# Introduction

In this notebook, we will do a comprehensive analysis of the Android app market by comparing thousands of apps in the Google Play store.

# About the Dataset of Google Play Store Apps & Reviews

**Data Source:**
App and review data was scraped from the Google Play Store by Lavanya Gupta in 2018. Original files listed here.

# Import Statements

```
import pandas as pd
import plotly.express as px
```

# Notebook Presentation

```
# Show numeric output in decimal format e.g., 2.15
pd.options.display.float_format = '{:,.2f}'.format
```

# Read the Dataset

```
df_apps = pd.read_csv('data/apps.csv')
df_apps.head()
```

| | App | Category | Rating | Reviews | Size_MBs | Installs | Type | Price | Content_Rating | |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Ak Parti Yardım Toplama | SOCIAL | NaN | 0 | 8.70 | 0 | Paid | $13.99 | Teen | |
| **1** | Ain Arabic Kids Alif Ba ta | FAMILY | NaN | 0 | 33.00 | 0 | Paid | $2.99 | Everyone | |
| **2** | Popsicle Launcher for Android P 9.0 launcher | PERSONALIZATION | NaN | 0 | 5.50 | 0 | Paid | $1.49 | Everyone | Pers |
| **3** | Command & Conquer: Rivals | FAMILY | NaN | 0 | 19.00 | 0 | NaN | 0 | Everyone 10+ | |

# Data Cleaning

Checking How many rows and columns does `df_apps` have? What are the column names? Looking at a random sample of 5 different rows with .sample().

```
df_apps.shape
```

```
(10841, 12)
```

```
df_apps.columns
```

```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size_MBs', 'Installs', 'Type',
       'Price', 'Content_Rating', 'Genres', 'Last_Updated', 'Android_Ver'],
      dtype='object')
```

```
df_apps.sample(5)
```

| | App | Category | Rating | Reviews | Size_MBs | Installs | Type | Price | Content_Rating | |
|---|---|---|---|---|---|---|---|---|---|---|
| **3766** | T Uploader | FAMILY | 4.10 | 75 | 2.20 | 10,000 | Free | 0 | Everyone | |
| **5297** | Sugar Daddy Dating App | DATING | 2.50 | 277 | 5.70 | 100,000 | Free | 0 | Mature 17+ | |
| **4411** | Learning English for children | PARENTING | NaN | 67 | 15.00 | 50,000 | Free | 0 | Everyone | Par |
| **5796** | Morse Code Reader | COMMUNICATION | 3.90 | 1436 | 0.02 | 100,000 | Free | 0 | Everyone | |
| **2647** | go41cx | FAMILY | 4.80 | 171 | 1.00 | 1,000 | Paid | $10.00 | Everyone | |

## Drop Unused Columns

Removing the columns called `Last_Updated` and `Android_Version` from the DataFrame. We will not use these columns. By setting axis=1 we are specifying that we want to drop certain columns.

```
df_apps.drop(['Last_Updated', 'Android_Ver'], axis=1, inplace=True)
df_apps.head()
```

| | App | Category | Rating | Reviews | Size_MBs | Installs | Type | Price | Content_Rating |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Ak Parti Yardım Toplama | SOCIAL | NaN | 0 | 8.70 | 0 | Paid | $13.99 | Teen |
| **1** | Ain Arabic Kids Alif Ba ta | FAMILY | NaN | 0 | 33.00 | 0 | Paid | $2.99 | Everyone |
| | Popsicle Launcher for | | | | | | | | |

## Find and Remove NaN values in Ratings

Checking how may rows have a NaN value (not-a-number) in the Ratings column? Creating DataFrame called `df_apps_clean` that does not include these rows.

```
nan_rows = df_apps[df_apps.Rating.isna()]
nan_rows.shape
```

```
(1474, 10)
```

droping the NaN rows

```
df_apps_clean = df_apps.dropna()
df_apps_clean.shape
```

```
(9367, 10)
```

## Find and Remove Duplicates

Checking for any duplicates in data? Checking for duplicates using the .duplicated() function. How many entries can you find for the "Instagram" app? Using .drop_duplicates() to remove any duplicates from `df_apps_clean`.

```
duplicated_rows = df_apps_clean[df_apps_clean.duplicated()]
print(duplicated_rows.shape)
duplicated_rows.head()
```

```
(476, 10)
```

| | App | Category | Rating | Reviews | Size_MBs | Installs | Type | Price | Content_Rating | Genres |
|---|---|---|---|---|---|---|---|---|---|---|
| **946** | 420 BZ Budeze Delivery | MEDICAL | 5.00 | 2 | 11.00 | 100 | Free | 0 | Mature 17+ | Medical |
| **1133** | MouseMingle | DATING | 2.70 | 3 | 3.90 | 100 | Free | 0 | Mature 17+ | Dating |
| **1196** | Cardiac diagnosis (heart rate, arrhythmia) | MEDICAL | 4.40 | 8 | 6.50 | 100 | Paid | $12.99 | Everyone | Medical |

```
df_apps_clean[df_apps_clean.App == 'Instagram']
```

| | App | Category | Rating | Reviews | Size_MBs | Installs | Type | Price | Content_Rating | Genre |
|---|---|---|---|---|---|---|---|---|---|---|
| **10806** | Instagram | SOCIAL | 4.50 | 66577313 | 5.30 | 1,000,000,000 | Free | 0 | Teen | Soci |
| **10808** | Instagram | SOCIAL | 4.50 | 66577446 | 5.30 | 1,000,000,000 | Free | 0 | Teen | Soci |
| **10809** | Instagram | SOCIAL | 4.50 | 66577313 | 5.30 | 1,000,000,000 | Free | 0 | Teen | Soci |
| **10810** | Instagram | SOCIAL | 4.50 | 66509917 | 5.30 | 1,000,000,000 | Free | 0 | Teen | Soci |

```
df_apps_clean = df_apps_clean.drop_duplicates()
df_apps_clean.shape
```

```
(8891, 10)
```

```
df_apps_clean[df_apps_clean.App == 'Instagram']
```

| | App | Category | Rating | Reviews | Size_MBs | Installs | Type | Price | Content_Rating | Genre |
|---|---|---|---|---|---|---|---|---|---|---|
| **10806** | Instagram | SOCIAL | 4.50 | 66577313 | 5.30 | 1,000,000,000 | Free | 0 | Teen | Soci |
| **10808** | Instagram | SOCIAL | 4.50 | 66577446 | 5.30 | 1,000,000,000 | Free | 0 | Teen | Soci |
| **10810** | Instagram | SOCIAL | 4.50 | 66509917 | 5.30 | 1,000,000,000 | Free | 0 | Teen | Soci |

```
# we need to specify the subset for indetifying duplicates
df_apps_clean = df_apps_clean.drop_duplicates(subset=['App', 'Type', 'Price'])
df_apps_clean[df_apps_clean.App == 'Instagram']
```

| | App | Category | Rating | Reviews | Size_MBs | Installs | Type | Price | Content_Rating | Genre |
|---|---|---|---|---|---|---|---|---|---|---|
| **10806** | Instagram | SOCIAL | 4.50 | 66577313 | 5.30 | 1,000,000,000 | Free | 0 | Teen | Soci |

```
df_apps_clean.shape
```

```
(8199, 10)
```

# Find Highest Rated Apps

Identify which apps are the highest rated. What problem might you encounter if you rely exclusively on ratings alone to determine the quality of an app?

```
df_apps_clean.sort_values(by=['Rating'] ,ascending=False).head()
```

| App | Category | Rating | Reviews | Size_MBs | Installs | Type | Price | Content_Rating |
|---|---|---|---|---|---|---|---|---|
| KBA-EZ | | | | | | | | |

## Find 5 Largest Apps in terms of Size (MBs)

Finding the size in megabytes (MB) of the largest Android apps in the Google Play Store. Based on the data, do you think there could be limit in place or can developers make apps as large as they please?

```
df_apps_clean.sort_values(by=['Size_MBs'] ,ascending=False).head()
```

| | App | Category | Rating | Reviews | Size_MBs | Installs | Type | Price | Content_Rat |
|---|---|---|---|---|---|---|---|---|---|
| **9942** | Talking Babsy Baby: Baby Games | LIFESTYLE | 4.00 | 140995 | 100.00 | 10,000,000 | Free | 0 | Every |
| **10687** | Hungry Shark Evolution | GAME | 4.50 | 6074334 | 100.00 | 100,000,000 | Free | 0 | T |
| **9943** | Miami crime simulator | GAME | 4.00 | 254518 | 100.00 | 10,000,000 | Free | 0 | Mature |

## Find the 5 App with Most Reviews

Finding apps that have the highest number of reviews? Are there any paid apps among the top 50?

```
df_apps_clean.sort_values(by=['Reviews'] ,ascending=False).head()
```

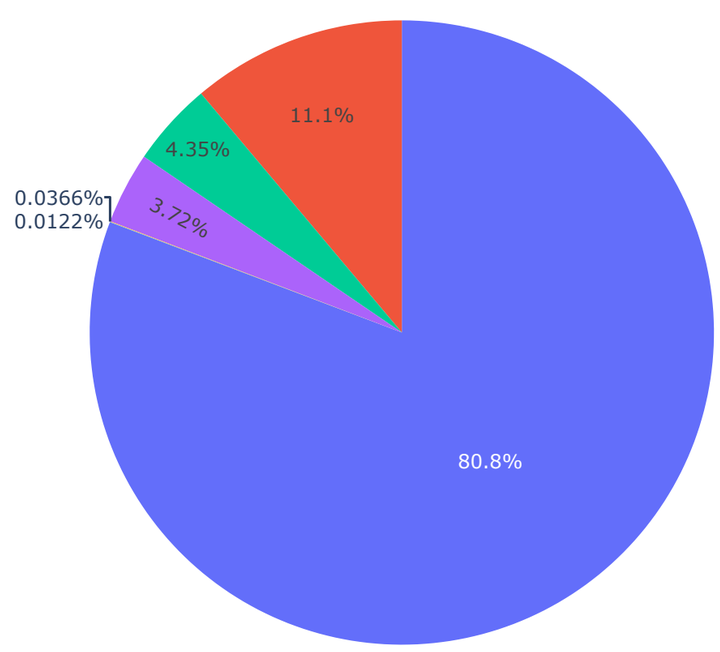
| | App | Category | Rating | Reviews | Size_MBs | Installs | Type | Price | Content_Rati |
|---|---|---|---|---|---|---|---|---|---|
| **10805** | Facebook | SOCIAL | 4.10 | 78158306 | 5.30 | 1,000,000,000 | Free | 0 | Te |
| **10785** | WhatsApp Messenger | COMMUNICATION | 4.40 | 69119316 | 3.50 | 1,000,000,000 | Free | 0 | Everyo |
| **10806** | Instagram | SOCIAL | 4.50 | 66577313 | 5.30 | 1,000,000,000 | Free | 0 | Te |
| **10784** | Messenger – Text and Video Chat for Free | COMMUNICATION | 4.00 | 56642847 | 3.50 | 1,000,000,000 | Free | 0 | Everyo |
| | Clash of | | | | | | | | |

## Plotly Pie and Donut Charts - Visualise Categorical Data: Content Ratings

```
ratings = df_apps_clean.Content_Rating.value_counts()
ratings
```

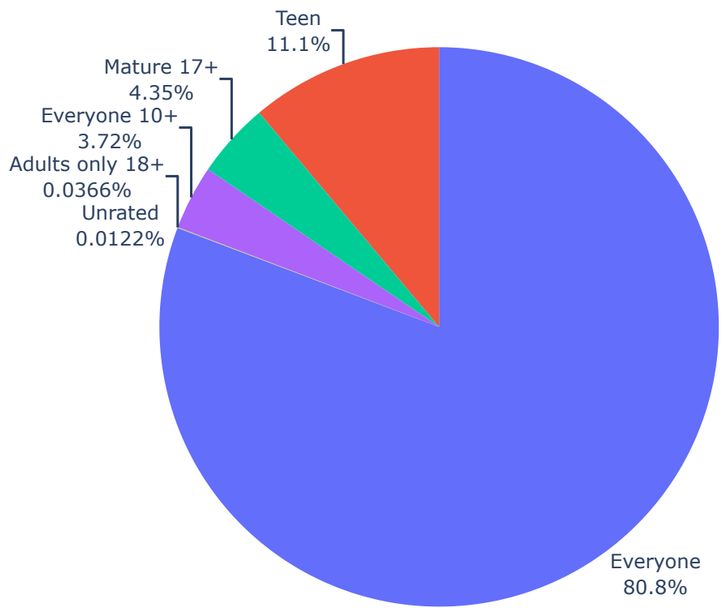
```
Everyone           6621
Teen                912
Mature 17+          357
Everyone 10+        305
Adults only 18+       3
Unrated               1
Name: Content_Rating, dtype: int64
```

```
fig = px.pie(labels=ratings.index, values=ratings.values)
fig.show()
```



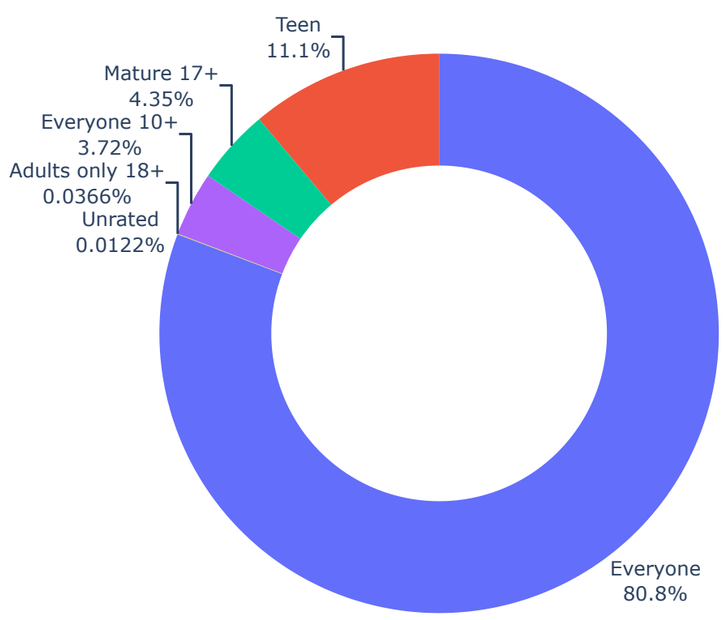

Customizing the pie chart

```
fig = px.pie(labels=ratings.index,
            values=ratings.values,
            title="Content Rating",
            names=ratings.index,
            )
fig.update_traces(textposition='outside', textinfo='percent+label')
fig.show()
```




Creating a donut chart

```
fig = px.pie(labels=ratings.index,
             values=ratings.values,
             title="Content Rating",
             names=ratings.index,
             hole=0.6,
             )
fig.update_traces(textposition='outside', textinfo='percent+label')
fig.show()
```




## Numeric Type Conversion: Examining the Number of Installs

Inspecting how many apps had over 1 billion installations, and how many apps just had a single install.

Checking the datatype of the Installs column.

Counting the number of apps at each level of installations.

Converting the number of installations (the Installs column) to a numeric data type. This is a 2-step process. First removing non-numeric characters.

```
df_apps_clean.Installs.describe()
df_apps_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8199 entries, 21 to 10835
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   App             8199 non-null   object
 1   Category        8199 non-null   object
 2   Rating          8199 non-null   float64
 3   Reviews         8199 non-null   int64
 4   Size_MBs        8199 non-null   float64
 5   Installs        8199 non-null   object
 6   Type            8199 non-null   object
 7   Price           8199 non-null   object
 8   Content_Rating  8199 non-null   object
 9   Genres          8199 non-null   object
dtypes: float64(2), int64(1), object(7)
memory usage: 704.6+ KB
```

Counting number of entries per level of installation

```
df_apps_clean[['App','Installs']].groupby('Installs').count()
```

| Installs | App |
|---|---|
| **1** | 3 |
| **1,000** | 698 |
| **1,000,000** | 1417 |
| **1,000,000,000** | 20 |
| **10** | 69 |
| **10,000** | 988 |
| **10,000,000** | 933 |
| **100** | 303 |
| **100,000** | 1096 |
| **100,000,000** | 189 |
| **5** | 9 |
| **5,000** | 425 |
| **5,000,000** | 607 |

removing comma chracter as it leads to confusing results. Look up.

```
df_apps_clean.Installs = df_apps_clean.Installs.astype(str).str.replace(",","")
df_apps_clean.Installs = pd.to_numeric(df_apps_clean.Installs)
df_apps_clean[['App','Installs']].groupby('Installs').count()
```

| Installs | App |
|---|---|
| **1** | 3 |
| **5** | 9 |
| **10** | 69 |
| **50** | 56 |
| **100** | 303 |
| **500** | 199 |
| **1000** | 698 |
| **5000** | 425 |
| **10000** | 988 |
| **50000** | 457 |
| **100000** | 1096 |
| **500000** | 504 |
| **1000000** | 1417 |
| **5000000** | 607 |
| **10000000** | 933 |
| **50000000** | 202 |
| **100000000** | 189 |
| **500000000** | 24 |
| **1000000000** | 20 |

## Finding the Most Expensive Apps, Filter out the Junk, and Calculate a (ballpark) Sales Revenue Estimate

Examining the Price column more closely.

Converting the price column to numeric data. Then investigate the top 20 most expensive apps in the dataset.

Removing all apps that cost more than $250 from the `df_apps_clean` DataFrame.

Adding a column called 'Revenue_Estimate' to the DataFrame. This column should hold the price of the app times the number of installs. What are the top 10 highest grossing paid apps according to this estimate? Out of the top 10 highest grossing paid apps, how many are games?

```
df_apps_clean.Price.describe()
```

```
count     8199
unique      73
top          0
freq      7595
Name: Price, dtype: object
```

replacing the $ sign

```
df_apps_clean.Price = df_apps_clean.Price.astype(str).str.replace("$","")
df_apps_clean.Price = pd.to_numeric(df_apps_clean.Price)
df_apps_clean.sort_values('Price', ascending=False).head(20)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning:

The default value of regex will change from True to False in a future version. In addition, single char
```

| | App | Category | Rating | Reviews | Size_MBs | Installs | Type | Price | Content_Rating | |
|---|---|---|---|---|---|---|---|---|---|---|
| **3946** | I'm Rich - Trump Edition | LIFESTYLE | 3.60 | 275 | 7.30 | 10000 | Paid | 400.00 | Everyone | l |
| **2461** | I AM RICH PRO PLUS | FINANCE | 4.00 | 36 | 41.00 | 1000 | Paid | 399.99 | Everyone | I |
| **4606** | I Am Rich Premium | FINANCE | 4.10 | 1867 | 4.70 | 50000 | Paid | 399.99 | Everyone | I |
| **3145** | I am rich(premium) | FINANCE | 3.50 | 472 | 0.94 | 5000 | Paid | 399.99 | Everyone | I |
| **3554** | 💎 I'm rich | LIFESTYLE | 3.80 | 718 | 26.00 | 10000 | Paid | 399.99 | Everyone | l |
| **5765** | I am rich | LIFESTYLE | 3.80 | 3547 | 1.80 | 100000 | Paid | 399.99 | Everyone | l |
| **1946** | I am rich (Most expensive app) | FINANCE | 4.10 | 129 | 2.70 | 1000 | Paid | 399.99 | Teen | I |
| **2775** | I Am Rich Pro | FAMILY | 4.40 | 201 | 2.70 | 5000 | Paid | 399.99 | Everyone | Entert |

## The most expensive apps sub $250 The app I am Rich looks dubious as it appears over 15 times. . We will sort it out by removing all pass that are over 250$

most

```
df_apps_clean = df_apps_clean[df_apps_clean.Price < 250]
df_apps_clean.sort_values('Price', ascending=False).head(5)
```

| | App | Category | Rating | Reviews | Size_MBs | Installs | Type | Price | Content_Rating | Genres |
|---|---|---|---|---|---|---|---|---|---|---|
| **2281** | Vargo Anesthesia Mega App | MEDICAL | 4.60 | 92 | 32.00 | 1000 | Paid | 79.99 | Everyone | Medical |
| **1407** | LTC AS Legal | MEDICAL | 4.00 | 6 | 1.30 | 100 | Paid | 39.99 | Everyone | Medical |
| **2629** | I am Rich Person | LIFESTYLE | 4.20 | 134 | 1.80 | 1000 | Paid | 37.99 | Everyone | Lifestyle |

## Highest Grossing Paid Apps (ballpark estimate)

addin a column

```
df_apps_clean["Revenue_Estimate"] = df_apps_clean.Installs.mul(df_apps_clean.Price)
df_apps_clean.sort_values('Revenue_Estimate', ascending=False)[:10]
```

| | App | Category | Rating | Reviews | Size_MBs | Installs | Type | Price | Content_Rating | |
|---|---|---|---|---|---|---|---|---|---|---|
| **9220** | Minecraft | FAMILY | 4.50 | 2376564 | 19.00 | 10000000 | Paid | 6.99 | Everyone 10+ | Arca & A |
| **8825** | Hitman Sniper | GAME | 4.60 | 408292 | 29.00 | 10000000 | Paid | 0.99 | Mature 17+ | |
| **7151** | Grand Theft Auto: San Andreas | GAME | 4.40 | 348962 | 26.00 | 1000000 | Paid | 6.99 | Mature 17+ | |
| **7477** | Facetune - For Free | PHOTOGRAPHY | 4.40 | 49553 | 48.00 | 1000000 | Paid | 5.99 | Everyone | Phc |
| **7977** | Sleep as Android Unlock | LIFESTYLE | 4.50 | 23966 | 0.85 | 1000000 | Paid | 5.99 | Everyone | |
| **6594** | DraStic DS Emulator | GAME | 4.60 | 87766 | 12.00 | 1000000 | Paid | 4.99 | Everyone | |
| **6082** | Weather Live | WEATHER | 4.50 | 76593 | 4.75 | 500000 | Paid | 5.99 | Everyone | |
| **7954** | Bloons | FAMILY | 4.60 | 190086 | 94.00 | 1000000 | Paid | 2.99 | Everyone | |

# Plotly Bar Charts & Scatter Plots: Analysing App Categories

```
df_apps_clean.Category.unique()
```

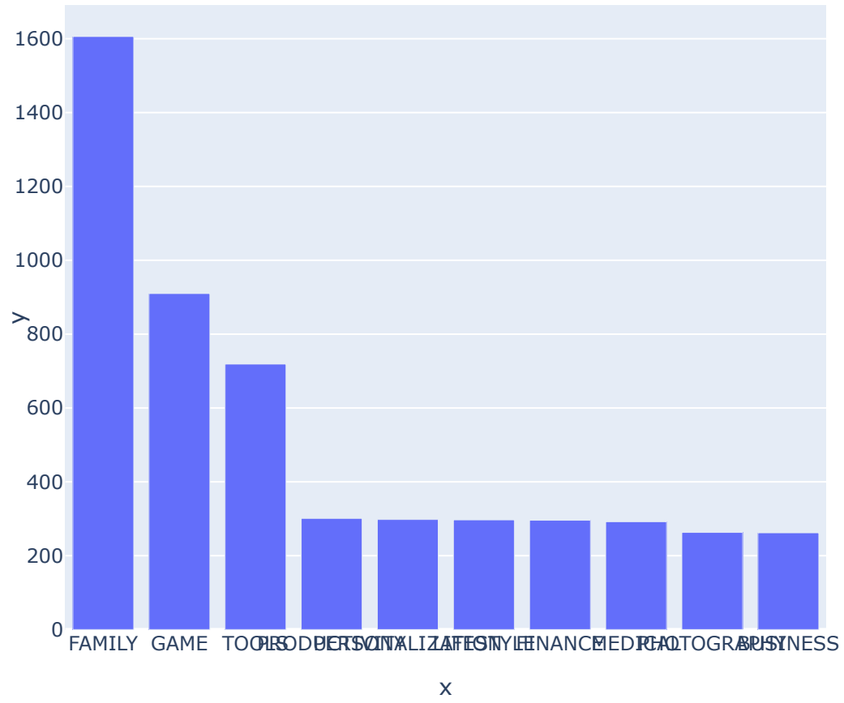
```
array(['MEDICAL', 'GAME', 'SPORTS', 'BUSINESS', 'BOOKS_AND_REFERENCE',
       'SOCIAL', 'TOOLS', 'FAMILY', 'COMMUNICATION', 'PRODUCTIVITY',
       'LIFESTYLE', 'DATING', 'EVENTS', 'MAPS_AND_NAVIGATION', 'SHOPPING',
       'PERSONALIZATION', 'PARENTING', 'PHOTOGRAPHY',
       'HEALTH_AND_FITNESS',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

```
top10_category = df_apps_clean.Category.value_counts()[:10]
top10_category
```

```
FAMILY             1606
GAME                910
TOOLS               719
PRODUCTIVITY        301
PERSONALIZATION     298
LIFESTYLE           297
FINANCE             296
MEDICAL             292
PHOTOGRAPHY         263
BUSINESS            262
Name: Category, dtype: int64
```

## Vertical Bar Chart - Highest Competition (Number of Apps)

```
bar = px.bar(x = top10_category.index, y = top10_category.values)
bar.show()
```

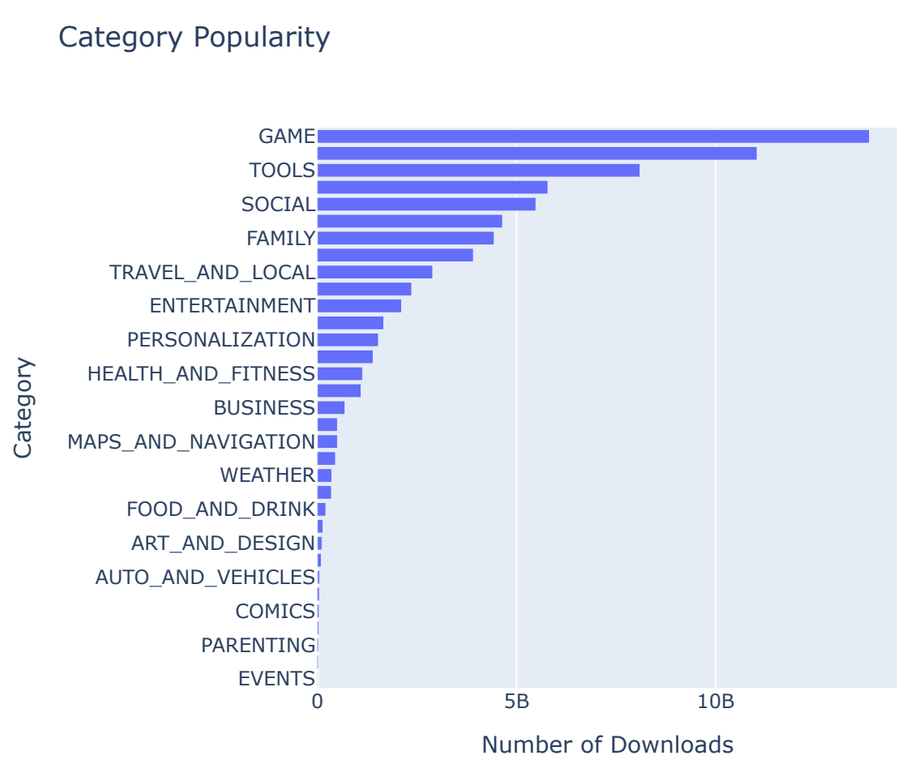



## Horizontal Bar Chart - Most Popular Categories (Highest Downloads)

by category - and how often all the apps from the given category were downloaded.

```
category_installs = df_apps_clean.groupby("Category").agg({'Installs': pd.Series.sum})
category_installs.sort_values("Installs", ascending=True, inplace=True)
```

```
h_bar = px.bar(x = category_installs.Installs,
                y = category_installs.index,
                orientation='h',
                title="Category Popularity",
                )
h_bar.update_layout(xaxis_title = "Number of Downloads",
                    yaxis_title = "Category")
h_bar.show()
```




## Category Concentration - Downloads vs. Competition

- First, we create a DataFrame that has the number of apps in one column and the number of installs in another:

```
cat_number = df_apps_clean.groupby("Category").agg({'App': pd.Series.count})
cat_merged_df = pd.merge(cat_number, category_installs,
                         on='Category', how='inner')
cat_merged_df.sort_values('Installs', ascending=False)[:10]
```

| Category | App | Installs |
|---|---|---|
| **GAME** | 910 | 13858762717 |
| **COMMUNICATION** | 257 | 11039241530 |
| **TOOLS** | 719 | 8099724500 |
| **PRODUCTIVITY** | 301 | 5788070180 |
| **SOCIAL** | 203 | 5487841475 |
| **PHOTOGRAPHY** | 263 | 4649143130 |
| **FAMILY** | 1606 | 4437554490 |
| **VIDEO_PLAYERS** | 148 | 3916897200 |
| **TRAVEL_AND_LOCAL** | 187 | 2894859300 |
| **NEWS_AND_MAGAZINES** | 204 | 2369110650 |

```
cat_merged_df = df_apps_clean.groupby("Category").agg({'App': pd.Series.count,
                                        'Installs': pd.Series.sum})
cat_merged_df.sort_values('Installs', ascending=False)[:10]
```

| Category | App | Installs |
|---|---|---|
| GAME | 910 | 13858762717 |
| COMMUNICATION | 257 | 11039241530 |
| TOOLS | 719 | 8099724500 |
| PRODUCTIVITY | 301 | 5788070180 |
| SOCIAL | 203 | 5487841475 |
| PHOTOGRAPHY | 263 | 4649143130 |
| FAMILY | 1606 | 4437554490 |
| VIDEO_PLAYERS | 148 | 3916897200 |
| TRAVEL_AND_LOCAL | 187 | 2894859300 |
| NEWS_AND_MAGAZINES | 204 | 2369110650 |

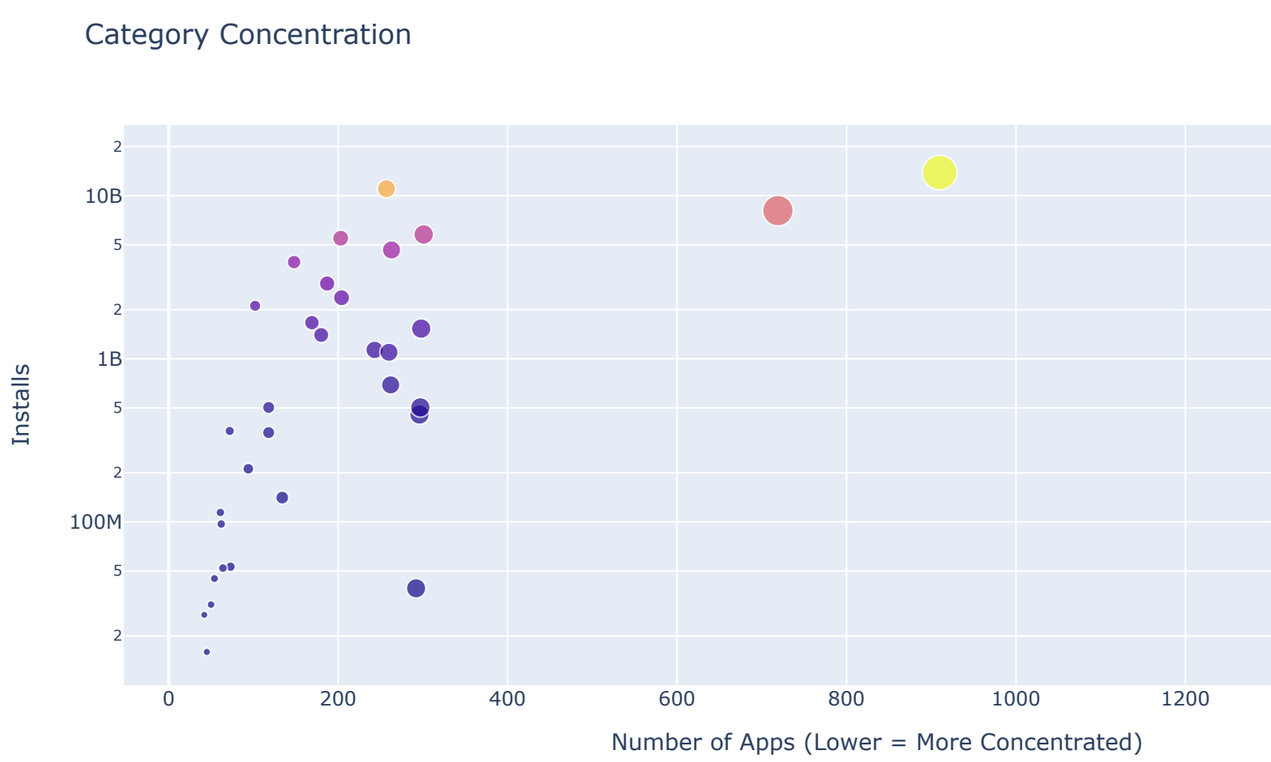
- Then we use the plotly express examples from the documentation alongside the .scatter() API referenceto create scatter plot.

Using the size, hover_name and color parameters in .scatter(). To scale the yaxis, call .update_layout() and specify that the yaxis should be on a log-scale like so: yaxis=dict(type='log')

```
scatter = px.scatter(
    cat_merged_df, #data
    x='App',
    y='Installs',
    title='Category Concentration',
    size='App',
    hover_name=cat_merged_df.index,
    color='Installs',
)

scatter.update_layout(
  xaxis_title='Number of Apps (Lower = More Concentrated)',
  yaxis_title='Installs',
  yaxis=dict(type='log'),
)

scatter.show()
```




## Extracting Nested Data from a Column

Checking how many different types of genres are there? An app can belong to more than one genre. Check what happens when you use .value_counts() on a column with nested values? Working around this problem by using the .split() function and the DataFrame's .stack() method.

```
# number of genres
len(df_apps_clean.Genres.unique())
```

```
114
```

```
# The problem is that we have multiple categories seperated by a semi-colon(;)
df_apps_clean.Genres.value_counts().sort_values(ascending=True)[:5]
```

```
Lifestyle;Pretend Play      1
Strategy;Education          1
Adventure;Education         1
Role Playing;Brain Games    1
Tools;Education             1
Name: Genres, dtype: int64
```

First we use split() method to seperate the genre names. and then with the method stack() we can add them all into one single column.

```
# split strings on semi-colon and then stack them
stack = df_apps_clean.Genres.str.split(";", expand=True).stack()
print(f"We now have a single column with shape: {stack.shape}")
num_genres = stack.value_counts()
print(f"Number of genres: {len(num_genres)}")
```

```
We now have a single column with shape: (8564,)
Number of genres: 53
```

## Colour Scales in Plotly Charts - Competition in Genres

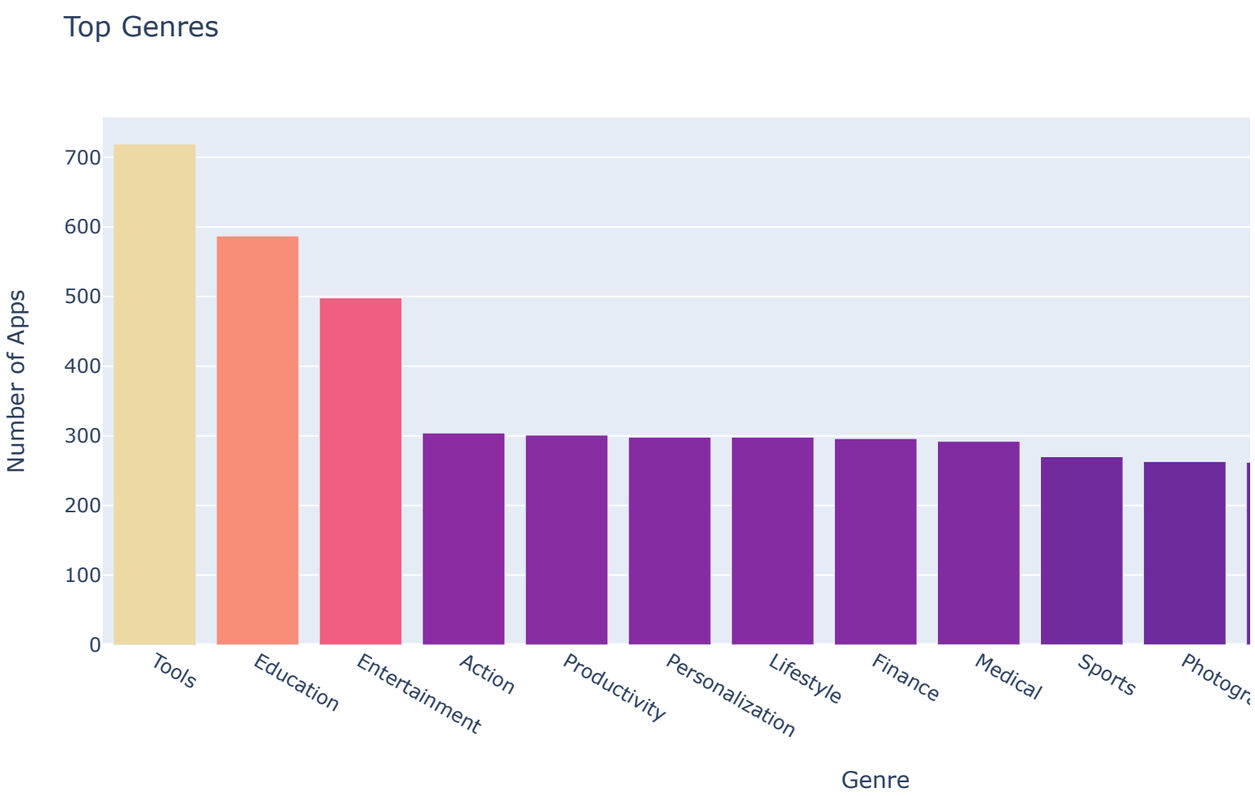
Experimenting with the built in colour scales in Plotly. We can find a full list here.

- setting the colour scale using the color_continuous_scale parameter.
- making the color axis disappear by using coloraxis_showscale.

```
bar = px.bar(
    x = num_genres.index[:15],
    y = num_genres.values[:15],
    title="Top Genres",
    hover_name=num_genres.index[:15],
    color=num_genres.values[:15],
    color_continuous_scale="Agsunset",
)

bar.update_layout(
    xaxis_title='Genre',
    yaxis_title='Number of Apps',
    coloraxis_showscale=False,
)

bar.show()
```




## Grouped Bar Charts: Free vs. Paid Apps per Category

```
df_apps_clean.Type.value_counts()
```

```
Free    7595
Paid     589
Name: Type, dtype: int64
```

```
df_free_vs_paid = df_apps_clean.groupby(
    ["Category", "Type"], as_index=False
    ).agg({'App': pd.Series.count})
df_free_vs_paid.head()
```

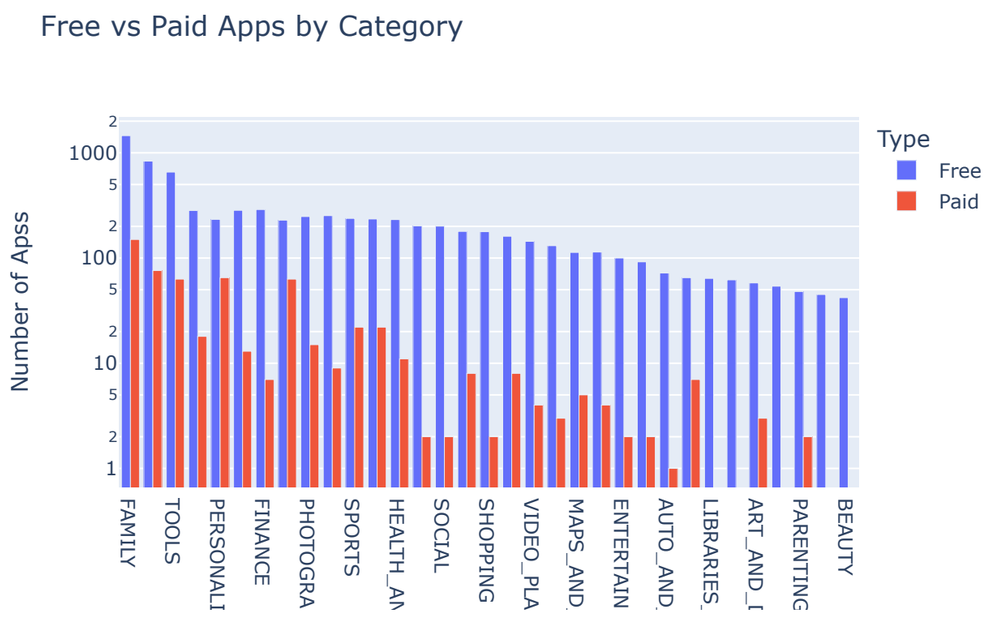
| | Category | Type | App |
|---|---|---|---|
| 0 | ART_AND_DESIGN | Free | 58 |
| 1 | ART_AND_DESIGN | Paid | 3 |
| 2 | AUTO_AND_VEHICLES | Free | 72 |
| 3 | AUTO_AND_VEHICLES | Paid | 1 |
| 4 | BEAUTY | Free | 42 |

Using the plotly express bar chart examples and the .bar() API reference to create a bar chart. Adjusting the look of the bar by changing the `categoryorder` to 'total descending' as outlined in the documentation here here.

```
g_bar = px.bar(
    df_free_vs_paid,
    x='Category',
    y='App',
    title='Free vs Paid Apps by Category',
    color='Type',
    barmode='group',
)

g_bar.update_layout(
    xaxis_title='Category',
    yaxis_title='Number of Apss',
    xaxis={'categoryorder': 'total descending'},
    yaxis=dict(type='log')
)

g_bar.show()
```




## Plotly Box Plots: Lost Downloads for Paid Apps

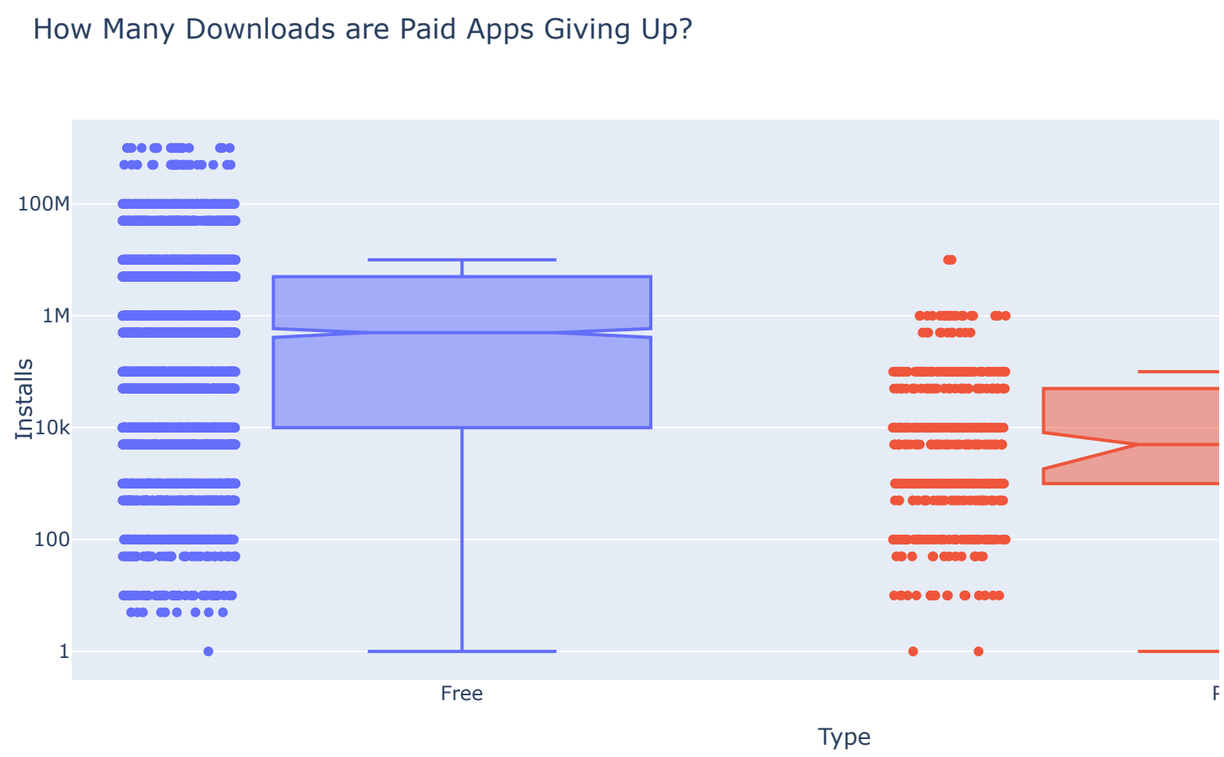
Creating a box plot that shows the number of Installs for free versus paid apps. It shows how does the median number of installations compare.

Using the [Box Plots Guide]() and the [.box API reference]() to create the box plot chart.

```
box = px.box(
    df_apps_clean,
    y='Installs',
    x='Type',
    color='Type',
    notched=True,
    points='all',
    title='How Many Downloads are Paid Apps Giving Up?',
)

box.update_layout(yaxis=dict(type='log'))

box.show()
```




## Plotly Box Plots: Revenue by App Category

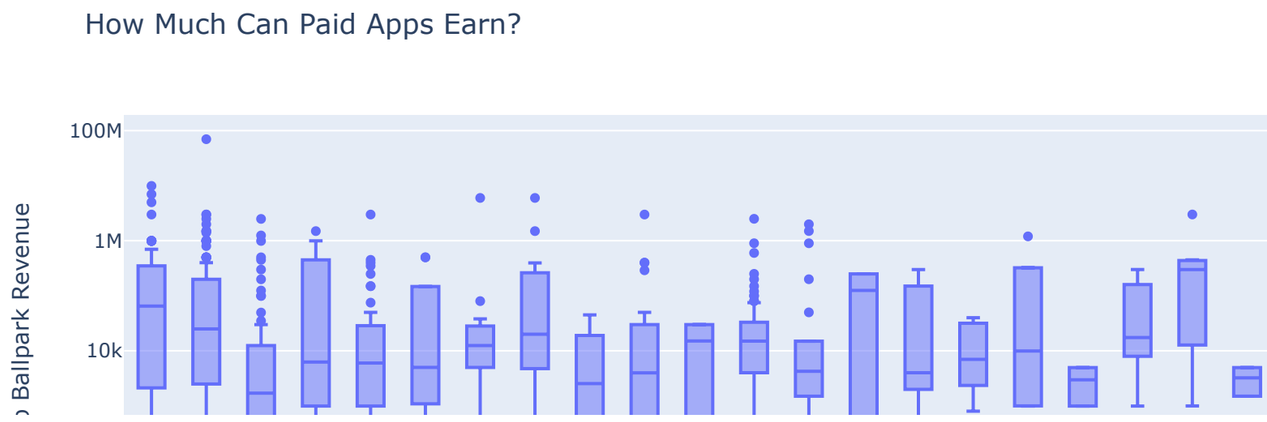
Looking at the hover text, how much does the median app earn in the Tools category? If developing an Android app costs $30,000 or thereabouts, does the average photography app recoup its development costs?

'min ascending' was used to sort the categories.

```
df_paid_apps = df_apps_clean[df_apps_clean['Type']=='Paid']
box = px.box(
    df_paid_apps,
    x='Category',
    y='Revenue_Estimate',
    title='How Much Can Paid Apps Earn?'
)

box.update_layout(
    xaxis_title='Category',
    yaxis_title='Paid App Ballpark Revenue',
    xaxis={'categoryorder': 'min ascending'},
    yaxis=dict(type='log')
)

box.show()
```




## How Much Can You Charge? Examine Paid App Pricing Strategies by Category

Showing the median price for a paid app. Then comparing pricing by category by creating another box plot. But this time we examine the prices (instead of the revenue estimates) of the paid apps. The `{categoryorder':'max descending'}` was used to sort the categories.

```
df_paid_apps.Price.median()
```

```
2.99
```

```
box = px.box(
    df_paid_apps,
    x='Category',
    y='Price',
    title='Price per Category'
)

box.update_layout(
    xaxis_title='Category',
    yaxis_title='Paid App Price',
    xaxis={'categoryorder': 'max descending'},
    yaxis=dict(type='log')
)

box.show()
```


Colab paid products - Cancel contracts here

✓ 0s completed at 7:59 PM