EDA on Googleplaystore

What is EDA?

EDA (Exploratory Data Analysis) is a crucial step in the data analysis process that involves summarizing the main characteristics of a dataset, often using visual methods. The goal is to explore the data before making any assumptions or building models.

Key Objectives of EDA:

1. **Understand the Structure of Data**: EDA helps in understanding the shape, size, and overall structure of the dataset.
2. **Detect Missing or Outlier Values**: It helps identify any missing values, errors, or outliers that could affect the analysis.
3. **Identify Patterns and Relationships**: EDA looks for trends, correlations, or any hidden patterns in the data.
4. **Validate Assumptions**: This step checks whether the data meets the assumptions for further statistical analysis or modeling.

Techniques in EDA:

- **Summary Statistics**: Functions like `.describe()`, `.info()` to get mean, median, standard deviation, etc.
- **Visualizations**:
  - **Histograms** and **Density Plots** for distribution of data.
  - **Box Plots** to identify outliers.
  - **Scatter Plots** to understand relationships between variables.
  - **Correlation Heatmaps** to analyze relationships between multiple variables.

EDA is often the first step when working with a dataset, enabling you to make better-informed decisions on how to proceed with data cleaning, transformation, and modeling.

Find Dataset : https://www.kaggle.com/datasets/lava18/google-play-store-apps

Information About My Dataset

The dataset has 10,841 rows and 13 columns. This indicates a large dataset with detailed information on a wide range of apps from the Google Play Store. The columns are:

1. **App**: Name of the app.
2. **Category**: Category of the app (e.g., Games, Productivity).
3. **Rating**: Rating given to the app, usually out of 5.
4. **Reviews**: Number of reviews the app has received.
5. **Size**: Size of the app.
6. **Installs**: Number of installs (often represented in ranges, e.g., 1,000+).
7. **Type**: Paid or Free app.
8. **Price**: Price of the app if it is paid.
9. **Content Rating**: Audience suitability (e.g., Everyone, Teen, Mature).
10. **Genres**: Specific genre(s) of the app.
11. **Last Updated**: Date when the app was last updated.
12. **Current Ver**: Current version of the app.
13. **Android Ver**: Minimum Android version required ton the app. !

Importing packages and dataset

In [ ]:
```
#import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

In [7]:
```
gpa_df = pd.read_csv("googleplaystore.csv")
gpa_df.head()
```

Out[7]:

	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

- The head() function in Python, particularly with pandas, is used to display the first 5 rows of a DataFrame

In [8]:
```
gpa_df.tail()
```

Out[8]:

	App	Category	Rating	Reviews	Size	Installs
10836	Sya9a Maroc - FR	FAMILY	4.5	38	53M	5,000+
10837	Fr. Mike Schmitz Audio Teachings	FAMILY	5.0	4	3.6M	100+
10838	Parkinson Exercices FR	MEDICAL	NaN	3	9.5M	1,000+
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+

- The tail() function is used to display the last 5 rows of the DataFrame

Data Preprocessing and Cleaning

- Data preparation and cleaning are crucial steps in data analysis to ensure that the dataset is accurate, consistent, and ready for analysis.

In [9]:
```
gpa_df.shape
```

Out[9]:
```
(10841, 13)
```

- If the output of gpa_df.shape is (10841, 13), it means there are 10,841 rows and 13 columns in the DataFrame.

In [10]:
```
gpa_df.columns
```

Out[10]:
```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type',
       'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver',
       'Android Ver'],
      dtype='object')
```

- The code gpa_df.columns returns an index object that lists all the column names of the DataFrame gpa_df.

- Random sample of rows or columns from a DataFrame

In [12]:
```
gpa_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
```

The gpa_df.info() function provides a summary of the DataFrame's structure, including:

- **Number of entries (rows)**: Total number of data points in the dataset.
- **Column names**: A list of all the column names in the DataFrame.
- **Non-null count**: The number of non-null values in each column (i.e., values that are not missing).
- **Data types**: The data type of each column (e.g., int64, float64, object).
- **Memory usage**: The total memory used by the DataFrame.

In [13]:
```
gpa_df.describe()
```

Out[13]:

	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

- The **describe()** function in pandas provides a summary of statistics for numerical columns in a DataFrame. It includes measures like count, mean, standard deviation, minimum, maximum, and percentiles.

In [14]:
```
gpa_df.isnull().sum()
```

Out[14]:
```
App                0
Category           0
Rating          1474
Reviews            0
Size               0
Installs           0
Type               1
Price              0
Content Rating     1
Genres             0
Last Updated       0
Current Ver        8
Android Ver        3
dtype: int64
```

- The **gpa_df.isnull().sum()** command will return a Series showing the count of missing (NaN) values for each column in your DataFrame. This helps identify which columns have missing values and how many there are. Here's how to interpret the output:
- **If the output is all zeros**, it means there are no missing values in any of the columns.
- **If there are non-zero values**, it indicates the number of missing values per column. You will need to address these missing values, either by filling them in or removing the affected rows.

In [15]:
```
gpa_df["Rating"].unique()
```

Out[15]:
```
array([ 4.1,  3.9,  4.7,  4.5,  4.3,  4.4,  3.8,  4.2,  4.6,  3.2,  4. ,
        nan,  4.8,  4.9,  3.6,  3.7,  3.3,  3.4,  3.5,  3.1,  5. ,  2.6,
        3. ,  1.9,  2.5,  2.8,  2.7,  1. ,  2.9,  2.3,  2.2,  1.7,  2. ,
        1.8,  2.4,  1.6,  2.1,  1.4,  1.5,  1.2, 19. ])
```

- The **unique()** function in pandas returns the unique values present in a specific column.
- When you run **gpa_df["Rating"].unique()**, it will return all the unique values in the "Rating" column of your dataset.

In [16]:
```
gpa_df["Category"].unique()
```

Out[16]:
```
array(['ART_AND_DESIGN', 'AUTO_AND_VEHICLES', 'BEAUTY',
       'BOOKS_AND_REFERENCE', 'BUSINESS', 'COMICS', 'COMMUNICATION',
       'DATING', 'EDUCATION', 'ENTERTAINMENT', 'EVENTS', 'FINANCE',
       'FOOD_AND_DRINK', 'HEALTH_AND_FITNESS', 'HOUSE_AND_HOME',
       'LIBRARIES_AND_DEMO', 'LIFESTYLE', 'GAME', 'FAMILY', 'MEDICAL',
       'SOCIAL', 'SHOPPING', 'PHOTOGRAPHY', 'SPORTS', 'TRAVEL_AND_LOCAL',
       'TOOLS', 'PERSONALIZATION', 'PRODUCTIVITY', 'PARENTING', 'WEATHER',
       'VIDEO_PLAYERS', 'NEWS_AND_MAGAZINES', 'MAPS_AND_NAVIGATION',
       '1.9'], dtype=object)
```

- When you run **gpa_df["Category"].unique()**, it will return all the unique values in the "Category" column of your dataset.

In [17]:
```
gpa_df.groupby(by = 'Category') ["Rating"].mean()
```

Out[17]:
```
Category
1.9                   19.000000
ART_AND_DESIGN         4.358065
AUTO_AND_VEHICLES      4.190411
BEAUTY                 4.278571
BOOKS_AND_REFERENCE    4.346067
BUSINESS               4.121452
COMICS                 4.155172
COMMUNICATION          4.158537
DATING                 3.970769
EDUCATION              4.389032
ENTERTAINMENT          4.126174
EVENTS                 4.435556
FAMILY                 4.192272
FINANCE                4.131889
FOOD_AND_DRINK         4.166972
GAME                   4.286326
HEALTH_AND_FITNESS     4.277104
HOUSE_AND_HOME         4.197368
LIBRARIES_AND_DEMO     4.178462
LIFESTYLE              4.094904
MAPS_AND_NAVIGATION    4.051613
MEDICAL                4.189143
NEWS_AND_MAGAZINES     4.132189
PARENTING              4.300000
PERSONALIZATION        4.335987
PHOTOGRAPHY            4.192114
PRODUCTIVITY           4.211396
SHOPPING               4.259664
SOCIAL                 4.255598
SPORTS                 4.223511
TOOLS                  4.047411
TRAVEL_AND_LOCAL       4.109292
VIDEO_PLAYERS          4.063750
WEATHER                4.244000
Name: Rating, dtype: float64
```

- The **groupby()** function in pandas allows you to group data by a specific column and perform aggregate functions like mean() on another column.
- In your case, you want to group the dataset by the **"Category"** column and calculate the mean of the **"Rating"** column for each category.

In [18]:
```
gpa_df["Rating"].mean()
```

Out[18]:
```
np.float64(4.193338315362443)
```

- **gpa_df["Rating"].mean():** Calculates the average (mean) value of the **"Rating"** column. This value represents the overall average rating for all apps in your dataset.

In [19]:
```
def imp_rating(category_classes):
    for category_class in category_classes:
        sub_set=gpa_df['Category']==category_class
        mean=gpa_df[gpa_df["Category"]==category_class]["Rating"].mean()
        gpa_df.loc[sub_set,"Rating"]=gpa_df.loc[sub_set,"Rating"].fillna(mea
```

In [21]:
```
category_classes = ['ART_AND_DESIGN', 'AUTO_AND_VEHICLES', 'BEAUTY',
       'BOOKS_AND_REFERENCE', 'BUSINESS', 'COMICS', 'COMMUNICATION',
       'DATING', 'EDUCATION', 'ENTERTAINMENT', 'EVENTS', 'FINANCE',
       'FOOD_AND_DRINK', 'HEALTH_AND_FITNESS', 'HOUSE_AND_HOME',
       'LIBRARIES_AND_DEMO', 'LIFESTYLE', 'GAME', 'FAMILY', 'MEDICAL',
       'SOCIAL', 'SHOPPING', 'PHOTOGRAPHY', 'SPORTS', 'TRAVEL_AND_LOCAL',
       'TOOLS', 'PERSONALIZATION', 'PRODUCTIVITY', 'PARENTING', 'WEATHER',
       'VIDEO_PLAYERS', 'NEWS_AND_MAGAZINES', 'MAPS_AND_NAVIGATION',
       '1.9']
imp_rating(category_classes)
```

- Loop Through Categories:
  - Iterate over each category in the category_classes list.
- Create a Boolean Mask:
  - sub_set is a boolean mask that selects rows where the "Category" matches the current category in the loop.
- Calculate Mean Rating:
  - Compute the mean rating for the current category.
- Fill Missing Values:
  - Use fillna() to replace NaN values in the "Rating" column with the computed mean for that category.

In [22]:
```
gpa_df.isnull().sum()
```

Out[22]:
```
App               0
Category          0
Rating            0
Reviews           0
Size              0
Installs          0
Type              1
Price             0
Content Rating    1
Genres            0
Last Updated      0
Current Ver       8
Android Ver       3
dtype: int64
```

- The isnull().sum() method in pandas is used to count the number of missing (NaN) values in each column of the DataFrame.

In [23]:
```
gpa_df['Type'] = gpa_df['Type'].fillna(value=0)
```

- The line of code you provided is intended to fill missing values (NaN) in the "Type" column with 0. However, the "Type" column is likely categorical (e.g., 'Free', 'Paid'), so replacing missing values with 0 might not be appropriate.

In [24]:
```
gpa_df.isnull().sum()
```

Out[24]:
```
App               0
Category          0
Rating            0
Reviews           0
Size              0
Installs          0
Type              0
Price             0
Content Rating    1
Genres            0
Last Updated      0
Current Ver       8
Android Ver       3
dtype: int64
```

In [25]:
```
gpa_df['Content Rating'] = gpa_df['Content Rating'].fillna(value=0)
```

- Filling missing values in the "Content Rating" column with 0 might not be appropriate since "Content Rating" typically contains categorical values such as 'Everyone', 'Teen', 'Mature 17+', etc. Using 0 could introduce inconsistencies or errors in your data.

In [26]:
```
gpa_df.isnull().sum()
```

Out[26]:
```
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       8
Android Ver       3
dtype: int64
```

In [ ]:
```
gpa_df['Current Ver'] = gpa_df['Current Ver'].fillna(value=0)
```

- Filling missing values in the "Current Ver" column with 0 might not be appropriate since "Current Ver" typically contains version numbers in a string format (e.g., '1.0', '2.3.4'). Using 0 can create inconsistencies in this column.

In [27]:
```
gpa_df.isnull().sum()
```

Out[27]:
```
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       8
Android Ver       3
dtype: int64
```

In [28]:
```
gpa_df['Android Ver'] = gpa_df['Android Ver'].fillna(value=0)
```

- Filling missing values in the "Android Ver" column with 0 might not be suitable since "Android Ver" typically contains version numbers in a string format (e.g., '4.4', '5.0', '6.0.1'). Using 0 can lead to inconsistencies in your data.

In [29]:
```
gpa_df.isnull().sum()
```

Out[29]:
```
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       8
Android Ver       0
dtype: int64
```

In [30]:
```
gpa_df.columns
```

Out[30]:
```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type',
       'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver',
       'Android Ver'],
      dtype='object')
```

- To view the column names in your DataFrame, you can use the columns attribute.

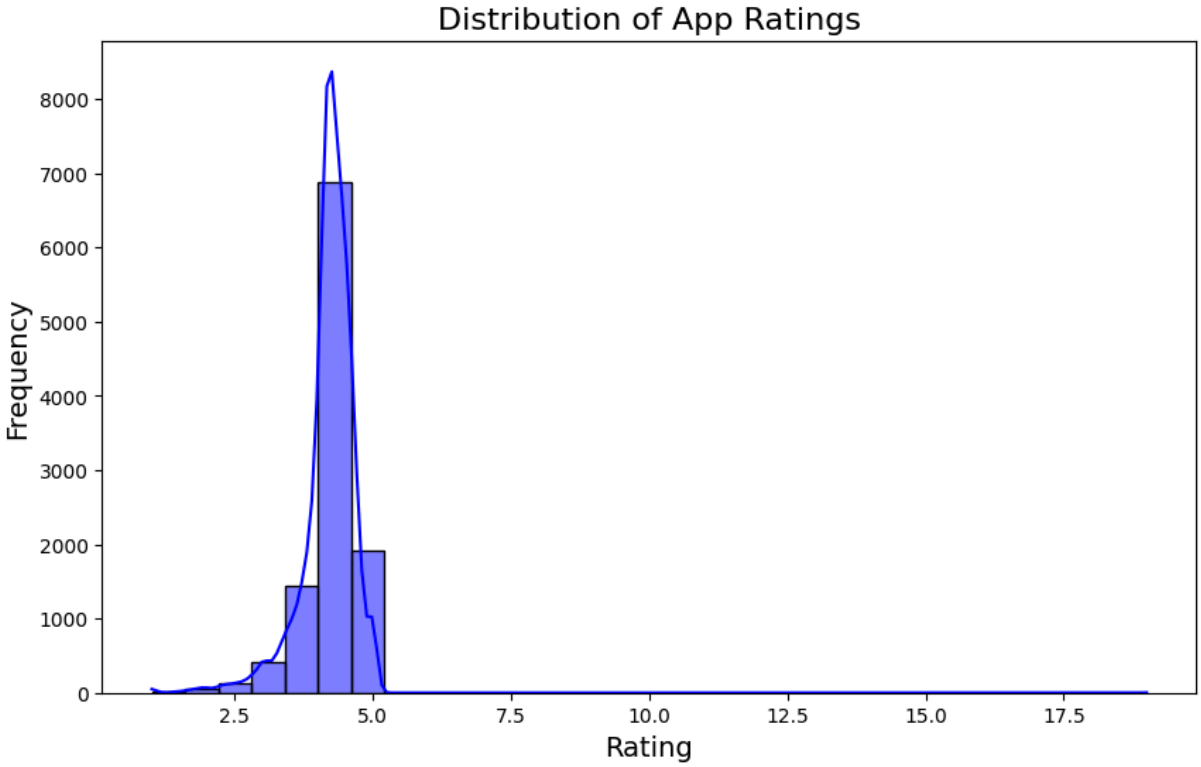
Exploratory Analysis and Visualization

- "The graphical representation of data to identify patterns, trends, and insights through visual elements such as charts, graphs, and maps."

1. Distribution of App Ratings

- Visualization: Histogram
- Purpose: To observe how app ratings are distributed across the dataset (e.g., are most apps rated between 4 and 5 stars?).

In [34]:
```
plt.figure(figsize=(10,6))
sns.histplot(gpa_df['Rating'].dropna(), bins=30, kde=True, color='blue')
plt.title('Distribution of App Ratings', fontsize=16)
plt.xlabel('Rating', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.show();
```

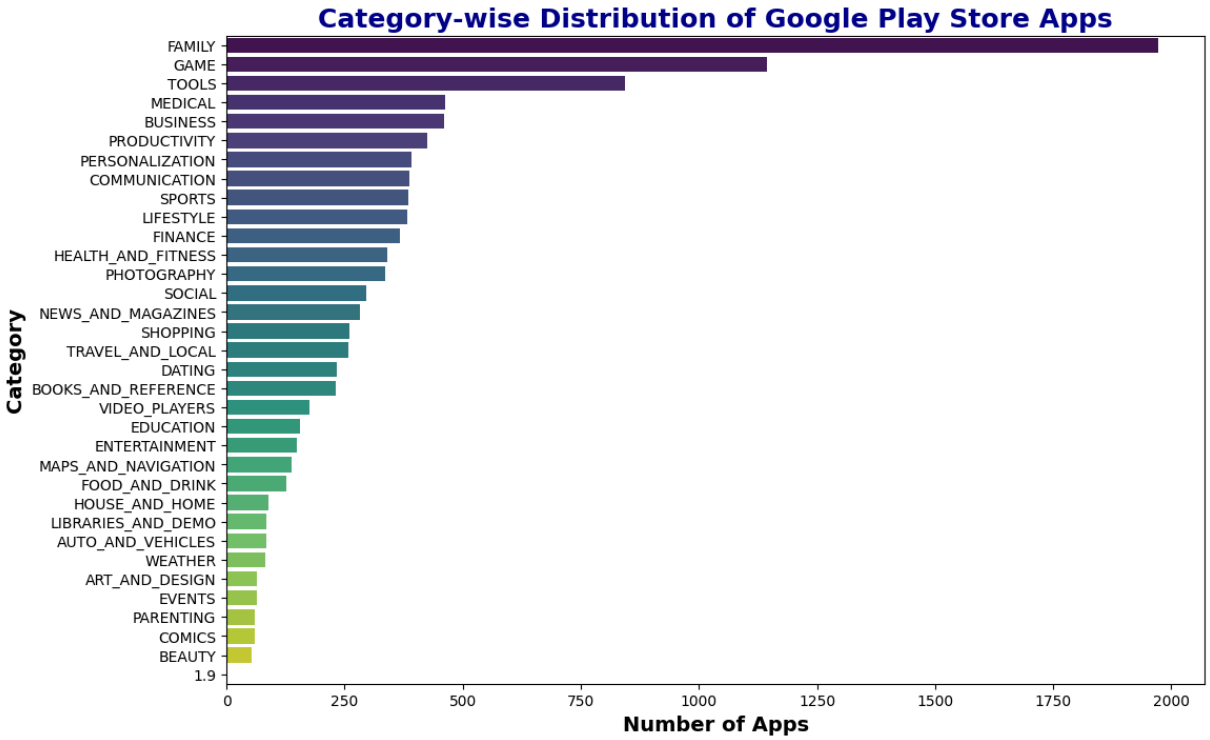


- The histogram shows the frequency distribution of the app ratings, with each bar representing the count of ratings that fall within a specific range.
- The KDE curve provides a smoothed line to show the overall distribution trend, giving an idea of how the data is distributed across different rating values.
- The x-axis represents the rating values (e.g., IMDB ratings, app ratings), and the y-axis represents the frequency or number of occurrences for each rating.
- `The plot helps visualize the spread and concentration of ratings in your dataset, showing whether most ratings are high, low, or evenly distributed.`

2. Category-wise Distribution of Apps

- Visualization: Bar Plot
- Purpose: To analyze the number of apps across different categories (e.g., games, education, business, etc.).

In [43]:
```
category_counts = df['Category'].value_counts()
plt.figure(figsize=(12,8))
sns.barplot(x=category_counts.values, y=category_counts.index, palette='viri
plt.title('Category-wise Distribution of Google Play Store Apps', fontsize=1
plt.xlabel('Number of Apps', fontsize=14, fontweight='bold')
plt.ylabel('Category', fontsize=14, fontweight='bold')
plt.show();
```



- The horizontal bar plot shows the number of apps in each category.
- The y-axis lists the categories (e.g., Games, Tools, Productivity) found in the Google Play Store.
- The x-axis shows the number of apps in each category, allowing for easy comparison across categories.
- The color gradient (using the viridis palette) enhances the visual appeal and highlights differences in app counts across categories.
- `This plot helps to quickly identify which categories have the most or least apps, providing insights into the app distribution across various categories on the Google Play Store.`

3. Free vs. Paid Apps Distribution

- Visualization: Pie Chart
- Purpose: To show the proportion of free versus paid apps in the store.

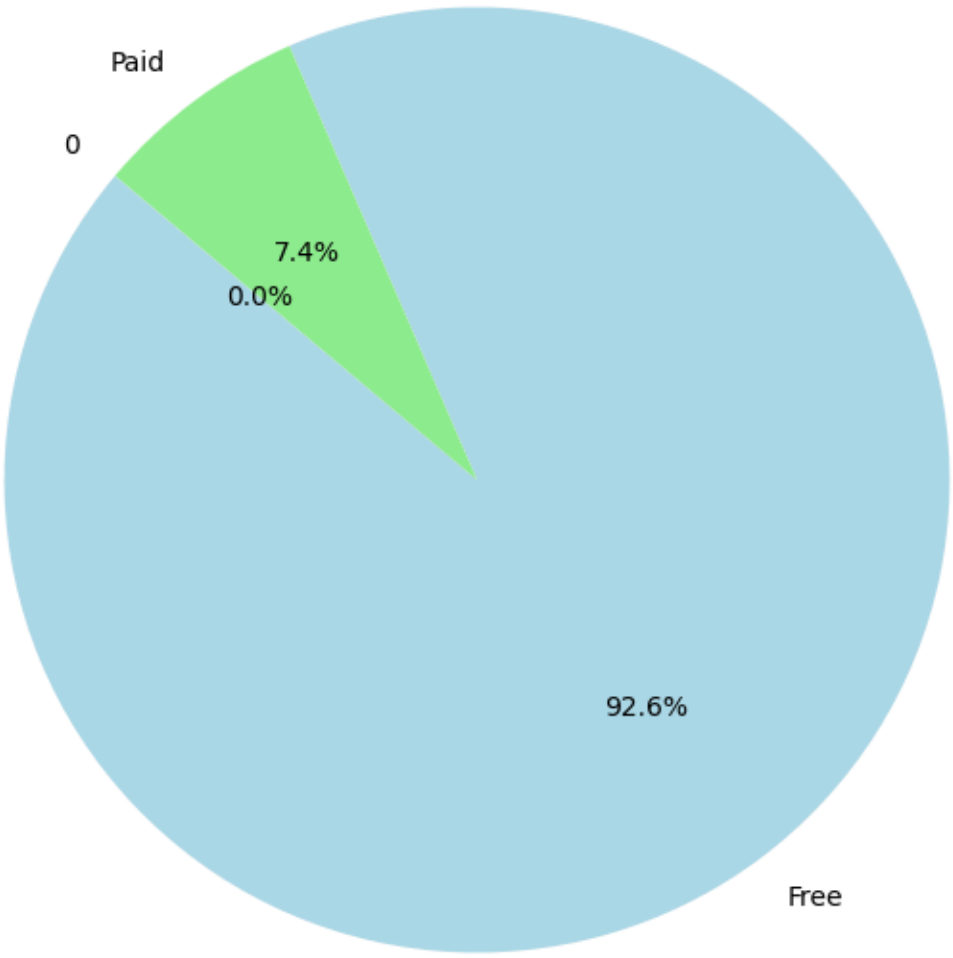
In [58]:
```
app_type_counts = df['Type'].value_counts()

plt.figure(figsize=(8, 8))

plt.pie(app_type_counts, labels=app_type_counts.index, autopct='%1.1f%%', cc

plt.title('Distribution of Free vs. Paid Apps', fontsize=18, fontweight='bol

plt.show()
```



- The pie chart will show the distribution of Free and Paid apps as proportional slices of the whole chart.
- Each slice represents the proportion of apps that are either Free or Paid, and the exact percentage is displayed on each slice.
- The colors (light blue and light green) distinguish between the two types of apps, and the chart starts at an angle of 140 degrees to ensure a balanced view.
- `This visualization helps to easily understand how the majority of apps are distributed between free and paid types, providing a clear comparison of the two categories.`

4. Number of Apps in Each Content Rating

- Visualization: Countplot
- Purpose: To visualize the distribution of apps across different content ratings, highlighting which ratings are most common.

In [127…

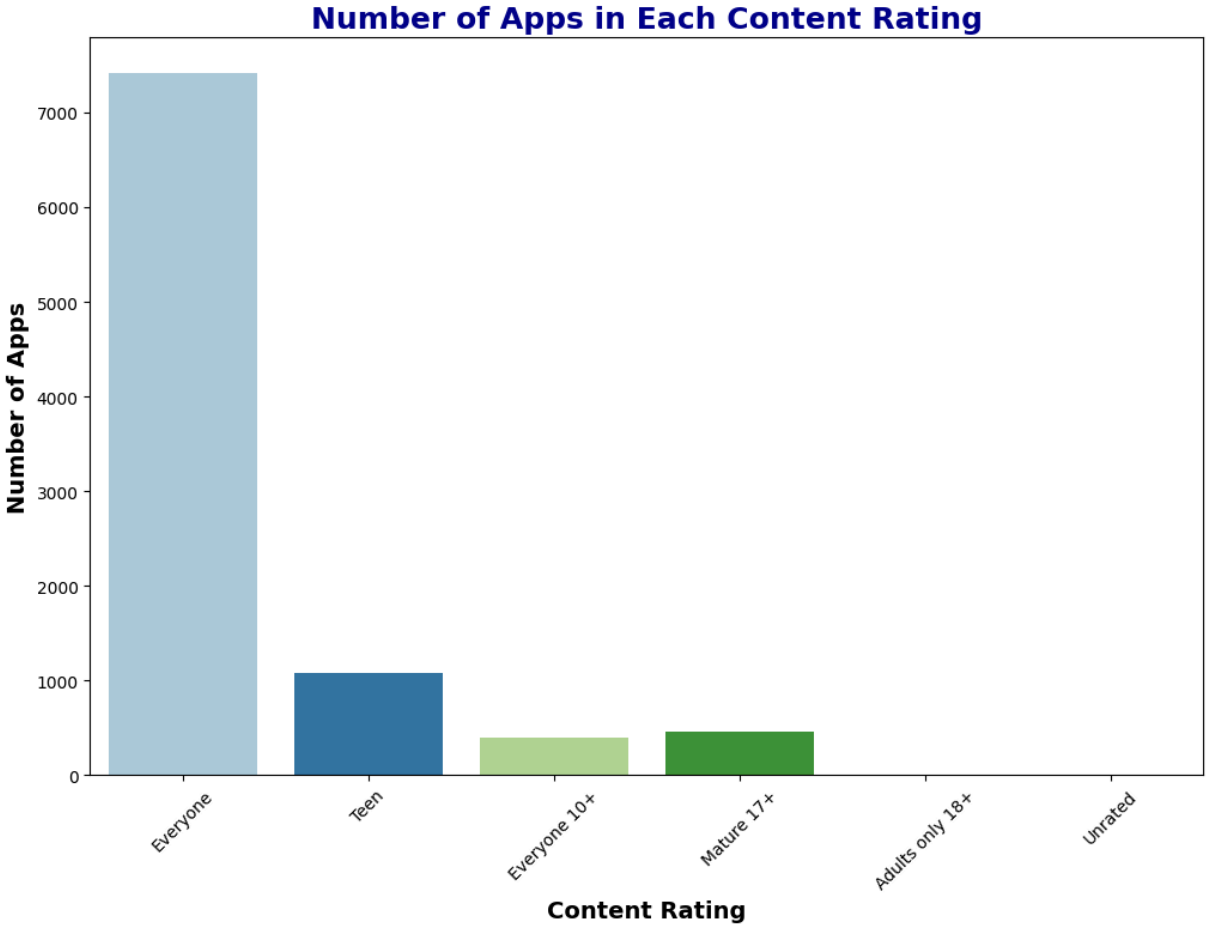
```
df = df.dropna(subset=['Content Rating'])

plt.figure(figsize=(12, 8))

sns.countplot(x='Content Rating', data=df, palette='Paired')

plt.title('Number of Apps in Each Content Rating', fontsize=18, fontweight='
plt.xlabel('Content Rating', fontsize=14, fontweight='bold')
plt.ylabel('Number of Apps', fontsize=14, fontweight='bold')

plt.xticks(rotation=45)
plt.show()
```



- The count plot shows the number of apps available for each content rating (e.g., Everyone, Teen, Mature, Adults Only).
- The x-axis lists the content rating categories, while the y-axis shows the number of apps in each category.
- Each bar represents the count of apps for a specific content rating, and the distinct colors (from the Paired palette) make it easy to differentiate between

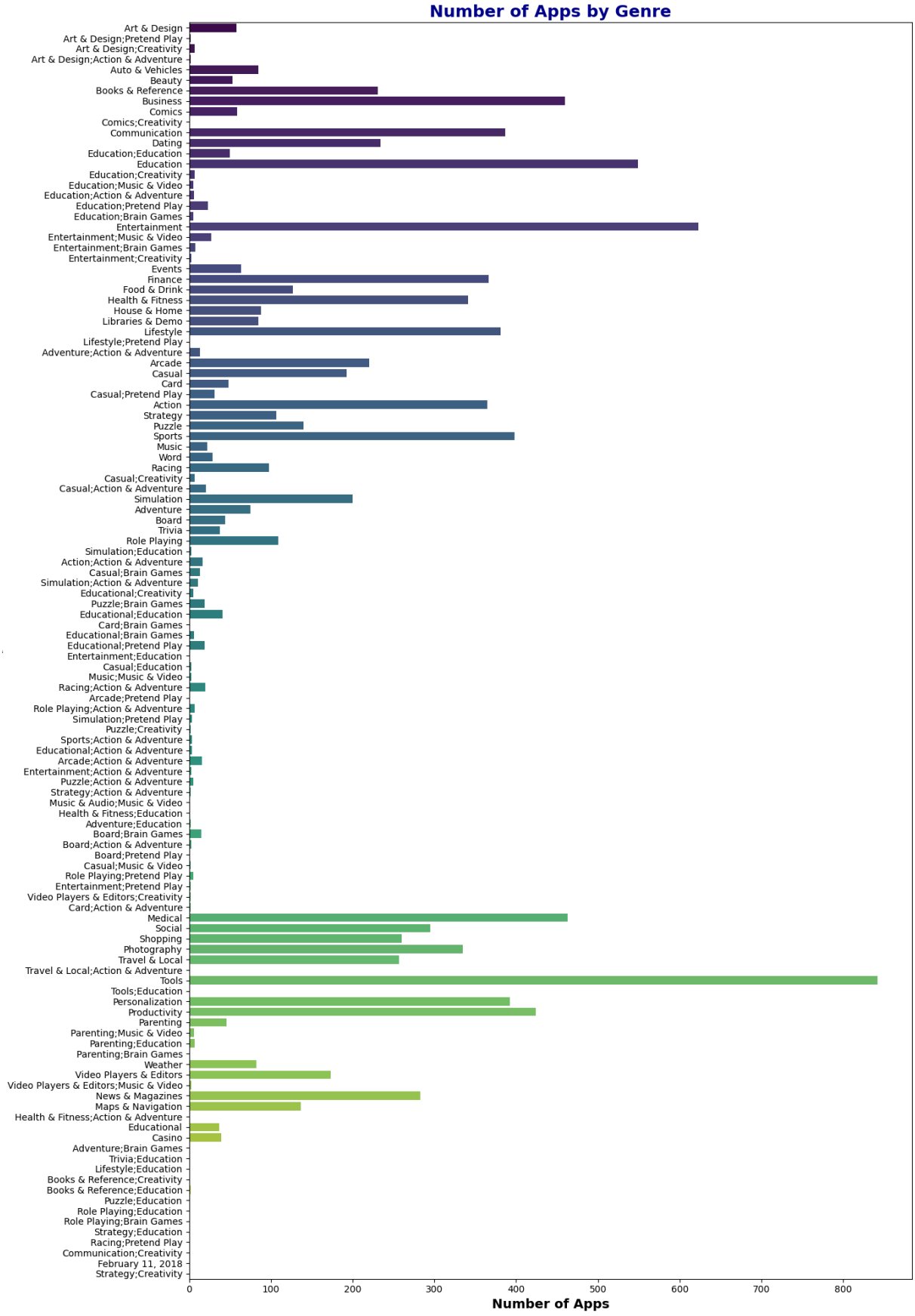
categories.

- Rotating the x-axis labels helps ensure that longer category names (if present) are readable.
- `This plot provides a clear overview of the distribution of apps based on their content ratings, highlighting which content rating categories have the most or fewest apps.`

5. Number of Apps by Genre

- Visualization: Countplot
- Purpose: To visualize the distribution of the number of apps across different genres to identify which genres are most prevalent.

In [82]:
```
plt.figure(figsize=(14, 20))
sns.countplot(y='Genres', data=df, palette='viridis')
plt.title('Number of Apps by Genre', fontsize=18, fontweight='bold', color='
plt.xlabel('Number of Apps', fontsize=14, fontweight='bold')
plt.ylabel('Genre', fontsize=1, fontweight='bold')
plt.show()
```



- The count plot will display the number of apps in each genre on the x-axis, with each genre listed on the y-axis.

- The bars represent the count of apps in each genre, and the colors from the viridis palette make the chart more visually engaging.
- The x-axis shows the number of apps for each genre, and the y-axis lists the genre names.

6. Top 10 Categories by Number of Apps

- Visualization: Barplot
- Purpose: Highlight the top categories with the most apps.

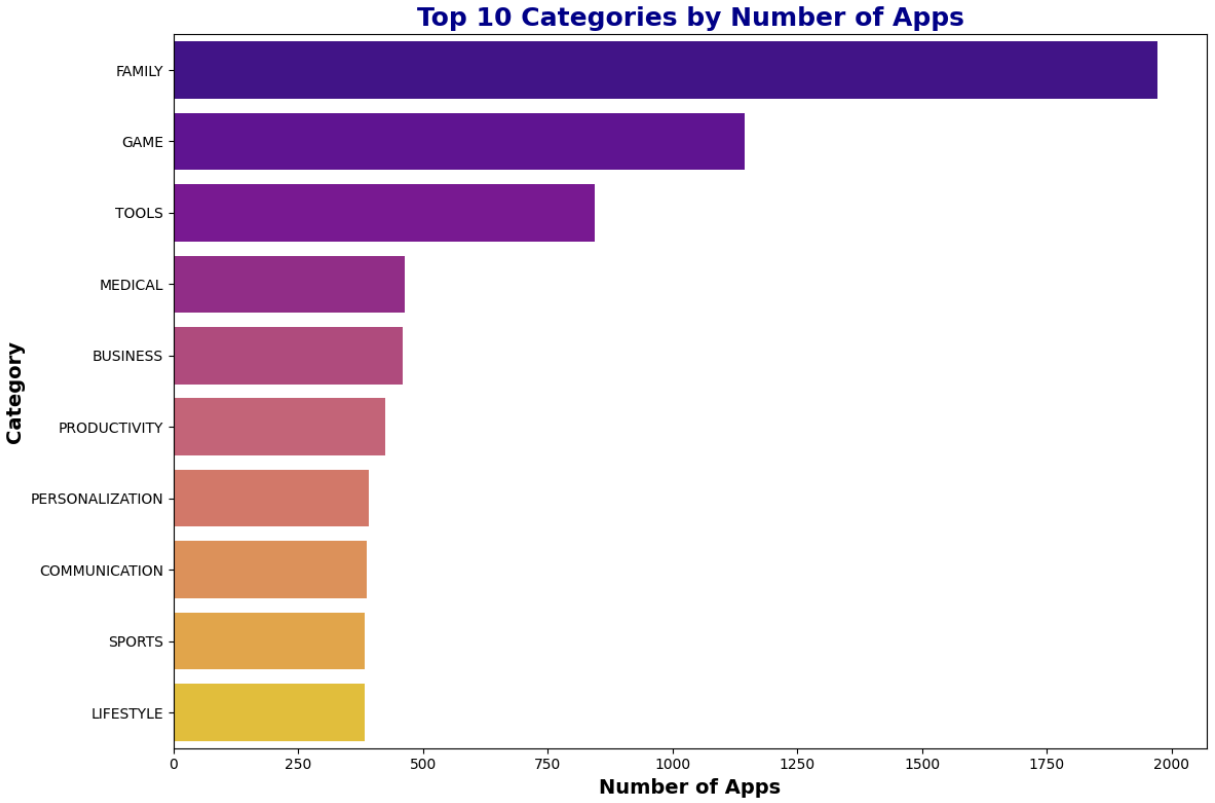
In [84]:
```
category_counts = df['Category'].value_counts().head(10)

plt.figure(figsize=(12, 8))

# Bar plot for top 10 categories
sns.barplot(x=category_counts.values, y=category_counts.index, palette='plas

# Customize the title and axis labels
plt.title('Top 10 Categories by Number of Apps', fontsize=18, fontweight='bo
plt.xlabel('Number of Apps', fontsize=14, fontweight='bold')
plt.ylabel('Category', fontsize=14, fontweight='bold')

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

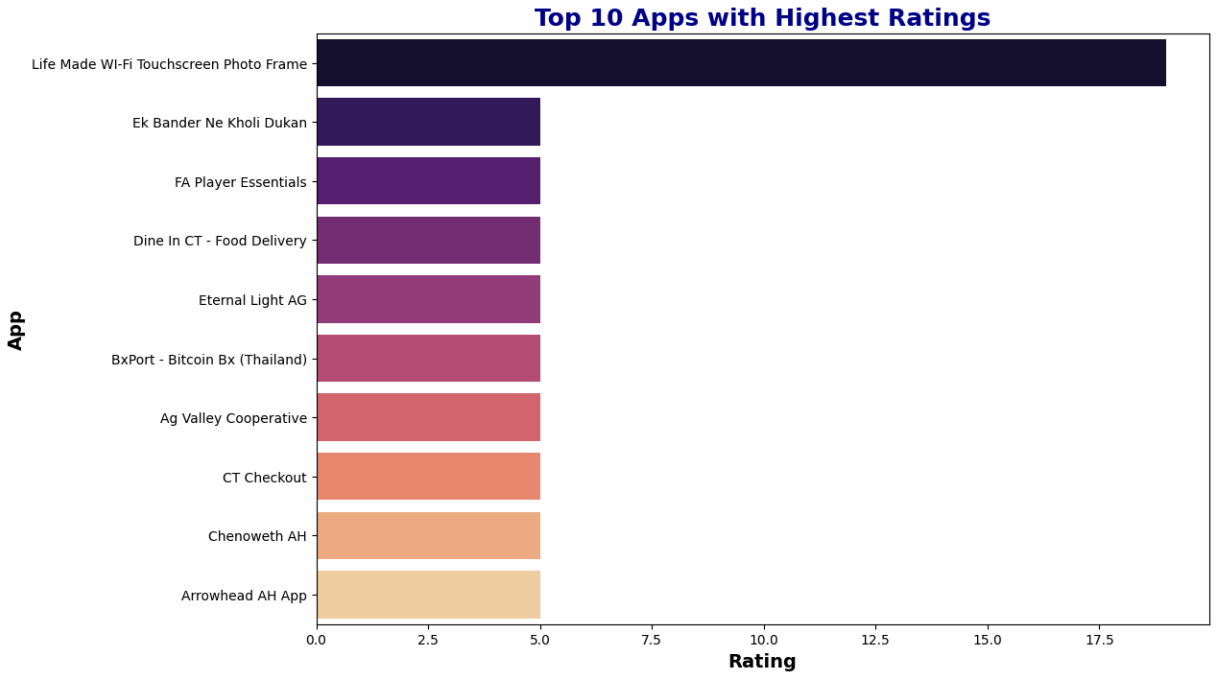


- The bar plot will show the top 10 categories with the highest number of apps.
- Each bar represents the count of apps in one of the top 10 categories, with the length of the bar corresponding to the number of apps.
- The x-axis displays the number of apps, and the y-axis lists the category names.

7. Top 10 Apps with Highest Ratings

- Visualization: Barplot
- Purpose: Identify the highest-rated apps.

In [92]:
```
df = df.dropna(subset=['Rating'])
top_rated_apps = df[['App', 'Rating']].sort_values(by='Rating', ascending=Fa
plt.figure(figsize=(12, 8))
sns.barplot(x='Rating', y='App', data=top_rated_apps, palette='magma')
plt.title('Top 10 Apps with Highest Ratings', fontsize=18, fontweight='bold'
plt.xlabel('Rating', fontsize=14, fontweight='bold')
plt.ylabel('App', fontsize=14, fontweight='bold')
plt.show()
```

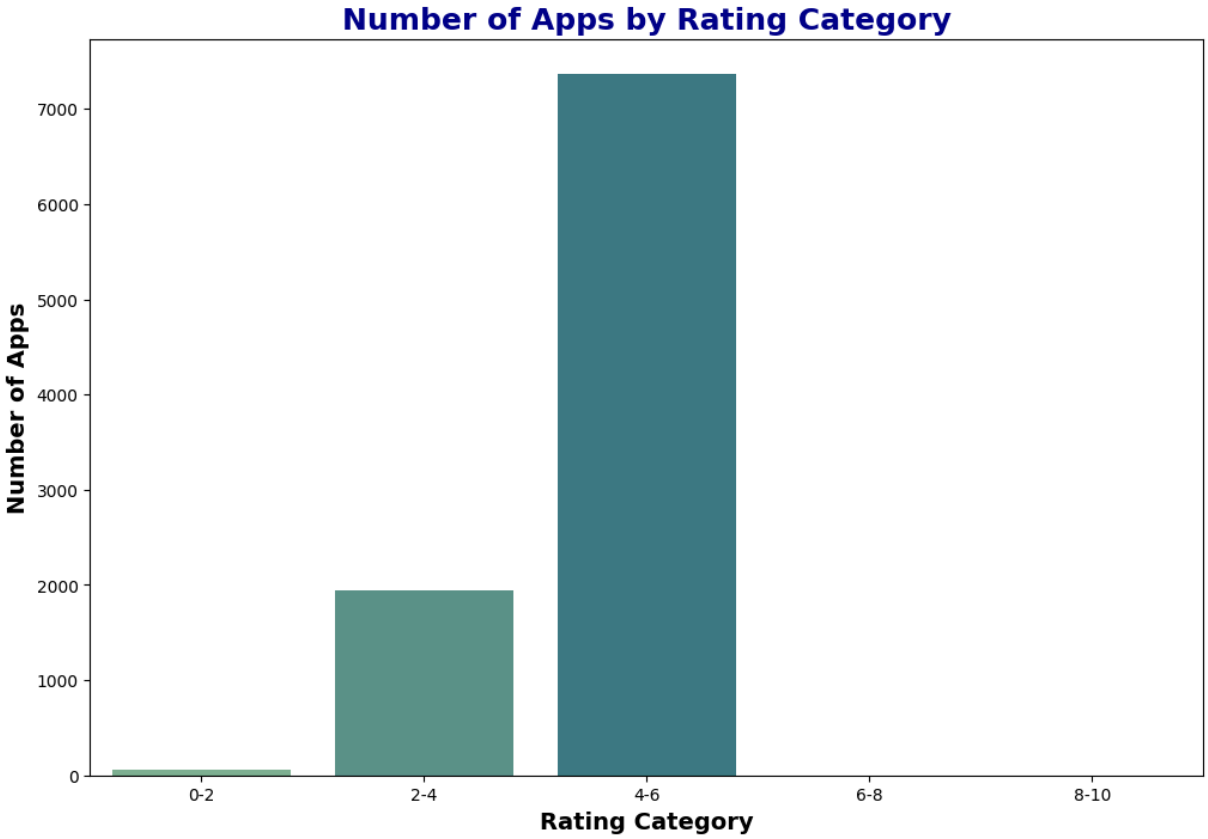


- The bar plot will display the top 10 apps with the highest ratings.
- Each bar represents an app, with the length of the bar corresponding to the app's rating.
- The x-axis shows the ratings, and the y-axis lists the app names.

8. Number of Apps by Rating Category

- Visualization: Countplot
- Purpose Understand how apps are distributed across rating categories..

In [93]:
```
df = df.dropna(subset=['Rating'])
bins = [0, 2, 4, 6, 8, 10]
labels = ['0-2', '2-4', '4-6', '6-8', '8-10']
df['Rating Category'] = pd.cut(df['Rating'], bins=bins, labels=labels, right
plt.figure(figsize=(12, 8))
sns.countplot(x='Rating Category', data=df, palette='crest')
plt.title('Number of Apps by Rating Category', fontsize=18, fontweight='bold
plt.xlabel('Rating Category', fontsize=14, fontweight='bold')
plt.ylabel('Number of Apps', fontsize=14, fontweight='bold')
plt.show()
```

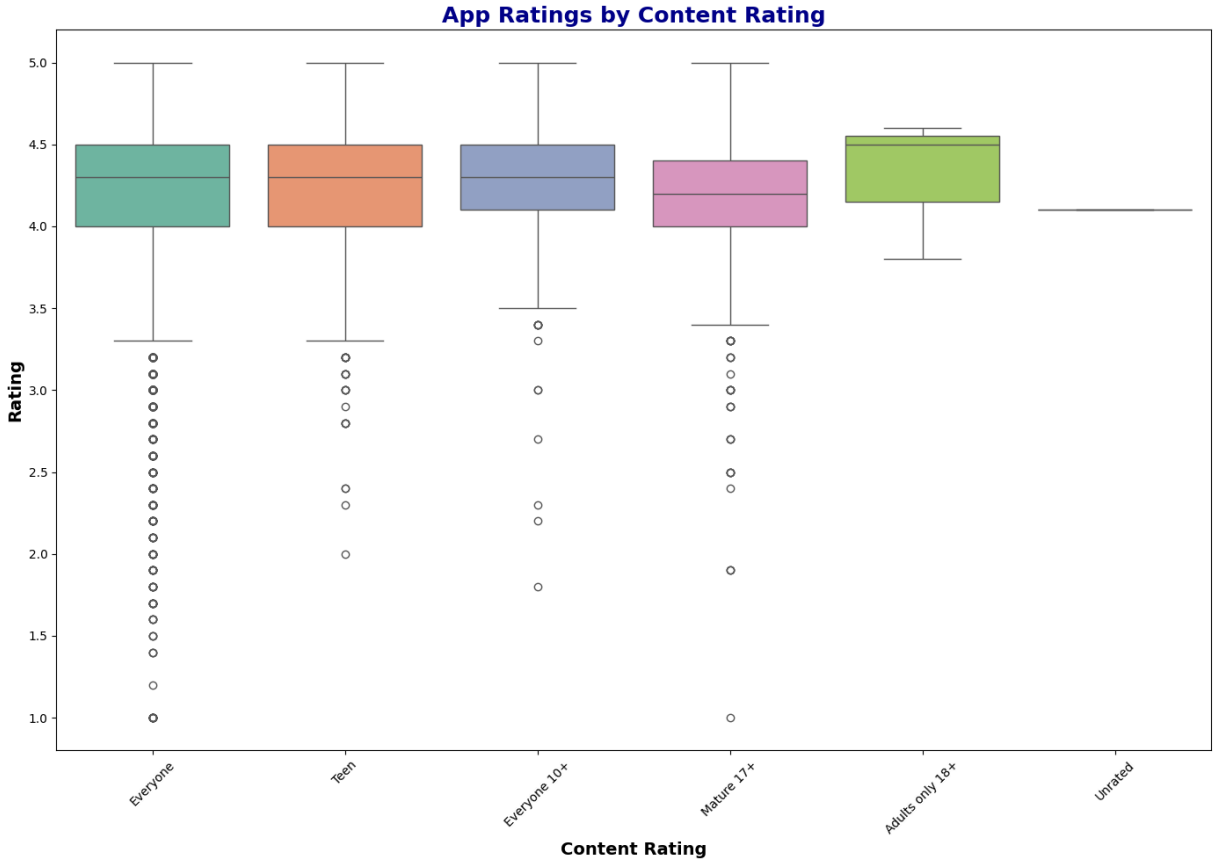


- The count plot will display the number of apps in each rating category.
- The x-axis shows the rating categories (e.g., 0-2, 2-4, etc.), and the y-axis shows the number of apps in each category.
- The bars represent the count of apps falling into each rating category, with the colors from the crest palette enhancing the visual appeal.
- `This plot helps to understand the distribution of app ratings across different categories, showing how many apps fall into each rating range.`

9. App Ratings by Content Rating

- Visualization: Boxplot
- Purpose: Compare the average size of apps across content ratings. .

In [95]:
```
df = df.dropna(subset=['Rating', 'Content Rating'])
plt.figure(figsize=(14, 10))
sns.boxplot(x='Content Rating', y='Rating', data=df, palette='Set2')
plt.title('App Ratings by Content Rating', fontsize=18, fontweight='bold', c
plt.xlabel('Content Rating', fontsize=14, fontweight='bold')
plt.ylabel('Rating', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```



- The box plot will show the distribution of app ratings for each content rating category.
- Each box represents the interquartile range (IQR) of ratings for a specific content rating category, with the line inside the box showing the median rating.
- The x-axis displays different content rating categories (e.g., Everyone, Teen, Mature), and the y-axis shows the app ratings.
- `This plot helps to understand how app ratings vary by content rating category, highlighting differences and trends in ratings across various categories.`

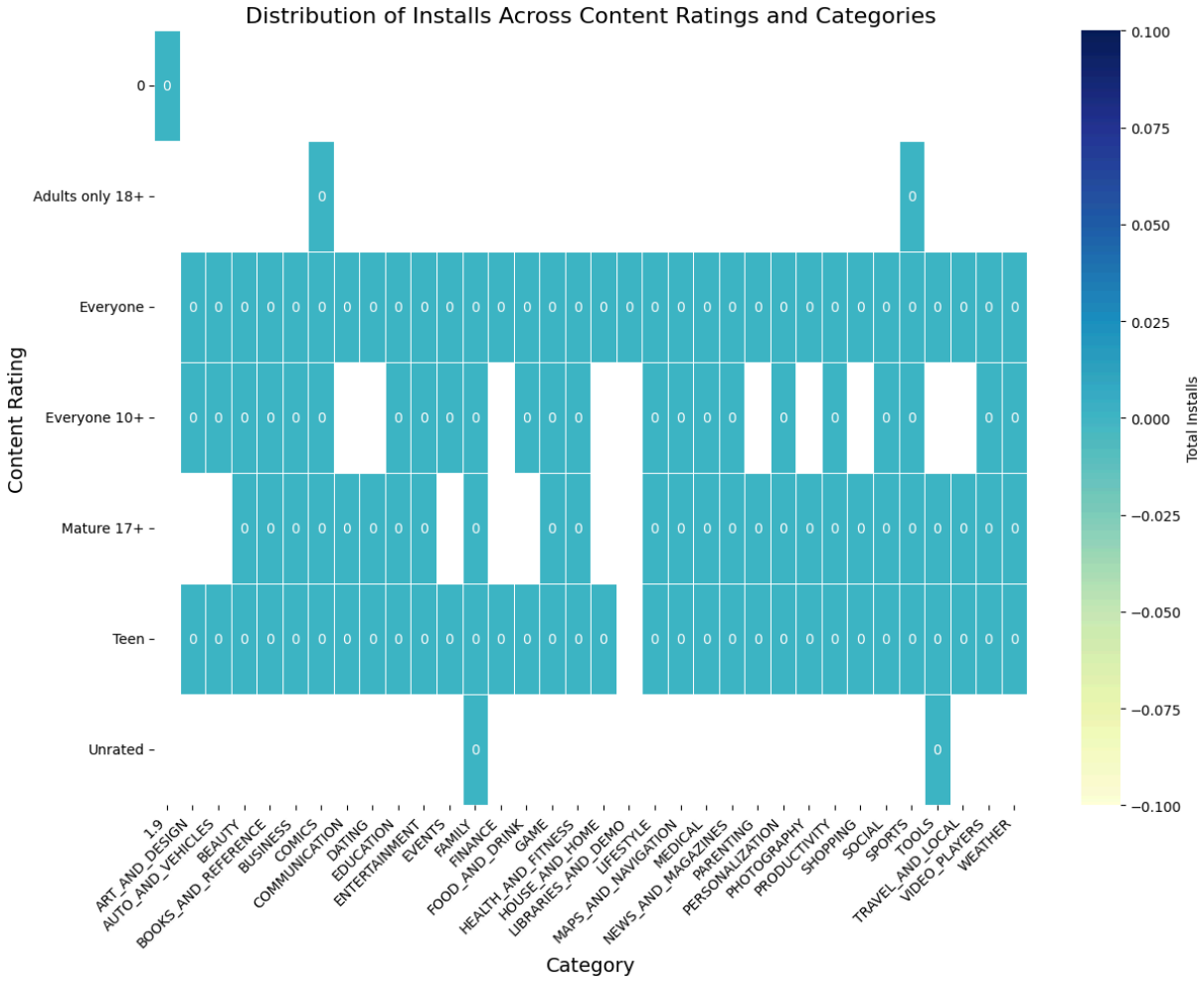
10.Distribution of Installs Across Content Ratings and Categories

- Visualization: Heatmap
- Purpose: To show how apps with different content ratings (e.g., Everyone, Teen, Mature) compare in terms of installs.

In [126…

```
plt.figure(figsize=(14, 10))
ax = sns.heatmap(
    pivot_df,
    annot=True,
    cmap='YlGnBu',
    fmt='g',
    linewidths=.5,
    cbar_kws={'label': 'Total Installs'}
)

plt.title('Distribution of Installs Across Content Ratings and Categories',
plt.xlabel('Category', fontsize=14)
plt.ylabel('Content Rating', fontsize=14)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.show()
```



- The heatmap visualizes the distribution of installs for different combinations of content ratings and categories.
- Each cell in the heatmap represents the total number of installs for a particular combination of content rating and category.
- The color intensity varies based on the value, with the color bar indicating the scale of total installs.
- The annotations within the cells show the exact number of installs, making it easy to read the data directly from the heatmap.
- The x-axis represents categories, and the y-axis represents content ratings.
- `This heatmap helps to identify patterns and trends in the distribution of installs across different content ratings and categories, making it easier to analyze how app popularity varies by category and content rating.`

Happy Learning! 📚✨

Thank You