1. Google Play Store apps and reviews

Mobile apps are everywhere. They are easy to create and can be lucrative. Because of these two factors, more and more apps are being developed. In this notebook, we will do a comprehensive analysis of the Android app market by comparing over ten thousand apps in Google Play across different categories. We'll look for insights in the data to devise strategies to drive growth and retention.

GET IT ON Google Play

Let's take a look at the data, which consists of two files:

- `apps.csv` : contains all the details of the applications on Google Play. There are 13 features that describe a given app.
- `user_reviews.csv` : contains 100 reviews for each app, **most helpful first**. The text in each review has been pre-processed and attributed with three new features: Sentiment (Positive, Negative or Neutral), Sentiment Polarity and Sentiment Subjectivity.

```
# Step 1: Import pandas library
import pandas as pd

# Step 2: Load the dataset into a DataFrame
apps_with_duplicates = pd.read_csv('datasets/apps.csv')

# Step 3: Drop duplicate rows and save the result into 'apps'
apps = apps_with_duplicates.drop_duplicates()

# Step 4: Print the total number of apps
print("Total number of apps in the dataset:", len(apps))

# Step 5: Display a random sample of 5 rows from 'apps'
print(apps.sample(5))
```

```
Total number of apps in the dataset: 9659
      Unnamed: 0  ...         Android Ver
2999        3758  ...          4.1 and up
6351        7398  ...          5.0 and up
8552        9694  ...          4.3 and up
811         1022  ...  Varies with device
7374        8468  ...        4.0.3 and up

[5 rows x 14 columns]
```

2. Data cleaning

Data cleaning is one of the most essential subtask any data science project. Although it can be a very tedious process, it's worth should never be undermined.

By looking at a random sample of the dataset rows (from the above task), we observe that some entries in the columns like `Installs` and `Price` have a few special characters (`+` `,` `$`) due to the way the numbers have been represented. This prevents the columns from being purely numeric, making it difficult to use them in subsequent future mathematical calculations. Ideally, as their names suggest, we would want these columns to contain only digits from [0-9].

Hence, we now proceed to clean our data. Specifically, the special characters `,` and `+` present in `Installs` column and `$` present in `Price` column need to be removed.

It is also always a good practice to print a summary of your dataframe after completing data cleaning. We will use the `info()` method to acheive this.

```python
# Step 1: Create a list of characters to remove
chars_to_remove = ['+', ',', '$']

# Step 2: Create a list of column names to clean
cols_to_clean = ['Installs', 'Price']

# Step 3: Loop over each column in 'cols_to_clean' and remove unwanted characters
for col in cols_to_clean:
    for char in chars_to_remove:
        apps[col] = apps[col].apply(lambda x: x.replace(char, '') if isinstance(x, str) else x)

# Step 4: Print a summary of the apps dataframe
print(apps.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9659 entries, 0 to 9658
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Unnamed: 0      9659 non-null   int64  
 1   App             9659 non-null   object 
 2   Category        9659 non-null   object 
 3   Rating          8196 non-null   float64
 4   Reviews         9659 non-null   int64  
 5   Size            8432 non-null   float64
 6   Installs        9659 non-null   object 
 7   Type            9659 non-null   object 
 8   Price           9659 non-null   object 
 9   Content Rating  9659 non-null   object 
 10  Genres          9659 non-null   object 
 11  Last Updated    9659 non-null   object 
 12  Current Ver     9651 non-null   object 
 13  Android Ver     9657 non-null   object 
dtypes: float64(2), int64(2), object(10)
memory usage: 1.1+ MB
None
```

3. Correcting data types

From the previous task we noticed that `Installs` and `Price` were categorized as `object` data type (and not `int` or `float`) as we would like. This is because these two columns originally had mixed input types: digits and special characters. To know more about Pandas data types, read **this**.

The four features that we will be working with most frequently henceforth are `Installs`, `Size`, `Rating` and `Price`. While `Size` and `Rating` are both `float` (i.e. purely numerical data types), we still need to work on `Installs` and `Price` to make them numeric.

```python
# Step 1: Convert 'Installs' column to float
apps['Installs'] = apps['Installs'].astype(float)
```

```python
# Step 2: Convert 'Price' column to float
apps['Price'] = apps['Price'].astype(float)

# Step 3: Verify the corrected data types
print(apps.dtypes)
```

```
Unnamed: 0          int64
App                object
Category           object
Rating            float64
Reviews             int64
Size              float64
Installs          float64
Type               object
Price             float64
Content Rating     object
Genres             object
Last Updated       object
Current Ver        object
Android Ver        object
dtype: object
```

4. Exploring app categories

With more than 1 billion active users in 190 countries around the world, Google Play continues to be an important distribution platform to build a global audience. For businesses to get their apps in front of users, it's important to make them more quickly and easily discoverable on Google Play. To improve the overall search experience, Google has introduced the concept of grouping apps into categories.

This brings us to the following questions:

- Which category has the highest share of (active) apps in the market?
- Is any specific category dominating the market?
- Which categories have the fewest number of apps?

We will see that there are `33` unique app categories present in our dataset. *Family* and *Game* apps have the highest market prevalence. Interestingly, *Tools*, *Business* and *Medical* apps are also at the top.

```python
# Step 1: Find the number of unique app categories
num_categories = apps['Category'].nunique()
print("Number of unique categories is:", num_categories)

# Step 2: Count the number of apps in each category
num_apps_in_category = apps['Category'].value_counts()

# Step 3: Sort categories by the number of apps in descending order
sorted_num_apps_in_category = 
num_apps_in_category.sort_values(ascending=False)

# Display the sorted results
print(sorted_num_apps_in_category)
```

```
Number of unique categories is: 33
FAMILY                 1832
GAME                    959
TOOLS                   827
BUSINESS                420
MEDICAL                 395
PERSONALIZATION         376
PRODUCTIVITY            374
LIFESTYLE               369
FINANCE                 345
SPORTS                  325
COMMUNICATION           315
HEALTH_AND_FITNESS      288
PHOTOGRAPHY             281
NEWS_AND_MAGAZINES      254
SOCIAL                  239
BOOKS_AND_REFERENCE     222
TRAVEL_AND_LOCAL        219
SHOPPING                202
DATING                  171
VIDEO_PLAYERS           163
MAPS_AND_NAVIGATION     131
EDUCATION               119
FOOD_AND_DRINK          112
ENTERTAINMENT           102
AUTO_AND_VEHICLES        85
```

5. Distribution of app ratings

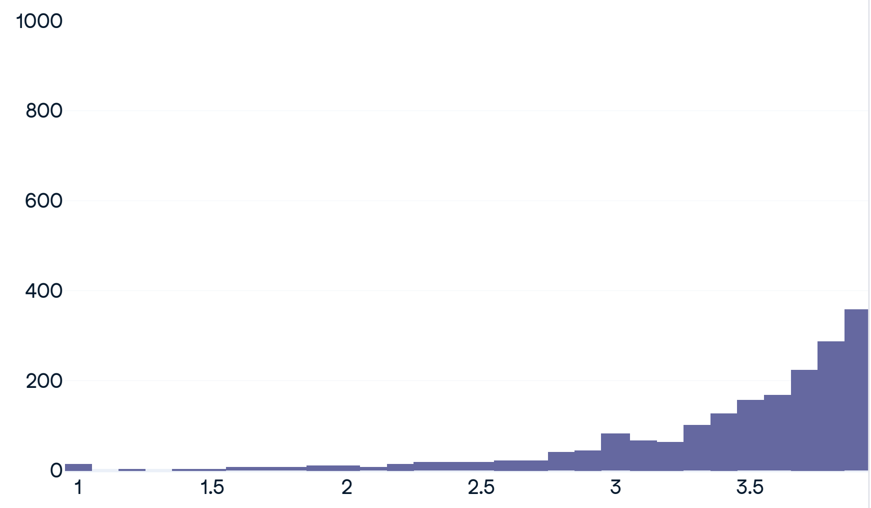
After having witnessed the market share for each category of apps, let's see how all these apps perform on an average. App ratings (on a scale of 1 to 5) impact the discoverability, conversion of apps as well as the company's overall brand image. Ratings are a key performance indicator of an app.

From our research, we found that the average volume of ratings across all app categories is `4.17`. The histogram plot is skewed to the left indicating that the majority of the apps are highly rated with only a few exceptions in the low-rated apps.

```
import plotly.graph_objs as go
import plotly.offline as pyo

# Step 1: Calculate the average app rating
avg_app_rating = apps['Rating'].mean()
print("Average app rating =", avg_app_rating)

# Step 2: Create a histogram for app ratings
data = [go.Histogram(x=apps['Rating'])]

# Step 3: Add a vertical dashed line for the average app rating
layout = {
    'shapes': [
        {
            'type': 'line',
            'x0': avg_app_rating,
            'x1': avg_app_rating,
            'y0': 0,
            'y1': 1000,  # Adjusted as per the data
            'line': {'dash': 'dashdot'}
        }
    ]
}
```

```
# Step 4: Combine the data and layout, and plot the figure
pyo.iplot({'data': data, 'layout': layout})
```

```
Average app rating = 4.173243045387994
```



6. Size and price of an app

Let's now examine app size and app price. For size, if the mobile app is too large, it may be difficult and/or expensive for users to download. Lengthy download times could turn users off before they even experience your mobile app. Plus, each user's device has a finite amount of disk space. For price, some users expect their apps to be free or inexpensive. These problems compound if the developing world is part of your target market; especially due to internet speeds, earning power and exchange rates.

How can we effectively come up with strategies to size and price our app?

- Does the size of an app affect its rating?
- Do users really care about system-heavy apps or do they prefer light-weighted apps?
- Does the price of an app affect its rating?
- Do users always prefer free apps over paid apps?

We find that the majority of top rated apps (rating over 4) range from 2 MB to 20 MB. We also find that the vast majority of apps price themselves under \$10.

```
import seaborn as sns
sns.set_style("darkgrid")
import warnings
warnings.filterwarnings("ignore")

# Step 1: Select rows where both 'Rating' and 'Size' are present (not null)
apps_with_size_and_rating_present = apps.dropna(subset=['Rating', 'Size'])

# Step 2: Subset for categories with at least 250 apps
large_categories =
apps_with_size_and_rating_present.groupby('Category').filter(lambda x: len(x) >= 250)
```

```python
# Step 3: Plot size vs. rating
plt1 = sns.jointplot(x='Size', y='Rating', data=large_categories)

# Step 4: Select apps whose 'Type' is 'Paid'
paid_apps = 
apps_with_size_and_rating_present[apps_with_size_and_rating_present['Type'] == 'Paid']

# Step 5: Plot price vs. rating
plt2 = sns.jointplot(x='Price', y='Rating', data=paid_apps)
```

7. Relation between app category and app price

So now comes the hard part. How are companies and developers supposed to make ends meet? What monetization strategies can companies use to maximize profit? The costs of apps are largely based on features, complexity, and platform.

There are many factors to consider when selecting the right pricing strategy for your mobile app. It is important to consider the willingness of your customer to pay for your app. A wrong price could break the deal before the download even happens. Potential customers could be turned off by what they perceive to be a shocking cost, or they might delete an app they've downloaded after receiving too many ads or simply not getting their money's worth.

Different categories demand different price ranges. Some apps that are simple and used daily, like the calculator app, should probably be kept free. However, it would make sense to charge for a highly-specialized medical app that diagnoses diabetic patients. Below, we see that *Medical and Family* apps are the most expensive. Some medical apps extend even up to \$80! All game apps are reasonably priced below \$20.

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

# Step 1: Filter apps by popular categories
popular_app_cats = apps[apps.Category.isin(['GAME', 'FAMILY',
'PHOTOGRAPHY',
                                            'MEDICAL', 'TOOLS',
'FINANCE',
                                            'LIFESTYLE', 'BUSINESS'])]

# Step 2: Examine price trends by category using strip plot
fig, ax = plt.subplots()
fig.set_size_inches(15, 8)
ax = sns.stripplot(x=popular_app_cats['Price'],
y=popular_app_cats['Category'],
                   jitter=True, linewidth=1)
```

```python
ax.set_title('App Pricing Trend Across Categories')

# Step 3: Filter apps where Price > $200 and print specific columns
apps_above_200 = apps[apps['Price'] > 200][['Category', 'App',
'Price']]
print(apps_above_200)
```

```
       Category                                 App   Price
3327     FAMILY              most expensive app (H)  399.99
3465  LIFESTYLE                          💎 I'm rich  399.99
3469  LIFESTYLE           I'm Rich - Trump Edition  400.00
4396  LIFESTYLE                           I am rich  399.99
4398     FAMILY                      I am Rich Plus  399.99
4399  LIFESTYLE                       I am rich VIP  299.99
4400    FINANCE                   I Am Rich Premium  399.99
4401  LIFESTYLE                 I am extremely Rich  379.99
4402    FINANCE                          I am Rich!  399.99
4403    FINANCE                  I am rich(premium)  399.99
4406     FAMILY                       I Am Rich Pro  399.99
4408    FINANCE      I am rich (Most expensive app)  399.99
4410     FAMILY                           I Am Rich  389.99
4413    FINANCE                           I am Rich  399.99
4417    FINANCE                  I AM RICH PRO PLUS  399.99
8763    FINANCE                         Eu Sou Rico  394.99
8780  LIFESTYLE  I'm Rich/Eu sou Rico/أنا غني/我很有錢   399.99
```

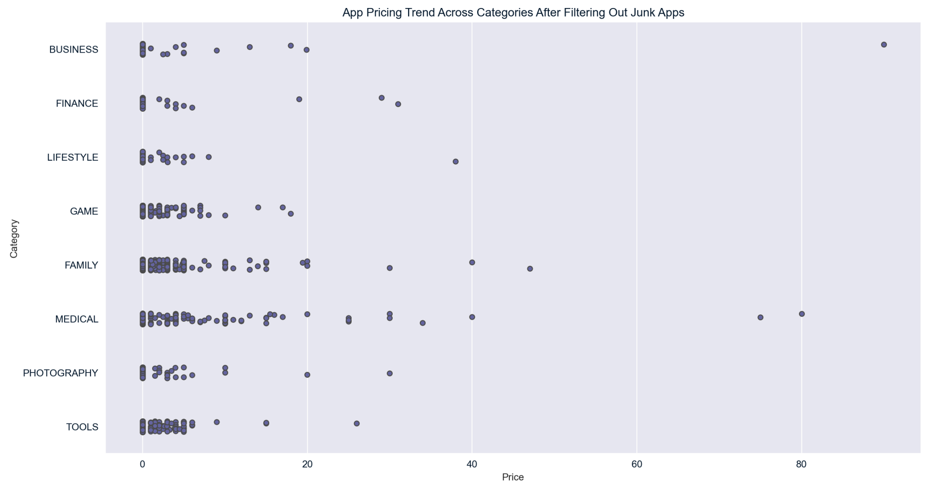
8. Filter out "junk" apps

It looks like a bunch of the really expensive apps are "junk" apps. That is, apps that don't really have a purpose. Some app developer may create an app called *I Am Rich Premium* or *most expensive app (H)* just for a joke or to test their app development skills. Some developers even do this with malicious intent and try to make money by hoping people accidentally click purchase on their app in the store.

Let's filter out these junk apps and re-do our visualization.

```python
# Step 1: Filter apps priced below $100
apps_under_100 = popular_app_cats[popular_app_cats['Price'] < 100]

# Step 2: Plot price trends for apps priced below $100
fig, ax = plt.subplots()
fig.set_size_inches(15, 8)
ax = sns.stripplot(x=apps_under_100['Price'],
y=apps_under_100['Category'],
                   jitter=True, linewidth=1)
ax.set_title('App Pricing Trend Across Categories After Filtering Out
Junk Apps')
```

```
plt.show()
```



9. Popularity of paid apps vs free apps

For apps in the Play Store today, there are five types of pricing strategies: free, freemium, paid, paymium, and subscription. Let's focus on free and paid apps only. Some characteristics of free apps are:

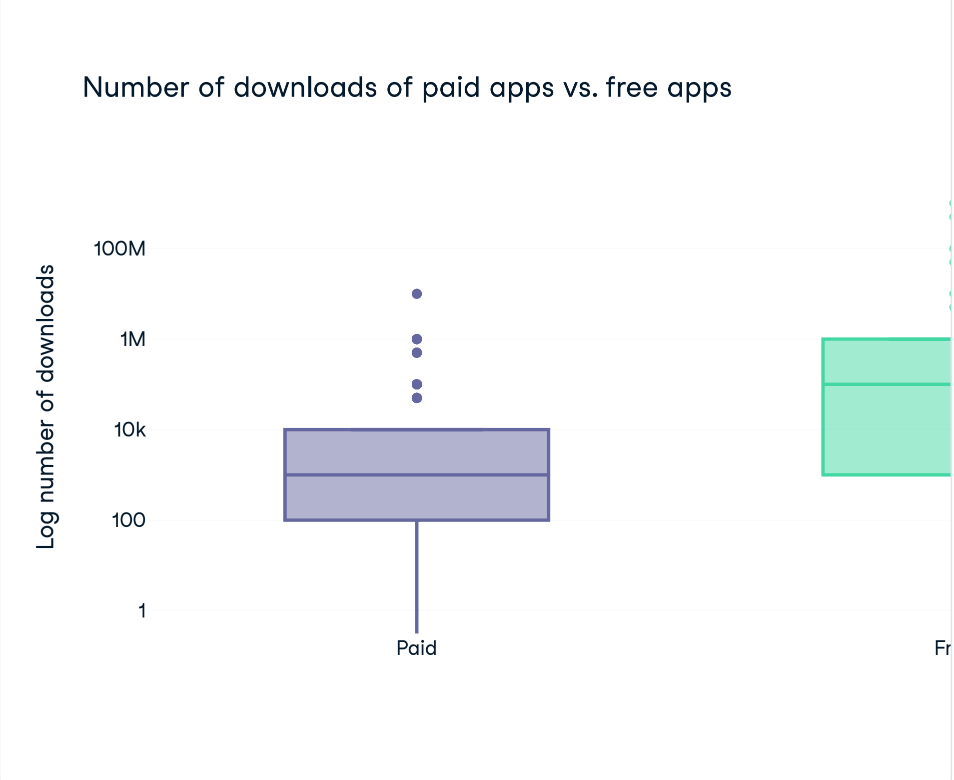
- Free to download.
- Main source of income often comes from advertisements.
- Often created by companies that have other products and the app serves as an extension of those products.
- Can serve as a tool for customer retention, communication, and customer service.

Some characteristics of paid apps are:

- Users are asked to pay once for the app to download and use it.
- The user can't really get a feel for the app before buying it.

Are paid apps installed as much as free apps? It turns out that paid apps have a relatively lower number of installs than free apps, though the difference is not as stark as I would have expected!

```
import plotly
import plotly.graph_objs as go

trace0 = go.Box(
    y = apps[apps['Type'] == 'Paid']['Installs'],
    name = 'Paid'
)

trace1 = go.Box(
    y = apps[apps['Type'] == 'Free']['Installs'],
    name = 'Free'
)

layout = go.Layout(
    title = "Number of downloads of paid apps vs. free apps",
    yaxis = dict(title = "Log number of downloads",
            type = 'log',
            autorange = True)
)

data = [trace0, trace1]
plotly.offline.init_notebook_mode(connected=True)
plotly.offline.iplot({'data': data, 'layout': layout})
```



10. Sentiment analysis of user reviews

Mining user review data to determine how people feel about your product, brand, or service can be done using a technique called sentiment analysis. User reviews for apps can be analyzed to identify if the mood is positive, negative or neutral about that app. For example, positive words in an app review might include words such as 'amazing', 'friendly', 'good', 'great', and 'love'. Negative words might be words like 'malware', 'hate', 'problem', 'refund', and 'incompetent'.

By plotting sentiment polarity scores of user reviews for paid and free apps, we observe that free apps receive a lot of harsh comments, as indicated by the outliers on the negative y-axis. Reviews for paid apps appear never to be extremely negative. This may indicate something about app quality, i.e., paid apps being of higher quality than free apps on average. The median polarity score for paid apps is a little higher than free apps, thereby syncing with our previous observation.

In this notebook, we analyzed over ten thousand apps from the Google Play Store. We can use our findings to inform our decisions should we ever wish to create an app ourselves.