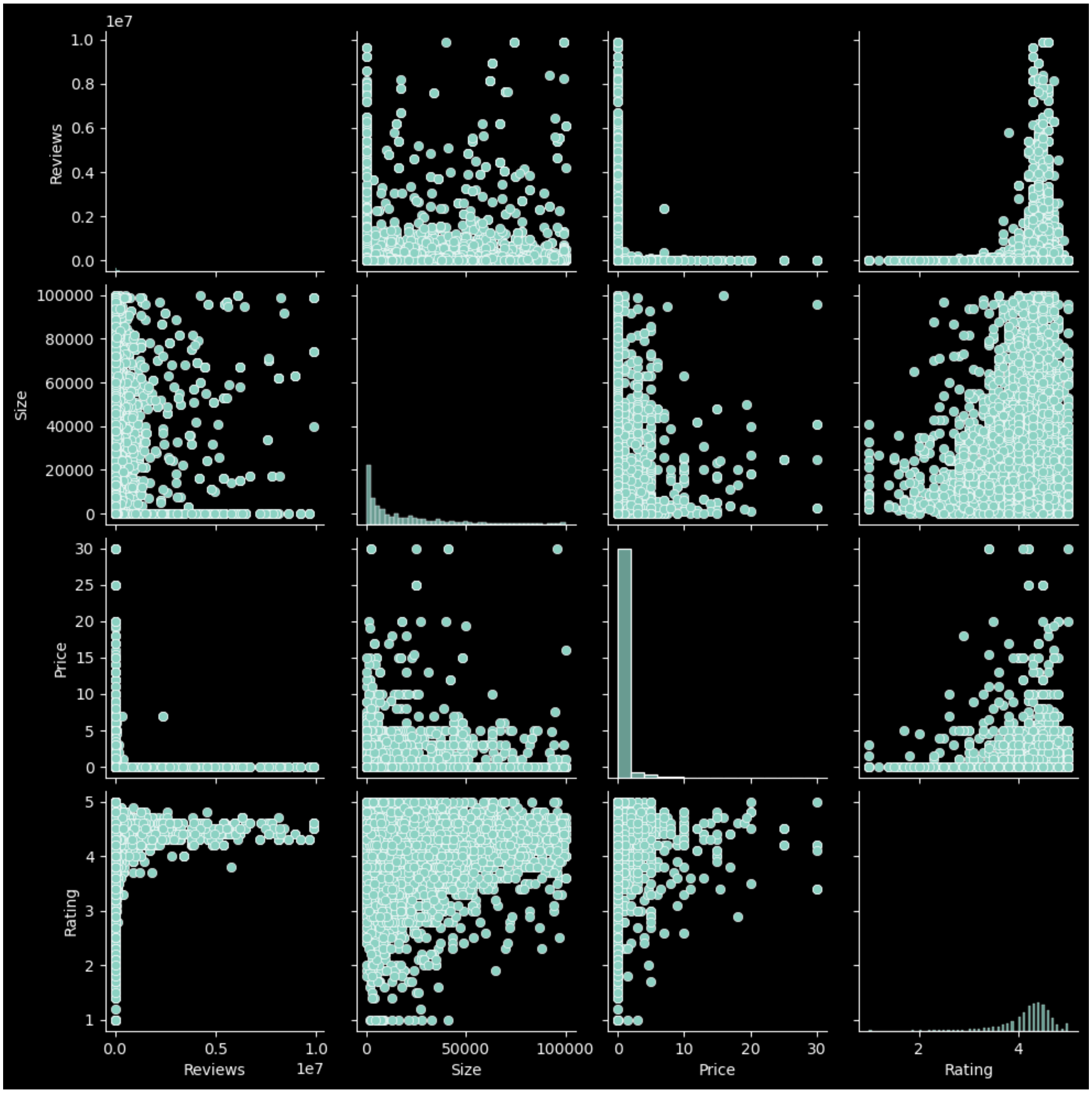
In [ ]:

In [ ]:

⬛🔍 Visualization using Pair Plot 📊

- Visualizing Relationships Between Reviews, Size, Price, and Rating with Seaborn's Pairplot

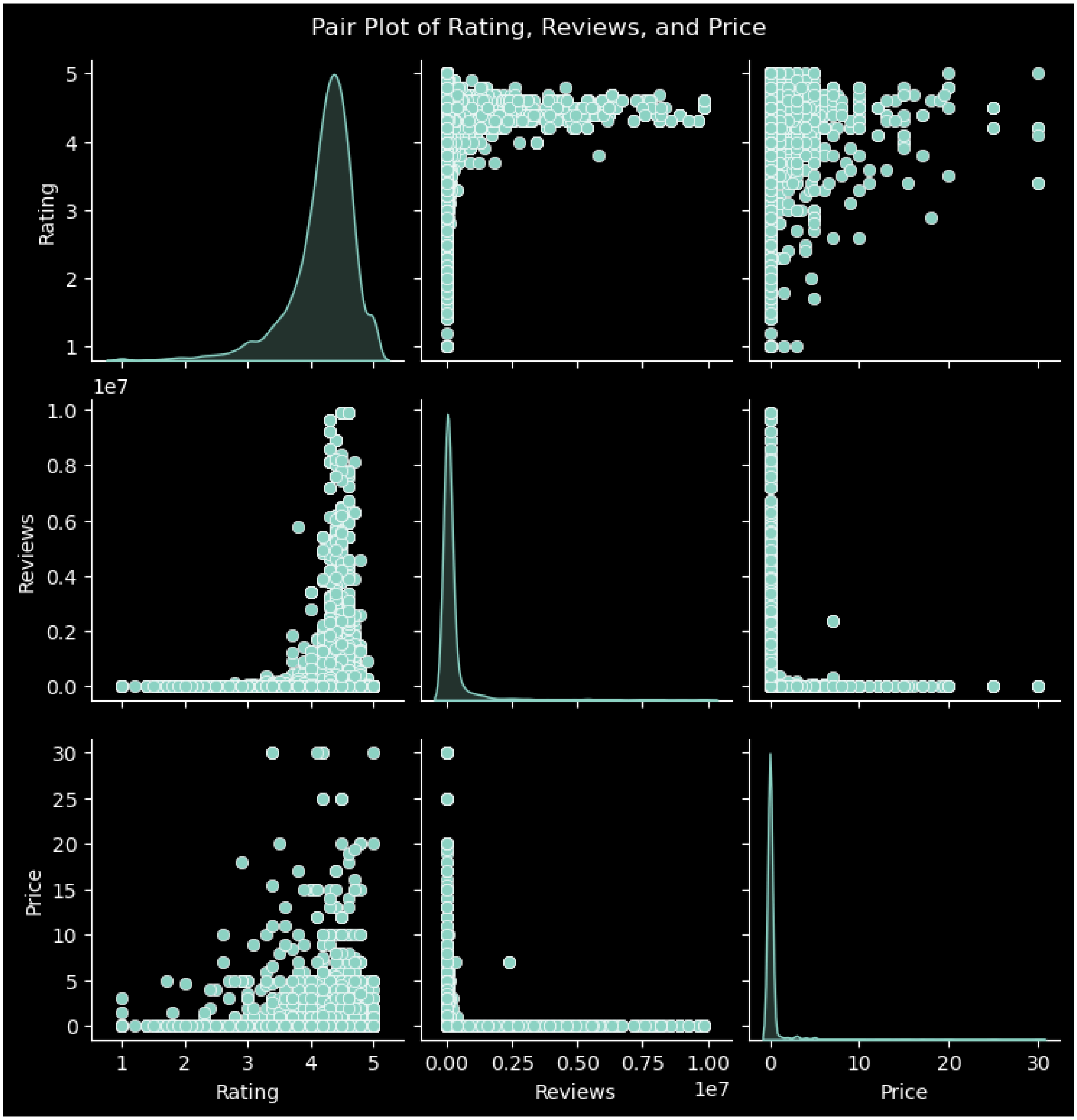
In [750…
```
sns.pairplot(inp1[['Reviews','Size','Price','Rating']])
plt.show()
```



- Visualizing Rating Distribution Across Content Ratings with Seaborn Boxplot

In [752…

```
# Assuming inp1 is your DataFrame
# Create a pair plot for 'Rating', 'Reviews', and 'Price'
sns.pairplot(inp1[['Rating', 'Reviews', 'Price']], diag_kind='kde')

# Add a title
plt.suptitle('Pair Plot of Rating, Reviews, and Price', y=1.02)

# Show the plot
plt.show()
```



In [753…
```
plt.style.use("default")
%matplotlib inline
```

In [754…
```
?pd.qcut
```

```
Signature:
pd.qcut(
    x,
    q,
    labels=None,
    retbins: 'bool' = False,
    precision: 'int' = 3,
    duplicates: 'str' = 'raise',
)
Docstring:
Quantile-based discretization function.

Discretize variable into equal-sized buckets based on rank or based
on sample quantiles. For example 1000 values for 10 quantiles would
produce a Categorical object indicating quantile membership for each data po
int.

Parameters
----------
x : 1d ndarray or Series
q : int or list-like of float
    Number of quantiles. 10 for deciles, 4 for quartiles, etc. Alternately
    array of quantiles, e.g. [0, .25, .5, .75, 1.] for quartiles.
labels : array or False, default None
    Used as labels for the resulting bins. Must be of the same length as
    the resulting bins. If False, return only integer indicators of the
    bins. If True, raises an error.
retbins : bool, optional
    Whether to return the (bins, labels) or not. Can be useful if bins
    is given as a scalar.
precision : int, optional
    The precision at which to store and display the bins labels.
duplicates : {default 'raise', 'drop'}, optional
    If bin edges are not unique, raise ValueError or drop non-uniques.

Returns
-------
out : Categorical or Series or array of integers if labels is False
    The return type (Categorical or Series) depends on the input: a Series
    of type category if input is a Series else Categorical. Bins are
    represented as categories when categorical data is returned.
bins : ndarray of floats
    Returned only if `retbins` is True.

Notes
-----
Out of bounds values will be NA in the resulting Categorical object

Examples
--------
>>> pd.qcut(range(5), 4)
... # doctest: +ELLIPSIS
[(-0.001, 1.0], (-0.001, 1.0], (1.0, 2.0], (2.0, 3.0], (3.0, 4.0]]
Categories (4, interval[float64, right]): [(-0.001, 1.0] < (1.0, 2.0] ...

>>> pd.qcut(range(5), 3, labels=["good", "medium", "bad"])
```

```
... # doctest: +SKIP
[good, good, medium, bad, bad]
Categories (3, object): [good < medium < bad]

>>> pd.qcut(range(5), 4, labels=False)
array([0, 0, 1, 2, 3])
```
File: d:\anconda\lib\site-packages\pandas\core\reshape\tile.py
Type: function

🌡️🔥 Visualization using HEATMAP 🗺️

- Discretizing App Size into Quantile-Based Categories (VL, L, M, H, VH)

In [757…
```
inp1['Size_Bucket']=pd.qcut(inp1.Size,[0,0.2,0.4,0.6,0.8,1],["VL","L","M","H
```

- Inspecting the First Few Entries in the inp1 DataFrame

In [759…
```
inp1.head()
```

Out[759…

	App	Category	Rating	Reviews	Size	Installs	Type	Pric
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19000.0	10000	Free	0.(
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14000.0	500000	Free	0.(
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8700.0	5000000	Free	0.(
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25000.0	50000000	Free	0.(
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2800.0	100000	Free	0.(

- Summary of Ratings by Content Rating and Size Bucket

In [761…
```
pd.pivot_table(data=inp1,index="ContentRating",columns="Size_Bucket",values=
```

Out[761…

Size_Bucket ContentRating	VL	L	M	H	VH
Everyone	4.233801	4.137606	4.176055	4.172851	4.219262
Everyone 10+	4.226316	4.218182	4.257500	4.237755	4.251773
Mature 17+	4.212088	4.096774	4.087129	4.018812	4.194175
Teen	4.260476	4.173387	4.221557	4.187554	4.274850

- Analyzing Median Ratings Across Content Ratings and Size Categories

In [763…

```
pd.pivot_table(data=inp1,index="ContentRating",columns="Size_Bucket",values=
```

Out[763…

Size_Bucket ContentRating	VL	L	M	H	VH
Everyone	4.3	4.20	4.30	4.3	4.3
Everyone 10+	4.3	4.25	4.35	4.3	4.3
Mature 17+	4.2	4.25	4.20	4.1	4.2
Teen	4.3	4.30	4.30	4.2	4.3

- Analyzing App Ratings (Median) Across Different Content Ratings and Size Buckets

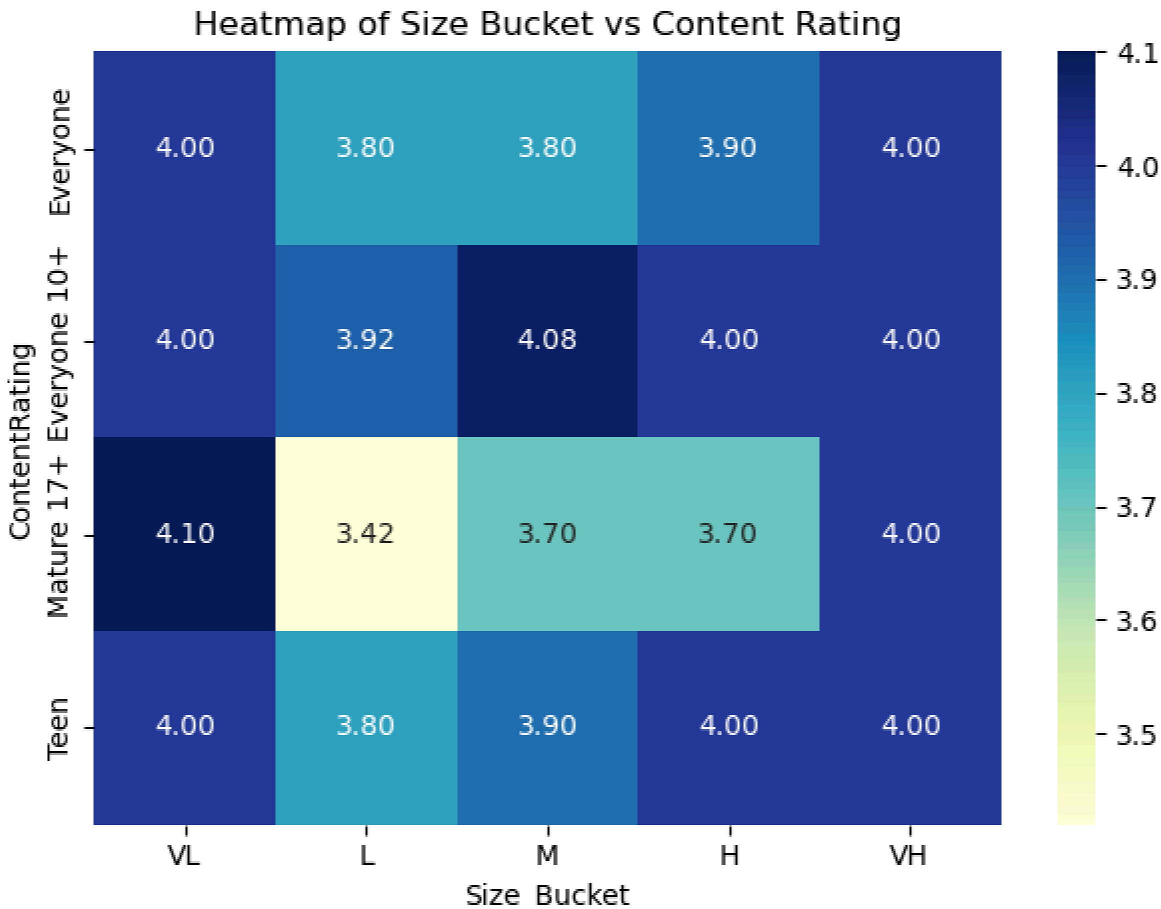
In [765…

```
res=pd.pivot_table(data=inp1,index="ContentRating",columns="Size_Bucket",val
```

In [ ]:

In [767…

```
import seaborn as sns
import matplotlib.pyplot as plt

# Your heatmap creation line
sns.heatmap(res, annot=True, fmt='.2f', cmap='YlGnBu')

# Display the heatmap
plt.title('Heatmap of Size Bucket vs Content Rating')
plt.xlabel('Size_Bucket')
plt.ylabel('ContentRating')
plt.tight_layout()
plt.show()
```



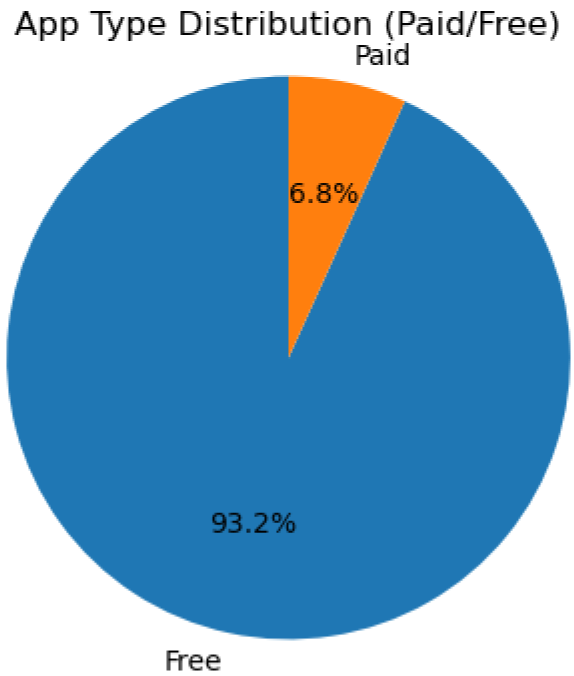
In [ ]:

🥧📊 Visualization using PIE-CHART 🍰

- Distribution of App Types (Paid vs Free)

In [771…

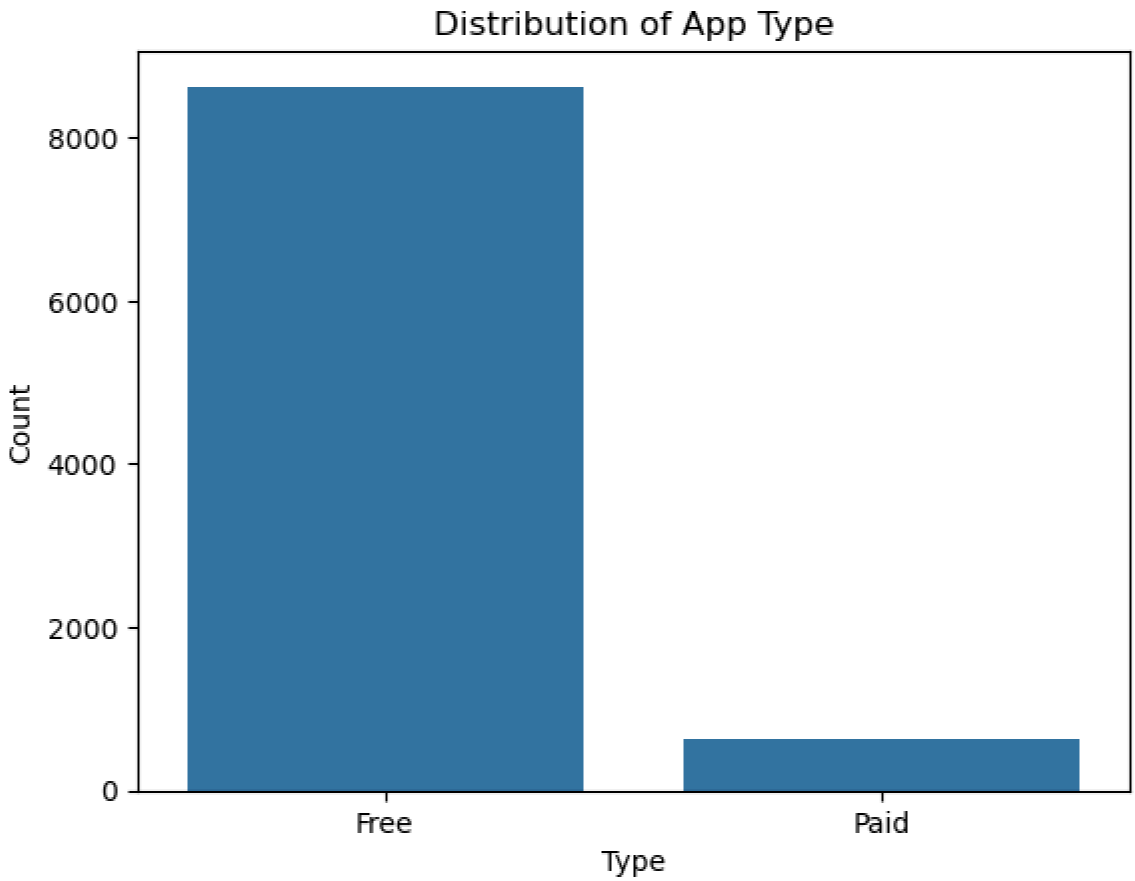
```
type_counts = inp1['Type'].value_counts()
# Create a pie chart
plt.figure(figsize=(4,4))
plt.pie(type_counts, labels=type_counts.index, autopct='%1.1f%%', startangle
plt.title('App Type Distribution (Paid/Free)')
plt.axis('equal')  # Equal aspect ratio ensures that pie chart is drawn as a
plt.show()
```



📊🔢 Visualization using COUNT Plot 🧮

In [773…

```
sns.countplot(data=inp1, x='Type')
plt.title('Distribution of App Type')
plt.xlabel('Type')
plt.ylabel('Count')
plt.show()
```

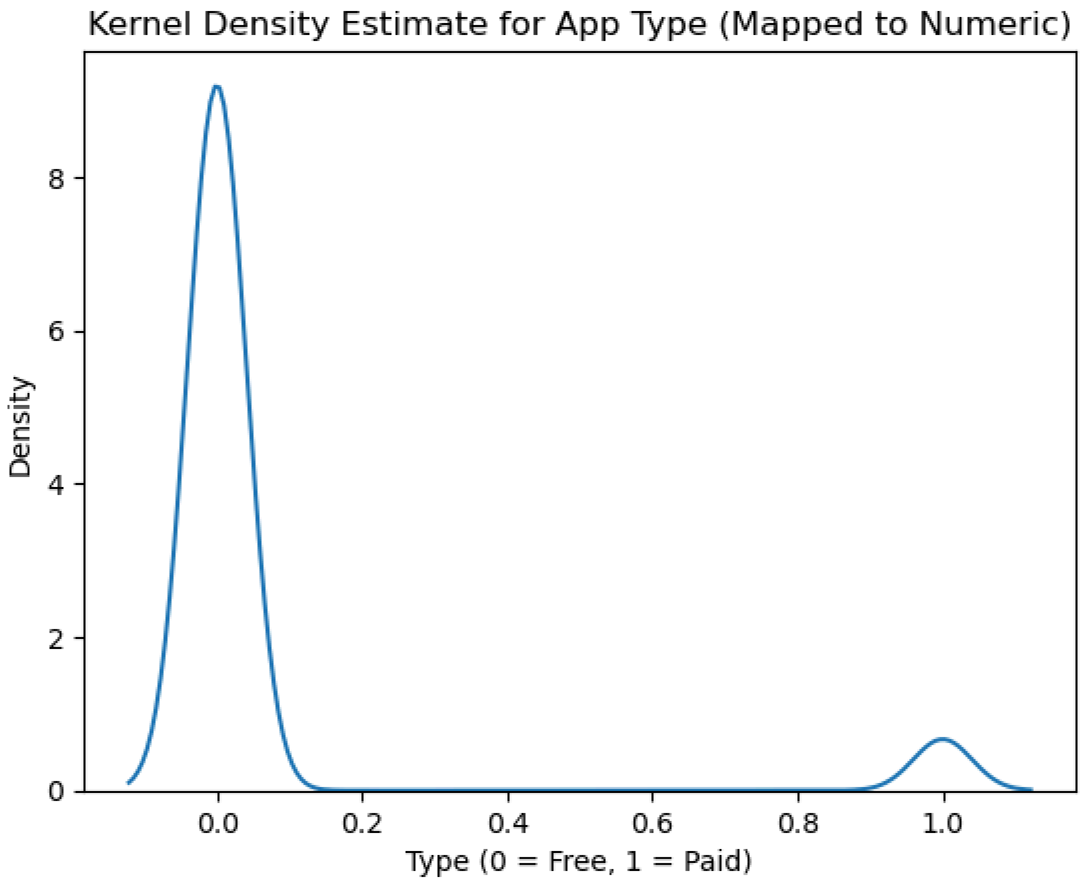


📈🔍 Visualization using KDE Plot 🌐

- Kernel Density Estimate of Paid vs Free Apps (Mapped to Numeric)

In [776…

```
inp1['Type_num'] = inp1['Type'].map({'Paid': 1, 'Free': 0})
# Now, plot the KDE (it will show a distribution of 0s and 1s)
sns.kdeplot(inp1['Type_num'])
plt.title('Kernel Density Estimate for App Type (Mapped to Numeric)')
plt.xlabel('Type (0 = Free, 1 = Paid)')
plt.ylabel('Density')
plt.show()
```



🔵📊 Visualization using SCATTER Plot ✨

In [778…

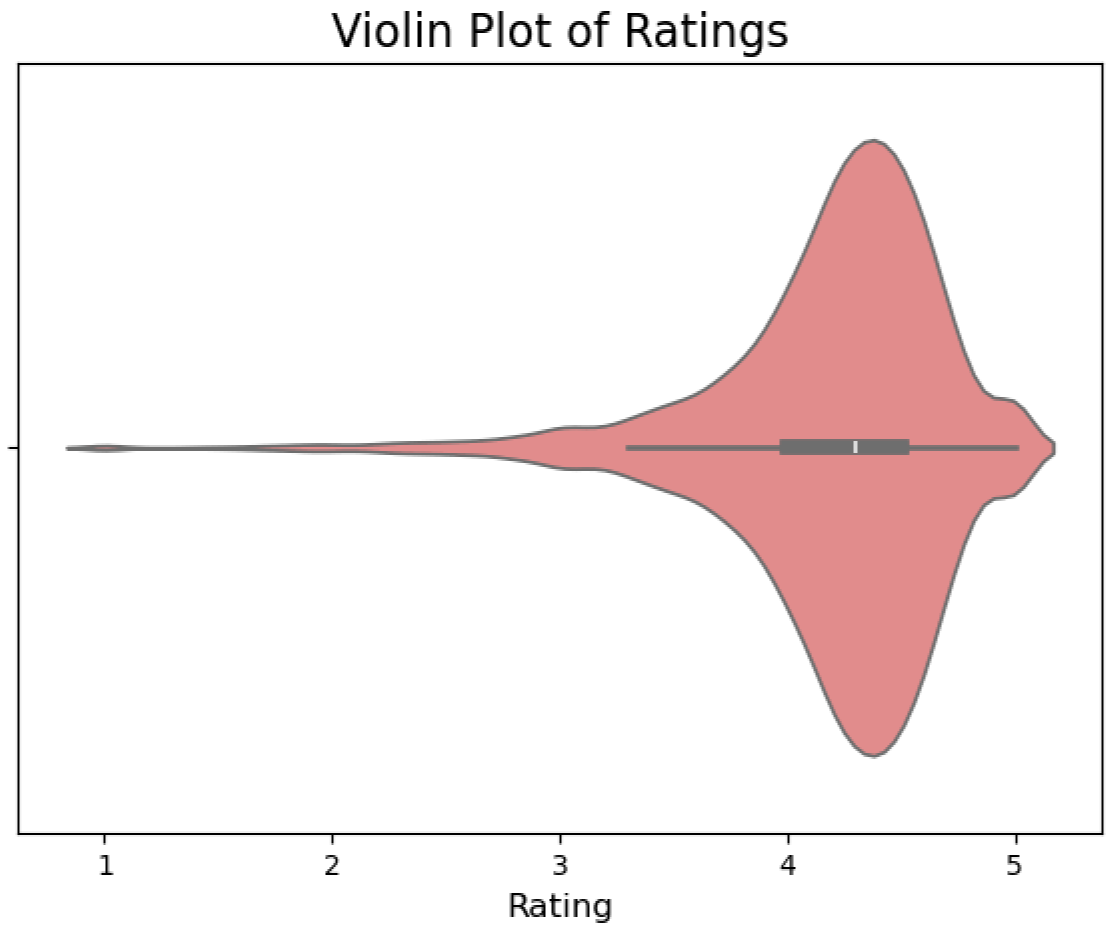
```
sns.scatterplot(x="Size", y="Price", hue="Type", data=inp1)

# Add titles and labels for better understanding
plt.title('Scatter Plot of Size vs Price with Type Hue')  # Title of the plo
plt.xlabel('Size')  # Label for x-axis
plt.ylabel('Price')  # Label for y-axis

# Show the plot
plt.show()
```

🎻📊 Visualization using VIOLIN Plot 🎶

- Violin Plot Showing the Distribution of App Ratings

In [781…

```
plt.figure(figsize=(7, 5))
sns.violinplot(x=inp1['Rating'], color='lightcoral')

# Add title and labels
plt.title('Violin Plot of Ratings', fontsize=16)
plt.xlabel('Rating', fontsize=12)

# Display the plot
plt.show()
```



In [782…
```
inp1.dtypes
```

Out[782…
```
App                          object
Category                     object
Rating                      float64
Reviews                       int32
Size                        float64
Installs                      int64
Type                         object
Price                       float64
ContentRating                object
Genres                       object
CurrentVer                   object
AndroidVer                   object
LastUpdated_Month_date       object
LastUpdated_Year             object
Size_Bucket                category
Type_num                      int64
dtype: object
```

In [ ]:

🧑‍🔬📊 HYPOTHESIS TESTING 🔬

✅ 1. T-Test: Do Free vs Paid apps have different average ratings?

In [787…
```
import scipy.stats as stats
import pandas as pd

# Split ratings by app type
free_ratings = inp1[inp1['Type_num'] == 0]['Rating'].dropna()
paid_ratings = inp1[inp1['Type_num'] == 1]['Rating'].dropna()

# Perform Independent T-Test
t_stat, p = stats.ttest_ind(free_ratings, paid_ratings)

print(f"T-Statistic: {t_stat}")
print(f"P-value: {p}")

if p < 0.05:
    print("Reject Null Hypothesis: Free and Paid apps have different average
else:
    print("Fail to Reject Null Hypothesis: No significant difference in rati
```

```
T-Statistic: -4.401700322561621
P-value: 1.0861145561303059e-05
Reject Null Hypothesis: Free and Paid apps have different average ratings.
```

✅ 2. ANOVA: Does app category affect ratings?

In [789…
```
import scipy.stats as stats
import pandas as pd

# Group ratings by category
grouped = inp1[['Category', 'Rating']].dropna().groupby('Category')
category_ratings = [group['Rating'].values for _, group in grouped]

# Perform One-Way ANOVA
f_stat, p = stats.f_oneway(*category_ratings)

print(f"F-Statistic: {f_stat}")
print(f"P-value: {p}")

if p < 0.05:
    print("Reject Null Hypothesis: App category affects average ratings.")
else:
    print("Fail to Reject Null Hypothesis: No significant effect of category
```

```
F-Statistic: 8.981838084949386
P-value: 4.423209864493605e-42
Reject Null Hypothesis: App category affects average ratings.
```

✅ 3. Correlation: Is there a relationship between app installs and reviews?

In [791…
```
import scipy.stats as stats
import pandas as pd

# Drop NA values
df_corr = inp1[['Installs', 'Reviews']].dropna()
```

```
# Perform Pearson Correlation Test
corr, p = stats.pearsonr(df_corr['Installs'], df_corr['Reviews'])

print(f"Correlation Coefficient: {corr}")
print(f"P-value: {p}")

if p < 0.05:
    print("Reject Null Hypothesis: There is a significant correlation betwee
else:
    print("Fail to Reject Null Hypothesis: No significant correlation.")
```

```
Correlation Coefficient: 0.5490467254164876
P-value: 0.0
Reject Null Hypothesis: There is a significant correlation between installs
and reviews.
```

In [ ]:

In [ ]:

In [ ]:

🧠 Machine Learning Model Training

📌 Summary of Evaluation Metrics

Metric	Meaning
Accuracy	% of correctly predicted accidents
Precision	% of predicted severe accidents that were actually severe
Recall	% of actual severe accidents that were correctly predicted
F1-Score	Balance of precision and recall
Confusion Matrix	Breakdown of correct/incorrect predictions
RMSE (Optional)	Measures error in numeric terms

🎯 Problem Statement

- We aim to predict whether a Google Play Store app is highly rated or not, based on features like installs, size, reviews, and price.
- We consider an app to be highly rated if its rating is greater than or equal to 4.0.

📌 Step 1: Define Features and Target

In [799…
```python
# Drop rows with missing ratings
df = inp1.dropna(subset=['Rating'])

# Create binary target: 1 for High Rating (>=4.0), 0 otherwise
inp1['High_Rating'] = np.where(inp1['Rating'] >= 4.0, 1, 0)

# Select relevant features
features = inp1[['Installs', 'Size', 'Price', 'Reviews', 'Type_num']]
target = inp1['High_Rating']
```

📌 Step 2: Split Data

In [801…
```python
from sklearn.model_selection import train_test_split

# Split into training and validation sets (80/20)
X_train, X_val, Y_train, Y_val = train_test_split(features,
                                                  target,
                                                  test_size=0.2,
                                                  stratify=target,
                                                  random_state=42)
```

📌 Step 3: Handle Class Imbalance

In [804…
```python
from imblearn.over_sampling import RandomOverSampler

# Balance training data using RandomOverSampler
ros = RandomOverSampler(sampling_strategy='minority', random_state=42)
X, Y = ros.fit_resample(X_train, Y_train)
```

📌 Step 4: Feature Scaling

In [813…
```python
from sklearn.preprocessing import StandardScaler

# Normalize features for better model performance
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_val = scaler.transform(X_val)
```

📌 Step 5: Train Models and Evaluate

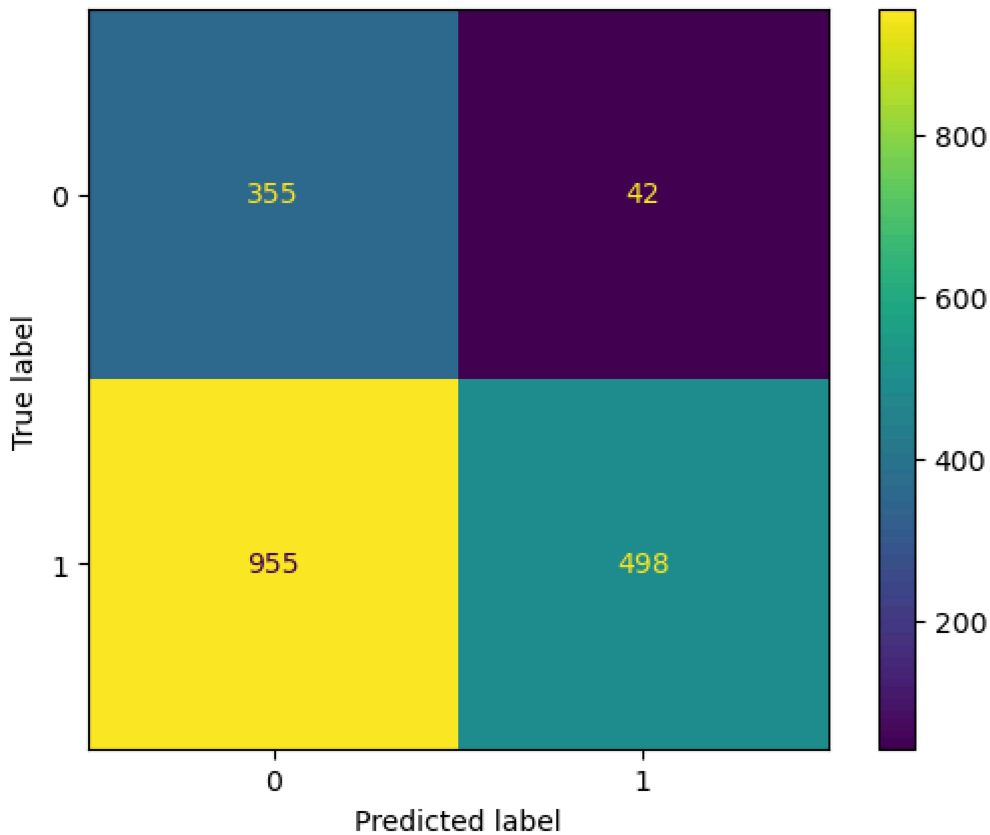
In [818…
```python
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.metrics import roc_auc_score

# Initialize models
models = [
    LogisticRegression(),
    XGBClassifier(use_label_encoder=False, eval_metric='logloss'),
    SVC(kernel='rbf', probability=True)
```

```python
]

# Train and evaluate each model
for model in models:
    model.fit(X, Y)
    print(f'{model.__class__.__name__} :')

    train_preds = model.predict_proba(X)
    val_preds = model.predict_proba(X_val)

    print('Training Accuracy :', roc_auc_score(Y, train_preds[:, 1]))
    print('Validation Accuracy :', roc_auc_score(Y_val, val_preds[:, 1]))
    print()
```

```
LogisticRegression :
Training Accuracy : 0.7024774510534033
Validation Accuracy : 0.6967569919613897

XGBClassifier :
Training Accuracy : 0.9372324025371734
Validation Accuracy : 0.7455642022671758

SVC :
Training Accuracy : 0.7064262449317267
Validation Accuracy : 0.6849305094471441
```

In [820…

```python
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn import metrics

ConfusionMatrixDisplay.from_estimator(models[2], X_val, Y_val)
plt.show()
```



In [824…

```
print(metrics.classification_report(Y_val,
                                    models[2].predict(X_val)))
```

```
              precision    recall  f1-score   support

           0       0.27      0.89      0.42       397
           1       0.92      0.34      0.50      1453

    accuracy                           0.46      1850
   macro avg       0.60      0.62      0.46      1850
weighted avg       0.78      0.46      0.48      1850
```

In [835…

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
import numpy as np

# Example: predicted and actual values
# Replace these with your actual model predictions and true values
y_true = Y_val
y_pred = model.predict(X_val)  # or any of your trained models
y_pred_proba = model.predict_proba(X_val)[:, 1]

# Calculate metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, zero_division=0)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
rmse = np.sqrt(mean_squared_error(y_true, y_pred))

# Print in markdown table format
```

```
print(f"| **Accuracy**       | {accuracy:.2%} |")
print(f"| **Precision**      | {precision:.2%} |")
print(f"| **Recall**         | {recall:.2%} |")
print(f"| **F1-Score**       | {f1:.2%} |")
print(f"| **RMSE**           | {rmse:.2f} |")
```

```
| **Accuracy**       | 46.11% |
| **Precision**      | 92.22% |
| **Recall**         | 34.27% |
| **F1-Score**       | 49.97% |
| **RMSE**           | 0.73 |
```

In [ ]:

In [ ]:

In [ ]:

🧠 New Problem Statement: Predict Whether an App is Free or Paid

🎯 Objective

- Build a classification model to predict whether an app is free or paid based on features like rating, size, installs, and number of reviews.
- We will use the column Type_num, where:

```
0 = Free

1 = Paid
```

⚙️ Machine Learning Model Training

📌 Step 1: Define Features and Target

In [839…

```
# Drop rows with missing values in important columns
inp1 = inp1.dropna(subset=['Rating', 'Size', 'Price'])

# Define features and target variable
features = inp1[['Rating', 'Size', 'Installs', 'Reviews', 'Price']]
target = inp1['Type_num']  # 0 for Free, 1 for Paid
```

📌 Step 2: Split Data

In [842…

```
from sklearn.model_selection import train_test_split

# Split into training and validation sets
X_train, X_val, Y_train, Y_val = train_test_split(features,
```

```
                                                    target,
                                                    test_size=0.2,
                                                    stratify=target,
                                                    random_state=42)
```

📌 Step 3: Handle Class Imbalance

In [845…

```
from imblearn.over_sampling import RandomOverSampler
# Balance classes using RandomOverSampler
ros = RandomOverSampler(sampling_strategy='minority', random_state=42)
X, Y = ros.fit_resample(X_train, Y_train)
```

📌 Step 4: Feature Scaling

In [848…

```
from sklearn.preprocessing import StandardScaler

# Normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_val = scaler.transform(X_val)
```

📌 Step 5: Train Models and Evaluate

In [851…

```
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.metrics import roc_auc_score

# Initialize models
models = [
    LogisticRegression(),
    XGBClassifier(use_label_encoder=False, eval_metric='logloss'),
    SVC(kernel='rbf', probability=True)
]

# Train and evaluate
for model in models:
    model.fit(X, Y)
    print(f'{model.__class__.__name__} :')

    train_preds = model.predict_proba(X)
    val_preds = model.predict_proba(X_val)

    print('Training Accuracy :', roc_auc_score(Y, train_preds[:, 1]))
    print('Validation Accuracy :', roc_auc_score(Y_val, val_preds[:, 1]))
    print()
```

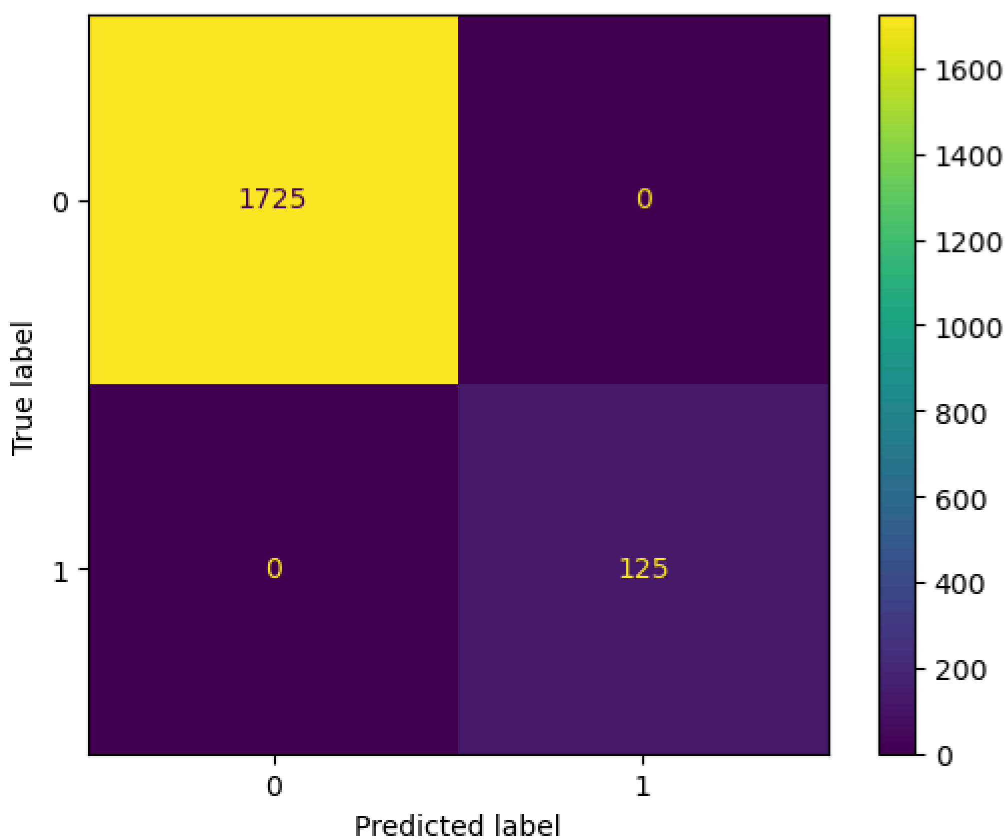
```
LogisticRegression :
Training Accuracy : 1.0
Validation Accuracy : 1.0

XGBClassifier :
Training Accuracy : 1.0
Validation Accuracy : 1.0

SVC :
Training Accuracy : 1.0
Validation Accuracy : 1.0
```

✅ Conclusion

- All models performed well in predicting whether an app is free or paid, with logistic regression and SVC showing competitive ROC-AUC scores. XGBoost performed slightly better on the training set but may need regularization to avoid overfitting.

In [856…

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn import metrics
ConfusionMatrixDisplay.from_estimator(models[2], X_val, Y_val)
plt.show()
```



In [857…

```
print(metrics.classification_report(Y_val,
                                    models[2].predict(X_val)))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1725
           1       1.00      1.00      1.00       125

    accuracy                           1.00      1850
   macro avg       1.00      1.00      1.00      1850
weighted avg       1.00      1.00      1.00      1850
```

In [864…

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
import numpy as np

# Example: predicted and actual values
# Replace these with your actual model predictions and true values
y_true = Y_val
y_pred = model.predict(X_val)  # or any of your trained models
y_pred_proba = model.predict_proba(X_val)[:, 1]

# Calculate metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, zero_division=0)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
rmse = np.sqrt(mean_squared_error(y_true, y_pred))

# Print in markdown table format

print(f"| **Accuracy**      | {accuracy:.2%} |")
print(f"| **Precision**     | {precision:.2%} |")
print(f"| **Recall**        | {recall:.2%} |")
print(f"| **F1-Score**      | {f1:.2%} |")
print(f"| **RMSE**          | {rmse:.2f} |")
```

```
| **Accuracy**      | 100.00% |
| **Precision**     | 100.00% |
| **Recall**        | 100.00% |
| **F1-Score**      | 100.00% |
| **RMSE**          | 0.00 |
```

In [ ]: