In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
import shap
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

In [5]:
```
#Load & Clean Dataset
df = pd.read_csv("googleplaystore.csv")

# Drop duplicates
df.drop_duplicates(inplace=True)

# Handle missing values
df.dropna(subset=['Rating'], inplace=True)
df['Reviews'] = pd.to_numeric(df['Reviews'], errors='coerce')
df['Reviews'].fillna(0, inplace=True)

# Clean 'Installs'
df['Installs'] = df['Installs'].str.replace('[+,]', '', regex=True)
df = df[df['Installs'].str.isnumeric()]
df['Installs'] = df['Installs'].astype(float)

# Clean 'Price'
df['Price'] = df['Price'].str.replace('$', '', regex=True).astype(float)

# Clean 'Size'
def size_to_mb(size):
    if 'M' in str(size):
        return float(str(size).replace('M',''))
    elif 'k' in str(size):
        return float(str(size).replace('k',''))/1024
    elif size == 'Varies with device':
        return np.nan
    else:
        return np.nan

df['Size'] = df['Size'].apply(size_to_mb)
df['Size'].fillna(df['Size'].median(), inplace=True)
```

In [6]:
```
#Feature Engineering
le = LabelEncoder()
for col in ['Category', 'Type', 'Content Rating', 'Genres']:
    df[col] = le.fit_transform(df[col].astype(str))

# Log-transform Reviews (safe even if 0)
df['Log_Reviews'] = np.log1p(df['Reviews'])

# Price Buckets — handle duplicate edges
try:
    df['Price_Bucket'] = pd.qcut(df['Price'], 4, labels=False, duplicates='drop')
```

```python
except ValueError:
    # Fallback: manually define bins if qcut still fails
    bins = [-0.01, 0, 1, 5, 50, df['Price'].max()]
    df['Price_Bucket'] = pd.cut(df['Price'], bins=bins, labels=False)

# Installs Buckets – same issue might occur, handle similarly
try:
    df['Install_Bucket'] = pd.qcut(df['Installs'], 5, labels=False, duplicates='drop')
except ValueError:
    # Fallback if too many identical install values
    bins = [0, 1000, 10000, 100000, 1000000, df['Installs'].max()]
    df['Install_Bucket'] = pd.cut(df['Installs'], bins=bins, labels=False)
```

In [7]:

```python
#Split Data
X = df[['Category','Reviews','Size','Installs','Type','Price','Genres',
        'Content Rating','Log_Reviews','Price_Bucket','Install_Bucket']]
y = df['Rating']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

In [ ]:

```python
#Models & Evaluation
def evaluate_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    return {"MSE": mse, "R2": r2, "Model": model}

# Random Forest
rf = RandomForestRegressor(random_state=42, n_estimators=200)
rf_results = evaluate_model(rf, X_train, y_train, X_test, y_test)

# Gradient Boosting
gb = GradientBoostingRegressor(random_state=42, n_estimators=200)
gb_results = evaluate_model(gb, X_train, y_train, X_test, y_test)

# XGBoost
xgb_model = xgb.XGBRegressor(n_estimators=300, learning_rate=0.1, random_state=42)
xgb_results = evaluate_model(xgb_model, X_train, y_train, X_test, y_test)

print("Random Forest:", rf_results)
print("Gradient Boosting:", gb_results)
print("XGBoost:", xgb_results)
```

```
Random Forest: {'MSE': 0.2409688522430425, 'R2': 0.092508417727442, 'Model': RandomFores
tRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=200, n_jobs=None, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)}
Gradient Boosting: {'MSE': 0.23508378853844625, 'R2': 0.11467163809117698, 'Model': Grad
ientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
```

```
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=200,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=42, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)}
XGBoost: {'MSE': 0.2509560051951473, 'R2': 0.05489668014999605, 'Model': XGBRegressor(ba
se_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.1, max_delta_step=0,
             max_depth=6, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=300, n_jobs=12,
             num_parallel_tree=1, objective='reg:squarederror',
             predictor='auto', random_state=42, reg_alpha=0, reg_lambda=1,
             scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)}
```

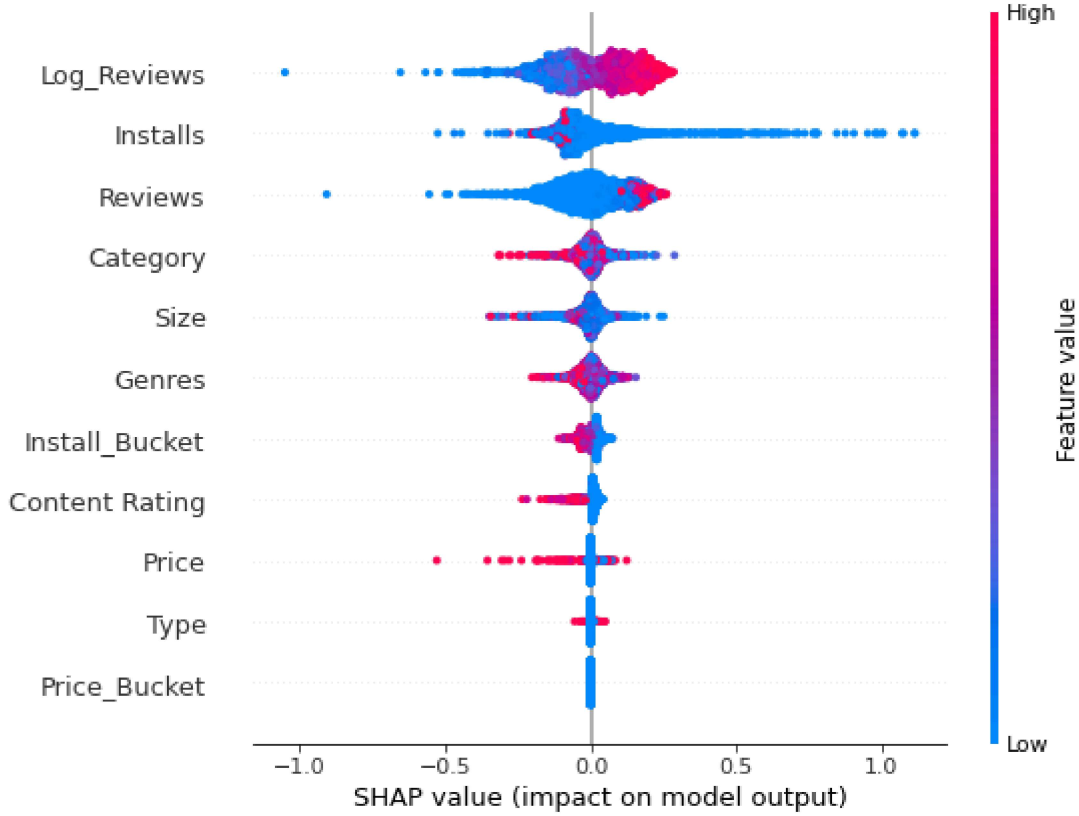
In [9]:

```
#Hyperparameter Tuning (GridSearch for RF)
param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}
rf_model = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(
    estimator=rf_model,
    param_grid=param_grid,
    scoring='r2',
    cv=5,
    n_jobs=-1,
    verbose=1
)
grid_search.fit(X_train, y_train)
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Cross-Validation R² Score:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:   13.3s
[Parallel(n_jobs=-1)]: Done 135 out of 135 | elapsed:   54.4s finished
Best Hyperparameters: {'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 500}
Best Cross-Validation R² Score: 0.14987331068239648
```

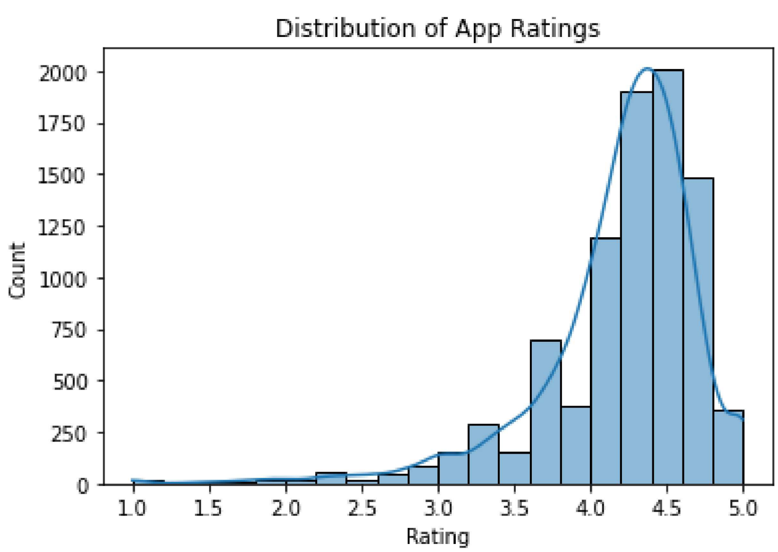
In [10]:

```
#Model Explainability with SHAP
explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test, feature_names=X.columns)
```



In [13]:

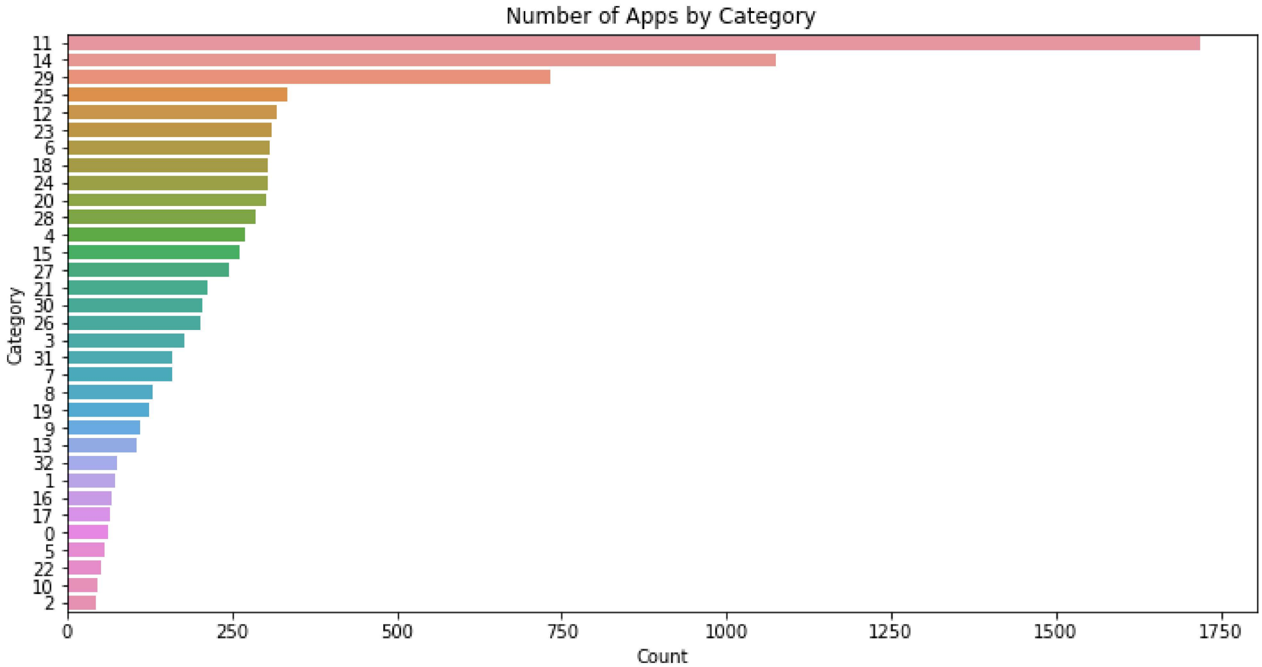
```
# Histogram of Ratings
sns.histplot(df['Rating'], bins=20, kde=True)
plt.title("Distribution of App Ratings")
plt.xlabel("Rating")
plt.ylabel("Count")
plt.show()
```



In [14]:

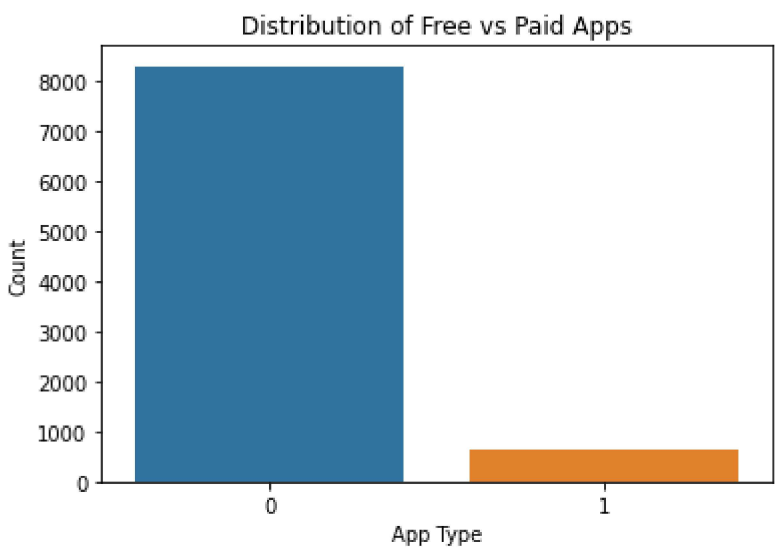
```
# Countplot of Categories
plt.figure(figsize=(12,6))
sns.countplot(y=df['Category'], order=df['Category'].value_counts().index)
plt.title("Number of Apps by Category")
plt.xlabel("Count")
```

```
plt.ylabel("Category")
plt.show()
```



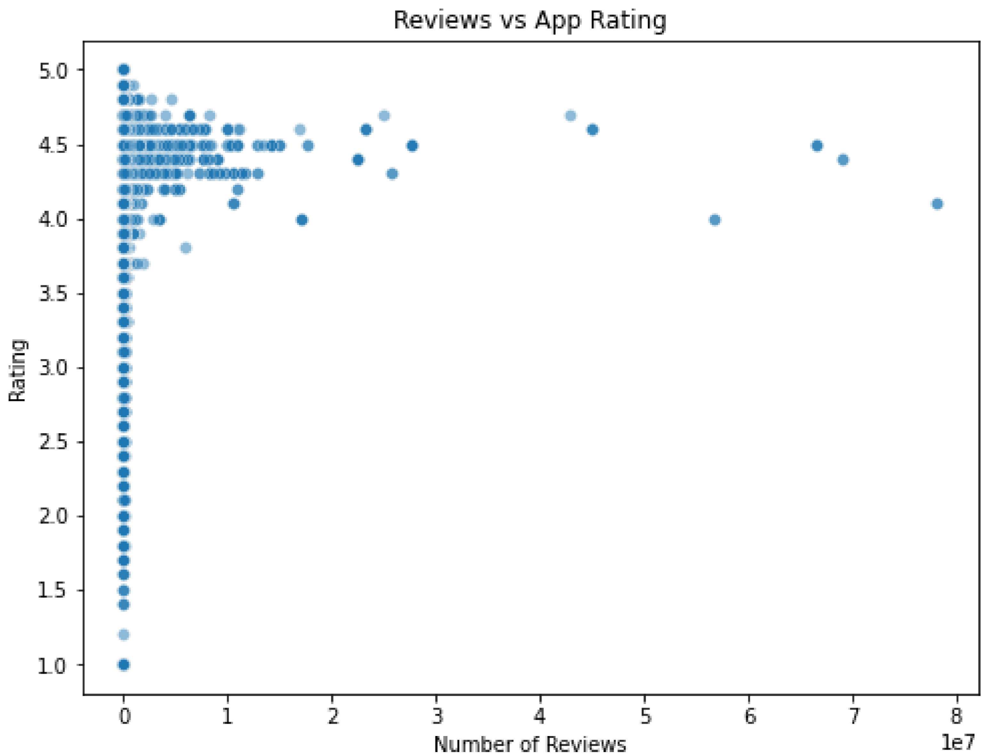
In [15]:

```
# Countplot of Free vs Paid
sns.countplot(x='Type', data=df)
plt.title("Distribution of Free vs Paid Apps")
plt.xlabel("App Type")
plt.ylabel("Count")
plt.show()
```

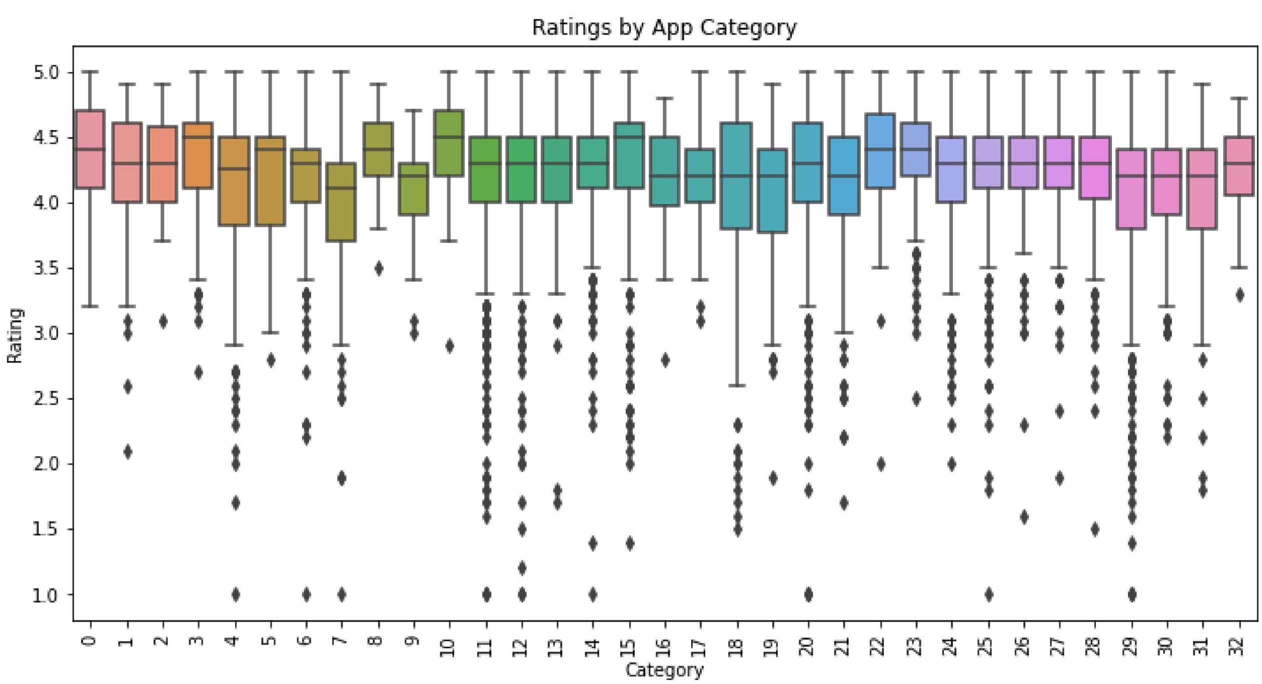


In [ ]:

```
# Scatter Plot: Reviews vs Rating
plt.figure(figsize=(8,6))
sns.scatterplot(x='Reviews', y='Rating', data=df, alpha=0.5)
plt.title("Reviews vs App Rating")
plt.xlabel("Number of Reviews")
plt.ylabel("Rating")
plt.show()
```



In [12]:
```
# Rating distribution by category
plt.figure(figsize=(12,6))
sns.boxplot(x="Category", y="Rating", data=df)
plt.xticks(rotation=90)
plt.title("Ratings by App Category")
plt.show()
```



In [17]:
```
# Correlation heatmap
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(), annot=False, cmap="coolwarm")
```

```
plt.title("Feature Correlation Heatmap")
plt.show()
```