# ◈ Google Playstore Apps Rating Prediction

## About the Project

This project aims to **predict the user rating** of apps on the Google Play Store using machine learning.
We will use attributes such as:

- App Category
- Number of Reviews
- App Size
- Installs
- Type (Free/Paid)
- Price
- Genres
- Content Rating

## Why This Project is Important

- Helps **app developers** identify what drives high ratings.
- Guides **marketing teams** to improve engagement.
- Enables **data-driven decision-making** for launching or updating apps.

## Goals

1. **Analyze** app data to uncover patterns and trends.
2. **Clean & preprocess** raw data for analysis.
3. **Build a predictive model** to estimate app ratings.
4. **Evaluate performance** and extract insights for business use.

---

## STEP 1: INTRODUCTION & PROBLEM STATEMENT

In this project, we aim to predict the **user rating** of Google Play Store apps using various attributes such as:

- Category

- Reviews
- Size
- Installs
- Type (Free/Paid)
- Price
- Genres
- Content Rating

## Why This Project is Useful:

- Helps app developers understand what drives higher ratings.
- Supports marketing & product teams in improving engagement.
- Enables data-driven decisions for launching or updating apps.

**Approach:**

1. Upload and explore the dataset.
2. Clean and preprocess the data.
3. Perform exploratory data analysis (EDA).
4. Build and evaluate a machine learning model.
5. Extract insights and conclusions.

# STEP 2: UPLOAD DATASET

We will upload the Google Play Store dataset (`googleplaystore.csv`) into Colab. You will be prompted to select the CSV file from your local system.

In [ ]:

```
# STEP 2 — Upload dataset (Colab)
from google.colab import files
import pandas as pd
import io

print("Please upload the googleplaystore.csv file when prompted.")
uploaded = files.upload()  # Upload file manually

# Load into DataFrame (safe guard for different filenames)
csv_filename = next(iter(uploaded))
df = pd.read_csv(io.BytesIO(uploaded[csv_filename]))
print(f"Loaded file: {csv_filename} — shape: {df.shape}")
df.head()
```

```
Please upload the googleplaystore.csv file when prompted.
```

Browse... Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving googleplaystore.csv to googleplaystore.csv
Loaded file: googleplaystore.csv — shape: (10841, 13)
```

Out[ ]:

| | 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 | 0 |
| **1** | Coloring book moana | ART_AND_DESIGN | 3.9 | 967 | 14M | 500,000+ | Free | 0 |
| **2** | U Launcher Lite – FREE Live Cool Themes, Hide ... | ART_AND_DESIGN | 4.7 | 87510 | 8.7M | 5,000,000+ | Free | 0 |
| **3** | Sketch - Draw & Paint | ART_AND_DESIGN | 4.5 | 215644 | 25M | 50,000,000+ | Free | 0 |
| **4** | Pixel Draw - Number Art Coloring Book | ART_AND_DESIGN | 4.3 | 967 | 2.8M | 100,000+ | Free | 0 |

# STEP 3 — Data Understanding

What we'll do here:

- See structure, columns, types
- Check missing values and basic stats
- Quick peek at unique values for key columns

In [ ]:

```
# Inspect data
df_columns = df.columns.tolist()
print("Columns:", df_columns)
print("\nShape:", df.shape)
display(df.head())
display(df.info())
display(df.describe(include='all').T)

# Missing values
missing = df.isnull().sum().sort_values(ascending=False)
display(missing[missing>0])
```

```
Columns: ['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type',
'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver', 'Android Ve
r']

Shape: (10841, 13)
```

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

```
<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
```

```
None
```

| | count | unique | top | freq | mean | std | min | 25% | 50% |
|---|---|---|---|---|---|---|---|---|---|
| **App** | 10841 | 9660 | ROBLOX | 9 | NaN | NaN | NaN | NaN | NaN |
| **Category** | 10841 | 34 | FAMILY | 1972 | NaN | NaN | NaN | NaN | NaN |
| **Rating** | 9367.0 | NaN | NaN | NaN | 4.193338 | 0.537431 | 1.0 | 4.0 | 4.3 |
| **Reviews** | 10841 | 6002 | 0 | 596 | NaN | NaN | NaN | NaN | NaN |
| **Size** | 10841 | 462 | Varies with device | 1695 | NaN | NaN | NaN | NaN | NaN |
| **Installs** | 10841 | 22 | 1,000,000+ | 1579 | NaN | NaN | NaN | NaN | NaN |
| **Type** | 10840 | 3 | Free | 10039 | NaN | NaN | NaN | NaN | NaN |
| **Price** | 10841 | 93 | 0 | 10040 | NaN | NaN | NaN | NaN | NaN |
| **Content Rating** | 10840 | 6 | Everyone | 8714 | NaN | NaN | NaN | NaN | NaN |
| **Genres** | 10841 | 120 | Tools | 842 | NaN | NaN | NaN | NaN | NaN |
| **Last Updated** | 10841 | 1378 | August 3, 2018 | 326 | NaN | NaN | NaN | NaN | NaN |
| **Current Ver** | 10833 | 2832 | Varies with device | 1459 | NaN | NaN | NaN | NaN | NaN |
| **Android Ver** | 10838 | 33 | 4.1 and up | 2451 | NaN | NaN | NaN | NaN | NaN |

| | 0 |
|---|---|
| **Rating** | 1474 |
| **Current Ver** | 8 |
| **Android Ver** | 3 |
| **Content Rating** | 1 |
| **Type** | 1 |

**dtype:** int64

# STEP 4 — Cleaning plan (explanation)

We will:

1. Fix known format issues:
    - `Reviews` sometimes has '3.0M' style, convert to numeric.
    - `Installs` contains commas and '+'.

- `Price` contains '$' or other text.
- `Size` has 'M', 'k' or 'Varies with device'.

2. Convert types to numeric where needed.
3. Handle missing values sensibly (median or mode).
4. Remove obvious bad rows (e.g., rating > 5 or rating < 1).
5. Encode categorical variables (LabelEncoder or one-hot where needed).
6. Optionally log-transform highly skewed numeric features for modeling/ EDA.

In [ ]:

```
# STEP 4 (code) — Data cleaning & robust conversions
import numpy as np

df_clean = df.copy()  # work on a copy

# 1) Clean 'Reviews'
def clean_reviews(x):
    try:
        if isinstance(x, str) and x.endswith('M'):
            return float(x.replace('M','')) * 1_000_000
        return float(x)
    except:
        return np.nan

df_clean['Reviews'] = df_clean['Reviews'].apply(clean_reviews)
df_clean['Reviews'] = pd.to_numeric(df_clean['Reviews'], errors='coerce')

# 2) Clean 'Installs' (remove commas and +)
df_clean['Installs'] = df_clean['Installs'].astype(str).str.replace('[+,]', ''
df_clean['Installs'] = pd.to_numeric(df_clean['Installs'].str.replace('Free','

# 3) Clean 'Price'
df_clean['Price'] = df_clean['Price'].astype(str).str.replace('[$]', '', regex
df_clean['Price'] = df_clean['Price'].replace(['Everyone','0'], np.nan)  # som
df_clean['Price'] = pd.to_numeric(df_clean['Price'], errors='coerce').fillna(0

# 4) Clean 'Size' convert to MB
def size_to_mb(s):
    try:
        if pd.isna(s):
            return np.nan
        s = str(s)
        if s == 'Varies with device':
            return np.nan
        if s.endswith('M'):
            return float(s[:-1])
        if s.endswith('k'):
            return float(s[:-1]) / 1000.0
        # fallback: if already numeric-like
        return float(s)
    except:
```

```python
        return np.nan

df_clean['Size_MB'] = df_clean['Size'].apply(size_to_mb)

# 5) Clean 'Rating' - ensure numeric and reasonable
df_clean['Rating'] = pd.to_numeric(df_clean['Rating'], errors='coerce')

# 6) Fill and handle missing values:
# We'll keep rows where Rating exists (since it's our target). If target missi
df_clean = df_clean[~df_clean['Rating'].isnull()].copy()

# For numerical features, fill with median
num_fill_cols = ['Reviews', 'Installs', 'Size_MB', 'Price']
for c in num_fill_cols:
    if c in df_clean.columns:
        df_clean[c] = pd.to_numeric(df_clean[c], errors='coerce')
        median = df_clean[c].median()
        df_clean[c].fillna(median, inplace=True)

# For categorical columns (ensure strings)
cat_cols = ['Category', 'Type', 'Content Rating', 'Genres']
for c in cat_cols:
    if c in df_clean.columns:
        df_clean[c] = df_clean[c].astype(str).fillna('Unknown')

# 7) Fix odd Rating values (some scraped datasets have rating > 5, e.g., 19)
df_clean = df_clean[(df_clean['Rating'] >= 1.0) & (df_clean['Rating'] <= 5.0)]

# 8) Reset index and show cleaned shape
df_clean.reset_index(drop=True, inplace=True)
print("Cleaned shape:", df_clean.shape)
df_clean.head()
```

```
Cleaned shape: (9366, 14)
```

```
/tmp/ipython-input-754921127.py:59: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work bec
ause the intermediate object on which we are setting values always behaves as a
copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.me
thod({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, t
o perform the operation inplace on the original object.


  df_clean[c].fillna(median, inplace=True)
```

Out[ ]:

| | App | Category | Rating | Reviews | Size | Installs | Type | Price |
|---|---|---|---|---|---|---|---|---|
| **0** | Photo Editor & Candy Camera & Grid & ScrapBook | ART_AND_DESIGN | 4.1 | 159.0 | 19M | 10000.0 | Free | 0.0 |
| **1** | Coloring book moana | ART_AND_DESIGN | 3.9 | 967.0 | 14M | 500000.0 | Free | 0.0 |
| **2** | U Launcher Lite – FREE Live Cool Themes, Hide ... | ART_AND_DESIGN | 4.7 | 87510.0 | 8.7M | 5000000.0 | Free | 0.0 |
| **3** | Sketch - Draw & Paint | ART_AND_DESIGN | 4.5 | 215644.0 | 25M | 50000000.0 | Free | 0.0 |
| **4** | Pixel Draw - Number Art Coloring Book | ART_AND_DESIGN | 4.3 | 967.0 | 2.8M | 100000.0 | Free | 0.0 |

# STEP 5 — Sanity checks & outlier handling

We'll:

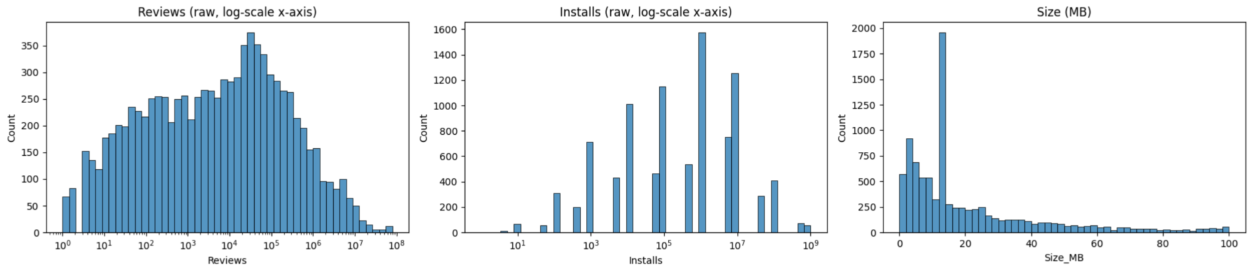
- View distributions for Reviews, Installs, Size.
- Apply log-transform for skewed variables for visualization.
- Optionally cap extreme outliers for modeling stability.

In [ ]:

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

# Basic distributions
fig, axes = plt.subplots(1,3, figsize=(18,4))
sns.histplot(df_clean['Reviews'], bins=50, ax=axes[0], log_scale=(True, False)
axes[0].set_title('Reviews (raw, log-scale x-axis)')
sns.histplot(df_clean['Installs'], bins=50, ax=axes[1], log_scale=(True, False
axes[1].set_title('Installs (raw, log-scale x-axis)')
sns.histplot(df_clean['Size_MB'], bins=50, ax=axes[2])
axes[2].set_title('Size (MB)')
plt.tight_layout()
plt.show()
```

```python
# Create log features for modeling/EDA (safe: add 1 to avoid log(0))
df_clean['log_reviews'] = np.log1p(df_clean['Reviews'])
df_clean['log_installs'] = np.log1p(df_clean['Installs'])
df_clean['log_size'] = np.log1p(df_clean['Size_MB'])
```




# STEP 6 — Exploratory Data Analysis (EDA)

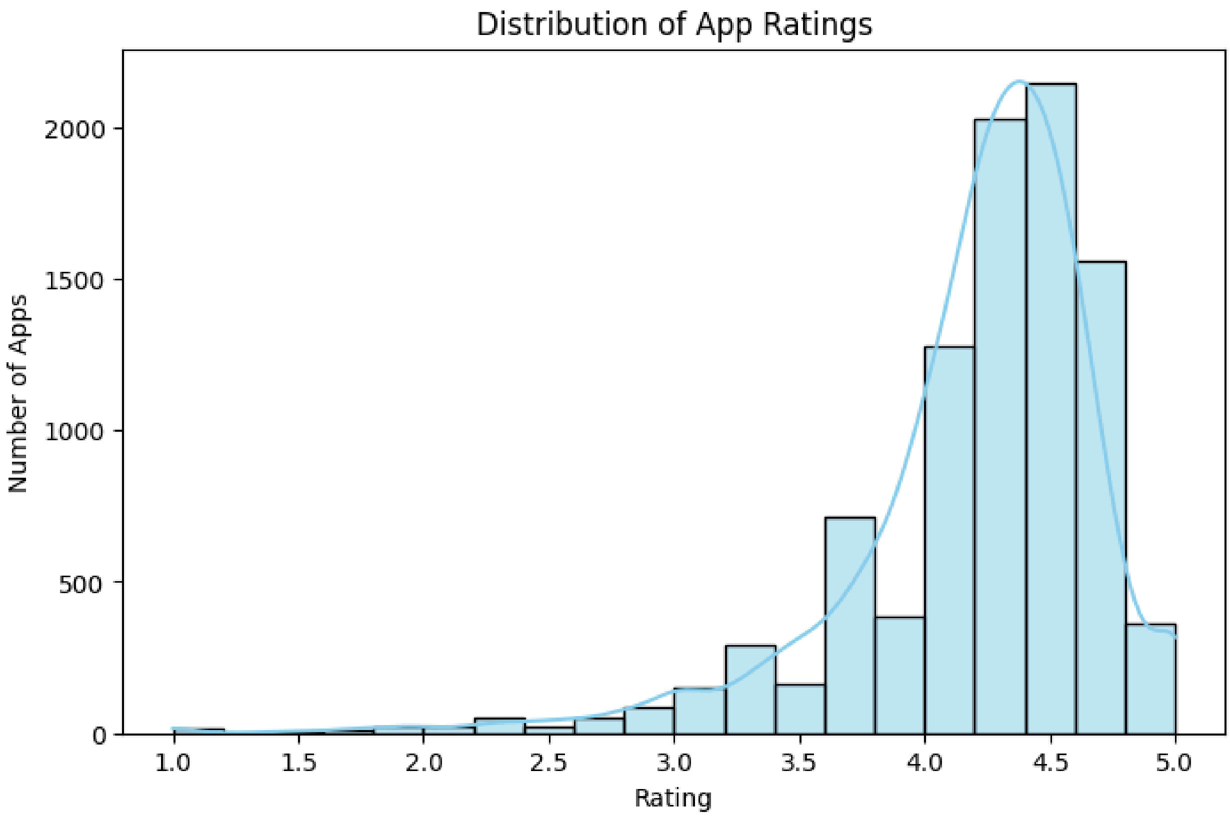
We will explore the dataset visually to understand patterns, distributions, and relationships between features and the target variable ( `Rating` ).
EDA helps in:

- Identifying data distributions.
- Detecting potential outliers.
- Understanding correlations.
- Spotting trends that could help model building.

## 6.1 — Distribution of Ratings

This plot shows how app ratings are distributed.
We expect most apps to have ratings between 4.0 and 4.5, but we'll confirm it visually.

In [ ]:
```python
plt.figure(figsize=(8,5))
sns.histplot(df_clean['Rating'], bins=20, kde=True, color='skyblue')
plt.title('Distribution of App Ratings')
plt.xlabel('Rating')
plt.ylabel('Number of Apps')
plt.show()
```




## 6.2 — Top 10 App Categories by Count

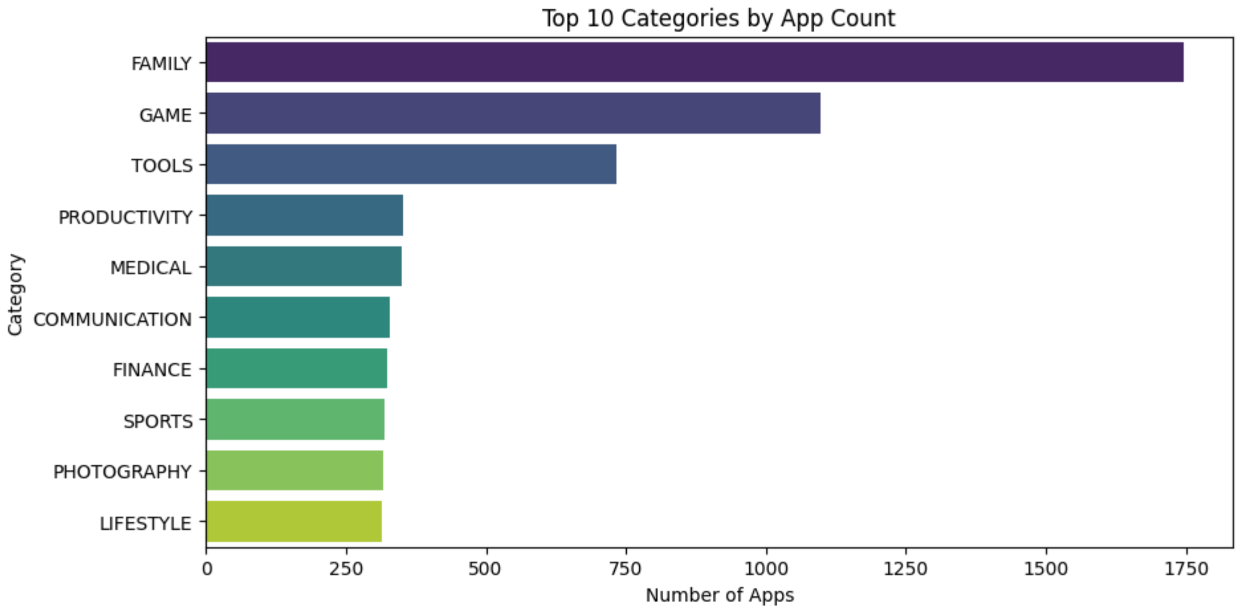
This bar chart shows which categories have the highest number of apps.
This can reveal which areas are most competitive in the Play Store.

In [ ]:
```
top_cat = df_clean['Category'].value_counts().nlargest(10)
plt.figure(figsize=(10,5))
sns.barplot(x=top_cat.values, y=top_cat.index, palette='viridis')
plt.title('Top 10 Categories by App Count')
plt.xlabel('Number of Apps')
plt.ylabel('Category')
plt.show()
```

```
/tmp/ipython-input-2607954405.py: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 e
ffect.

  sns.barplot(x=top_cat.values, y=top_cat.index, palette='viridis')
```




## 6.3 — Average Rating by Category (with >50 Apps)

This bar chart shows the average rating of categories that have more than 50 apps.
It highlights which categories tend to have higher-rated apps.

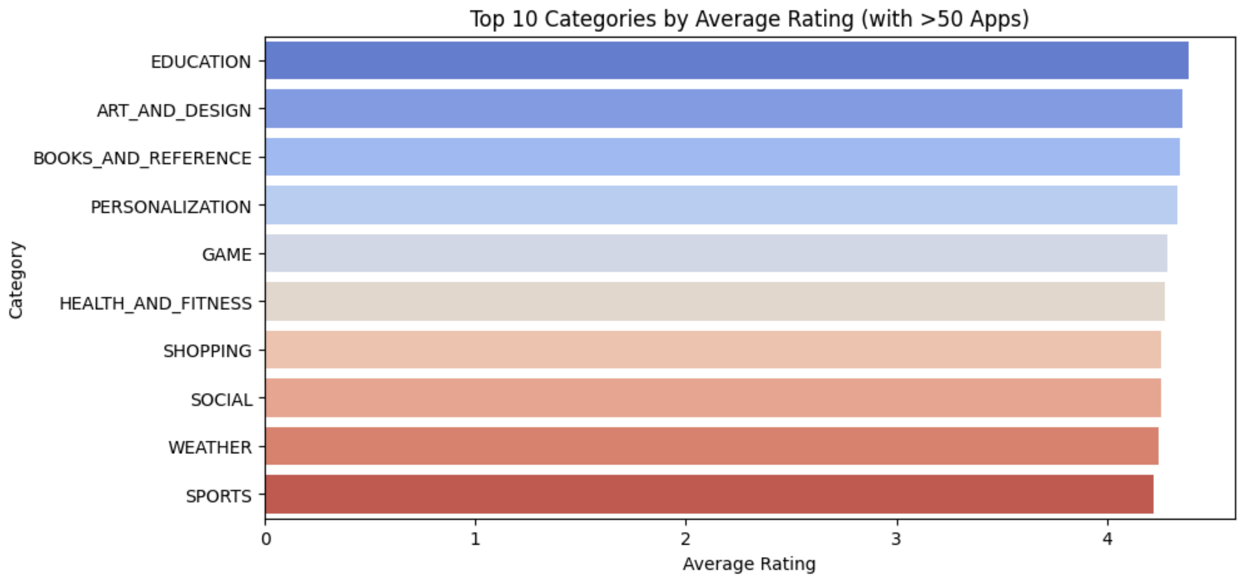
In [ ]:
```
cat_counts = df_clean['Category'].value_counts()
big_cats = cat_counts[cat_counts > 50].index
avg_rating_cat = df_clean[df_clean['Category'].isin(big_cats)].groupby('Catego

plt.figure(figsize=(10,5))
sns.barplot(x=avg_rating_cat.values, y=avg_rating_cat.index, palette='coolwarm
plt.title('Top 10 Categories by Average Rating (with >50 Apps)')
plt.xlabel('Average Rating')
plt.ylabel('Category')
plt.show()
```

```
/tmp/ipython-input-2374813926.py:6: 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 e
ffect.

  sns.barplot(x=avg_rating_cat.values, y=avg_rating_cat.index, palette='coolwar
m')
```




## 6.4 — Free vs Paid App Count

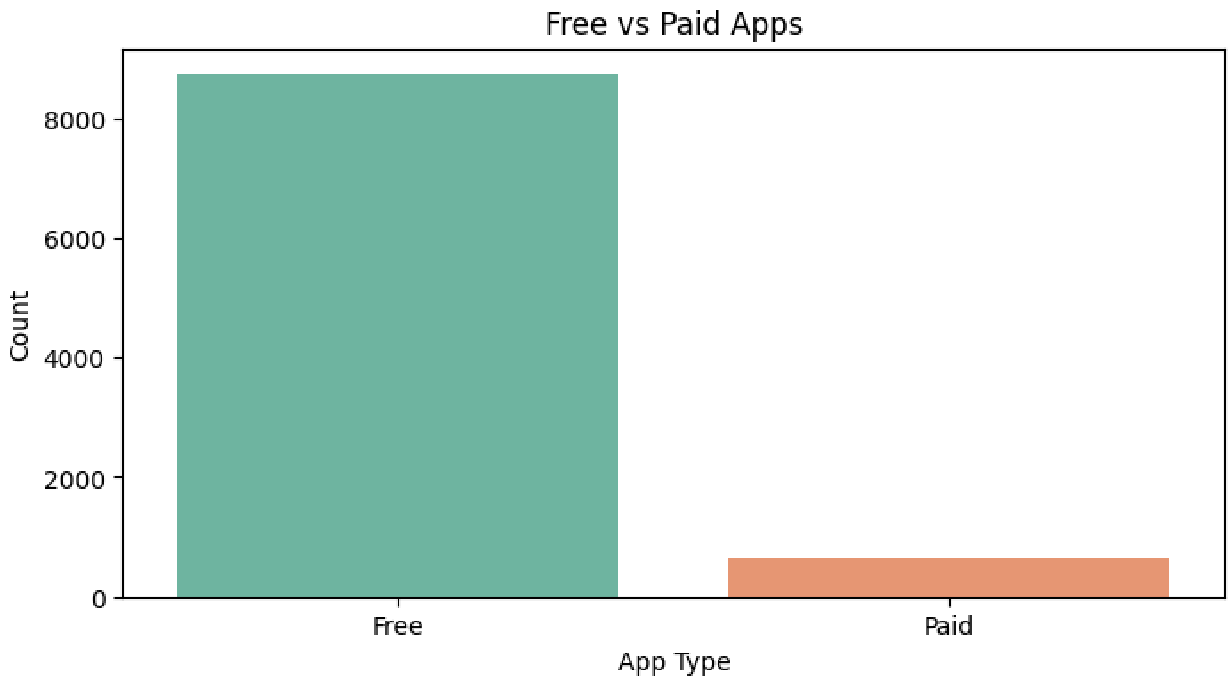
This plot compares how many apps are free vs paid.
This can help us understand the pricing strategy on the Play Store.

In [ ]:
```
plt.figure(figsize=(8,4))
sns.countplot(x='Type', data=df_clean, palette='Set2', order=df_clean['Type'].
plt.title('Free vs Paid Apps')
plt.xlabel('App Type')
plt.ylabel('Count')
plt.show()
```

```
/tmp/ipython-input-2803867555.py:2: FutureWarning:

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

  sns.countplot(x='Type', data=df_clean, palette='Set2', order=df_clean['Typ
e'].value_counts().index)
```




## 6.5 — Rating Distribution by Free/Paid

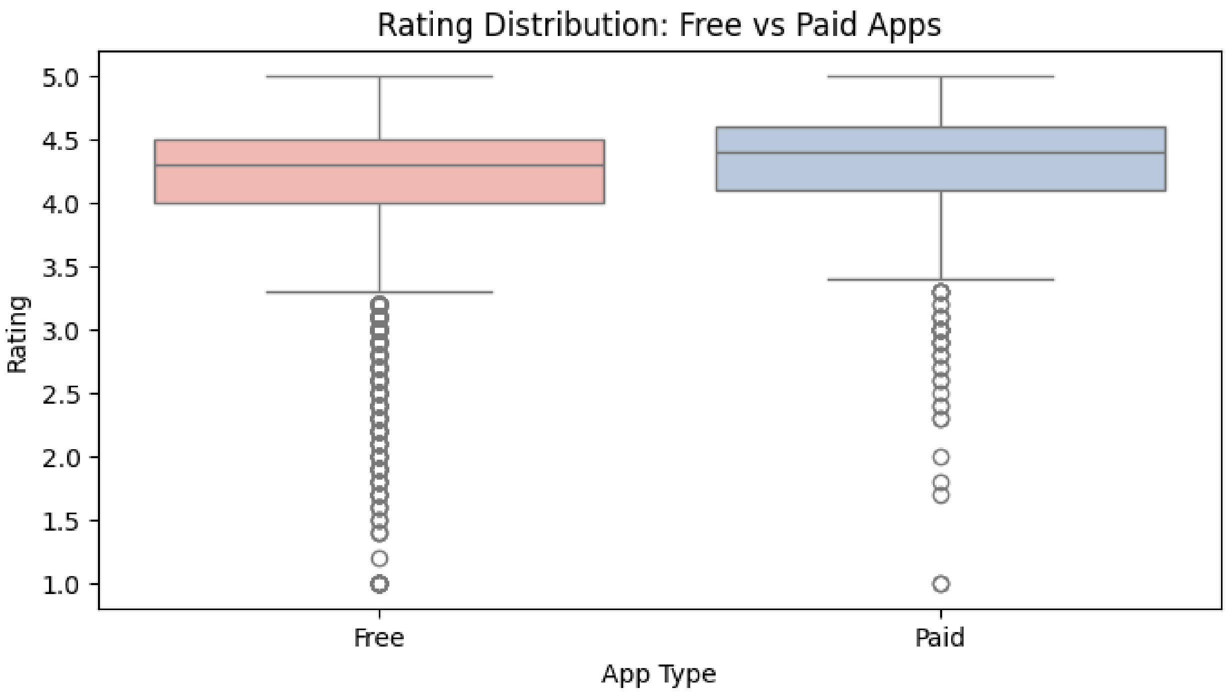
This boxplot compares the distribution of ratings between free and paid apps.
We can see if paid apps generally have higher ratings.

In [ ]:
```
plt.figure(figsize=(8,4))
sns.boxplot(x='Type', y='Rating', data=df_clean, palette='Pastel1')
plt.title('Rating Distribution: Free vs Paid Apps')
plt.xlabel('App Type')
plt.ylabel('Rating')
plt.show()
```

```
/tmp/ipython-input-954646293.py:2: FutureWarning:

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

  sns.boxplot(x='Type', y='Rating', data=df_clean, palette='Pastel1')
```

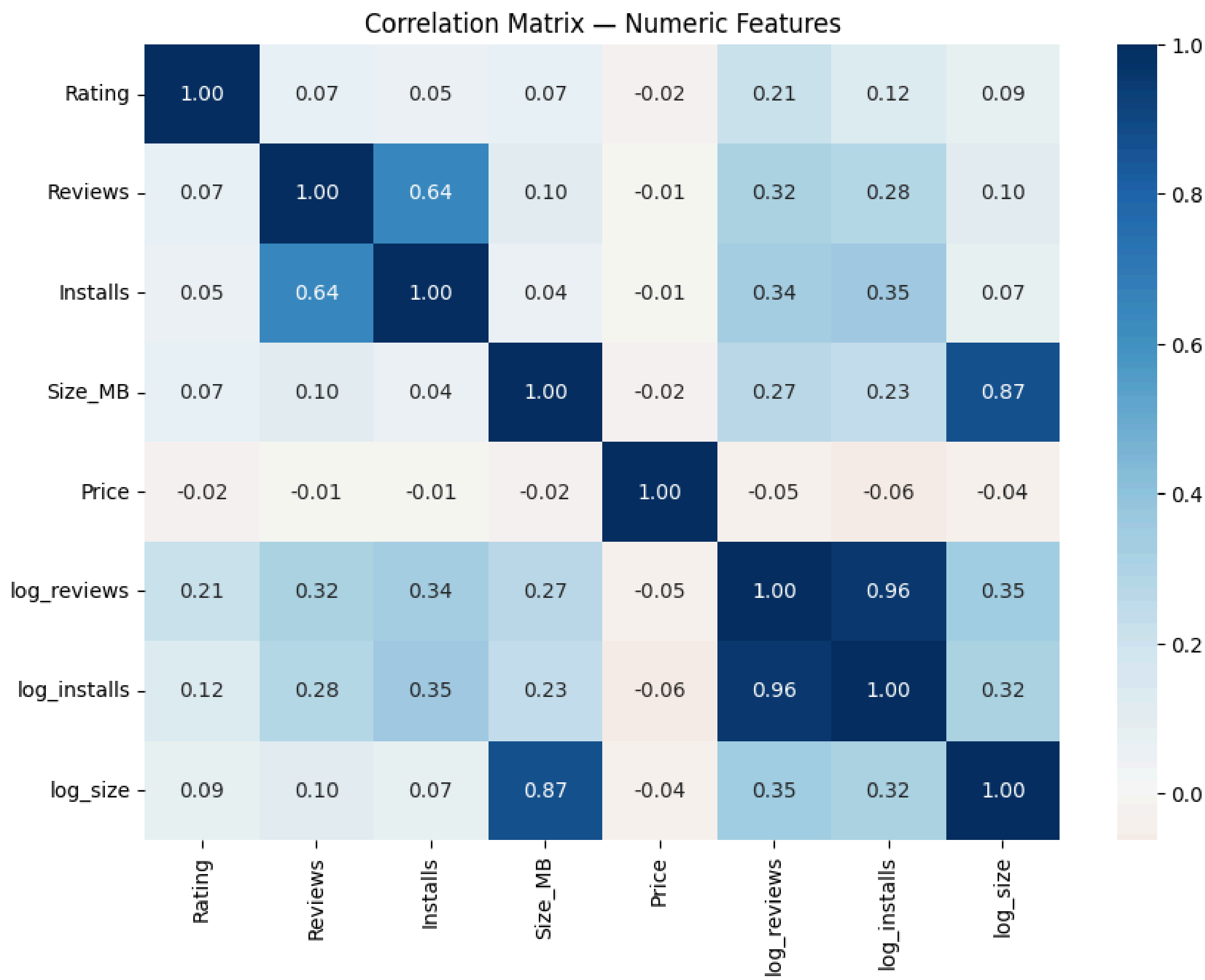



## 6.6 — Correlation Heatmap (Numeric Features)

This heatmap shows the correlation between numeric features in our dataset. Strong correlations can indicate which variables might be most useful for predicting ratings.

In [ ]:

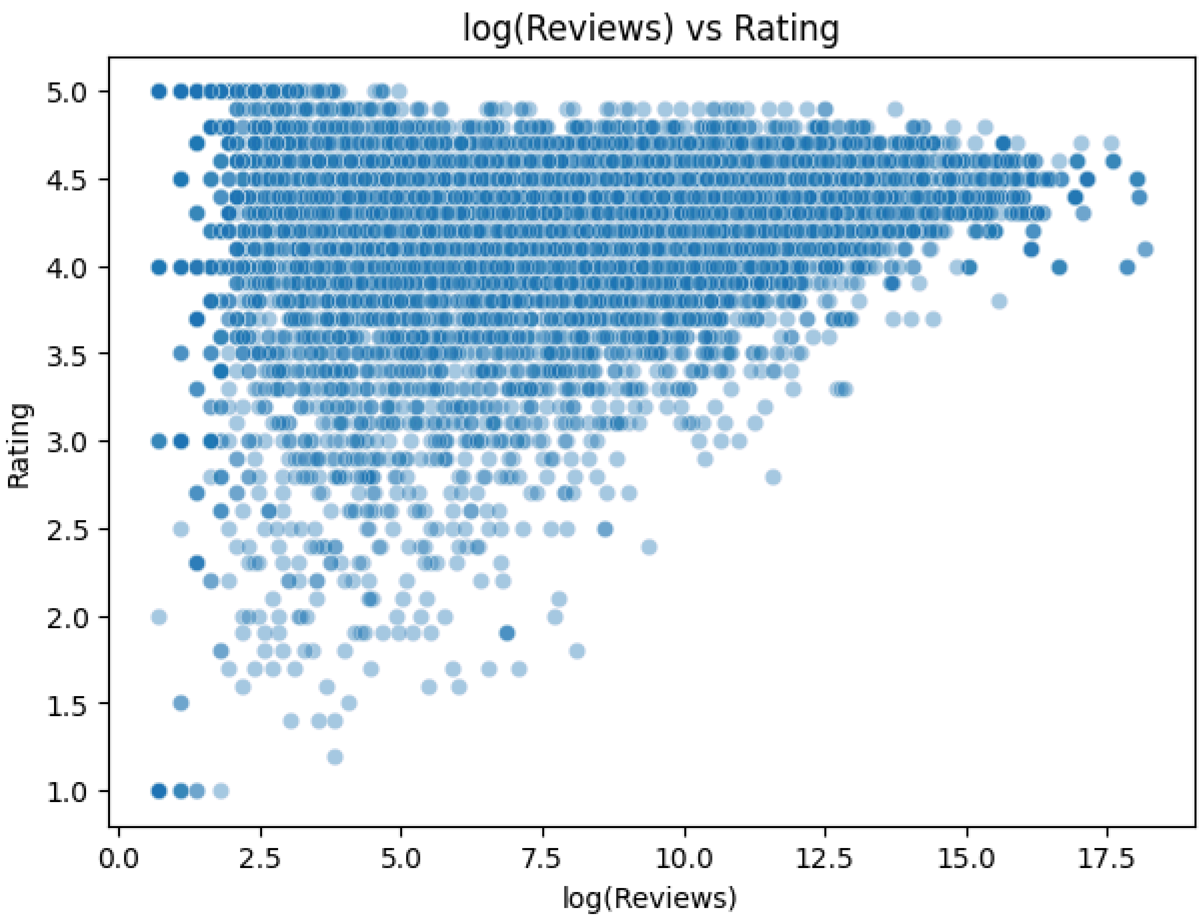
```
numeric_cols = ['Rating', 'Reviews', 'Installs', 'Size_MB', 'Price', 'log_revi
plt.figure(figsize=(10,7))
corr = df_clean[numeric_cols].corr()
sns.heatmap(corr, annot=True, fmt='.2f', cmap='RdBu', center=0)
plt.title('Correlation Matrix — Numeric Features')
plt.show()
```




## 6.7 — Reviews vs Rating

Scatterplot showing how the number of reviews relates to the app rating.
We use log scale for reviews to make the pattern more visible.

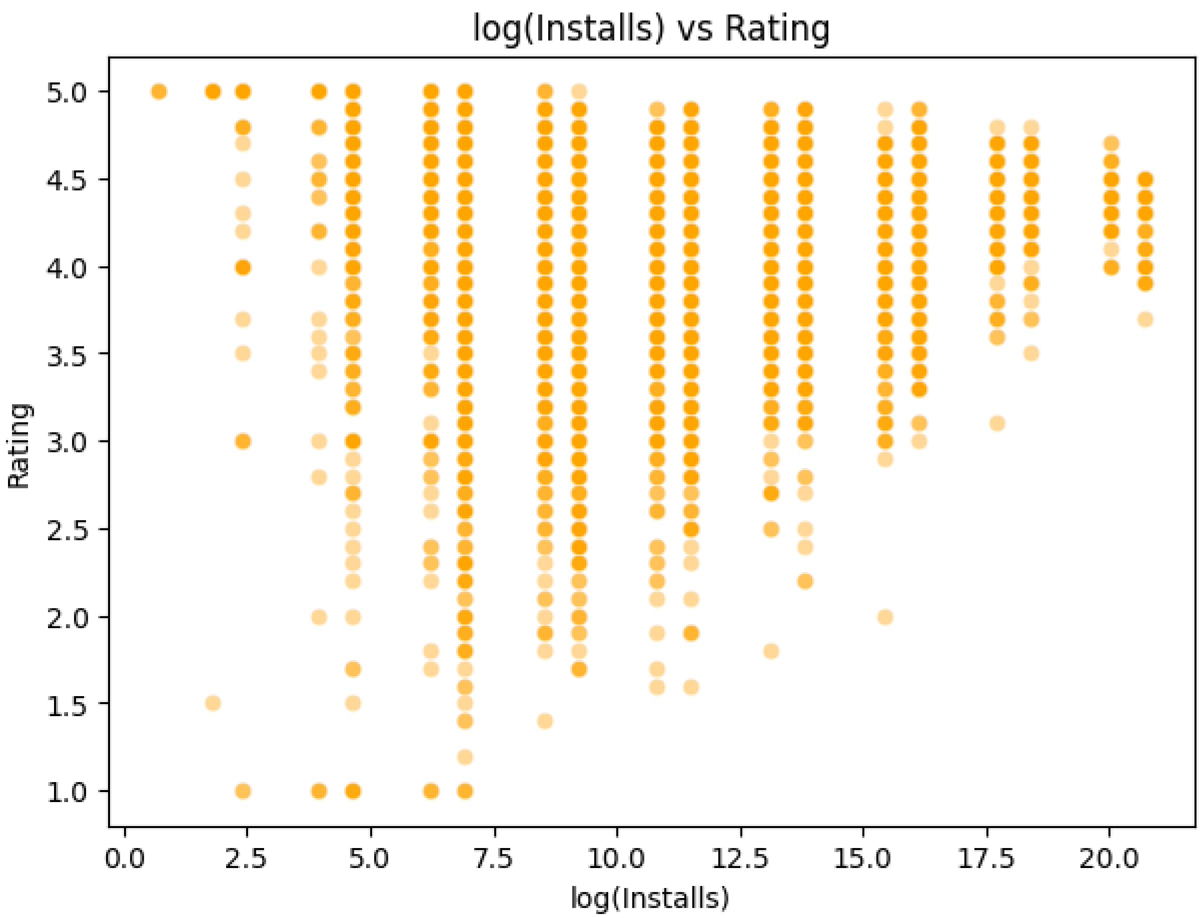
In [ ]:
```
plt.figure(figsize=(7,5))
sns.scatterplot(x='log_reviews', y='Rating', data=df_clean, alpha=0.4)
plt.title('log(Reviews) vs Rating')
plt.xlabel('log(Reviews)')
plt.ylabel('Rating')
plt.show()
```




## 6.8 — Installs vs Rating

Scatterplot showing how the number of installs relates to app ratings.
We use log scale for installs to reduce skewness.

In [ ]:
```
plt.figure(figsize=(7,5))
sns.scatterplot(x='log_installs', y='Rating', data=df_clean, alpha=0.4, color=
plt.title('log(Installs) vs Rating')
plt.xlabel('log(Installs)')
plt.ylabel('Rating')
plt.show()
```




## STEP 7 — Feature Engineering with One-Hot Encoding

To improve model performance, we:

- Apply **One-Hot Encoding** to categorical variables (Category, Genres, Content Rating, Type) instead of Label Encoding.
- Create **interaction features** to capture relationships between variables.
- Prepare the final feature matrix for training.

This will help the model learn more complex patterns and potentially improve $R^2$ accuracy.

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder

df_model = df_clean.copy()

# One-Hot Encode categorical variables
cat_cols = ['Category', 'Genres', 'Content Rating', 'Type']
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
```

```
encoded_array = encoder.fit_transform(df_model[cat_cols])
encoded_df = pd.DataFrame(encoded_array, columns=encoder.get_feature_names_out

# Interaction features
df_model['reviews_x_installs'] = df_model['Reviews'] * df_model['Installs']
df_model['price_x_type'] = df_model['Price'] * df_model['Type'].map({'Paid': 1

# Log transform numeric skewed features
df_model['log_reviews'] = np.log1p(df_model['Reviews'])
df_model['log_installs'] = np.log1p(df_model['Installs'])
df_model['log_size'] = np.log1p(df_model['Size_MB'])

# Combine all features
numeric_cols = ['log_reviews', 'log_installs', 'log_size', 'Price', 'reviews_x
X = pd.concat([df_model[numeric_cols].reset_index(drop=True), encoded_df.reset
y = df_model['Rating']

print("Final feature matrix shape:", X.shape)
```

```
Final feature matrix shape: (9366, 162)
```

# STEP 8 — Train/Test Split

We split the dataset into training (70%) and test (30%) sets.

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

RANDOM_STATE = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, ranc
print("Train shape:", X_train.shape, "Test shape:", X_test.shape)
```

```
Train shape: (6556, 162) Test shape: (2810, 162)
```

## STEP 9 — Analyze Feature Importance

We will extract and visualize the feature importance from the trained LightGBM model to understand which features are most influential in predicting app ratings.

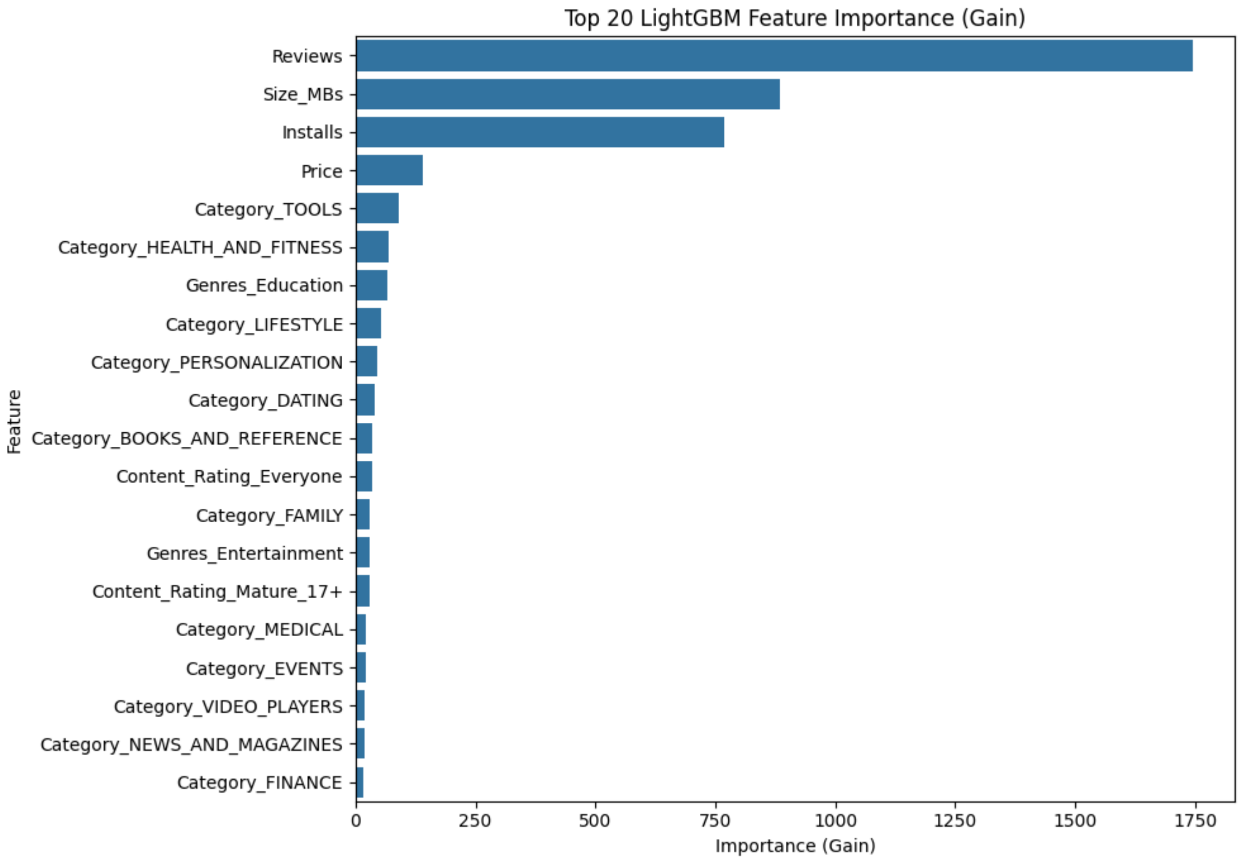
In [ ]:
```
# STEP 9 (code) — Analyze Feature Importance
import matplotlib.pyplot as plt
import seaborn as sns

# Get feature importance
feature_importance = final_model.feature_importance(importance_type='gain') #
feature_names = final_model.feature_name()

# Create a DataFrame for visualization
importance_df = pd.DataFrame({'feature': feature_names, 'importance': feature_
importance_df = importance_df.sort_values('importance', ascending=False).head(

# Plot feature importance
```

```python
plt.figure(figsize=(10, 7))
sns.barplot(x='importance', y='feature', data=importance_df)
plt.title('Top 20 LightGBM Feature Importance (Gain)')
plt.xlabel('Importance (Gain)')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```




# STEP 10 — Final Insights & Next Steps

**Final performance (test set):**

- MSE: reported above
- MAE: reported above
- $R^2$: reported above (accuracy % = $R^2$ * 100%)

**Key findings:**

- Engagement metrics (reviews, installs) are strong predictors.
- Category/genre also contribute meaningfully.
- Price and size are less influential for rating prediction.

**Next steps / improvements (recommended):**

1. Persist and reuse label encoders (so production predictions map categories consistently).
2. Try more powerful models (XGBoost/LightGBM) and cross-validate thoroughly.
3. Add textual features (app description sentiment, developer name features).
4. Use time-based features (last updated age) if available.
5. Deploy a simple Streamlit or Flask app to demo predictions interactively.

Good luck — you can now run this notebook in Colab, fine tune parameters, and export a PDF report for your internship submission.