Comprehensive Analysis Report

First, we load in the two datasets and import relevant packages. One dataset containing all sorts of information regarding the applications (e.g. Category, Num. of Installs, etc). The other contains written reviews regarding the applications.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
import re
from sklearn.preprocessing import RobustScaler, MinMaxScaler
from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np

# Load in the datasets
df = pd.read_csv('/content/drive/MyDrive/AndroidAppsProject/googleplaystore - googleplaystore.csv')
reviews = pd.read_csv('/content/drive/MyDrive/AndroidAppsProject/googleplaystore_user_reviews - googleplaystore_user_reviews.csv')
print(df.shape)
print(df.columns.tolist())
print(reviews.shape)
print(reviews.columns.tolist())
```

```
(10841, 13)
['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type', 'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current \
(64295, 5)
['App', 'Translated_Review', 'Sentiment', 'Sentiment_Polarity', 'Sentiment_Subjectivity']
```

Part I: Data Cleaning

I noticed that there were duplicate entries in both datasets, and hence first sought out to remove them

```
df = df.drop_duplicates()
reviews = reviews.drop_duplicates()
print(df.shape)
print(reviews.shape)
```

```
(10358, 13)
(30679, 5)
```

From the code below, we see that there are a lot of missing values in the "reviews" column. We drop the missing values, as those observations cannot be used.

```
print(reviews.isna().sum())
reviews.dropna(inplace=True)
print(reviews.shape)
```

```
App                         0
Translated_Review         987
Sentiment                 982
Sentiment_Polarity        982
Sentiment_Subjectivity    982
dtype: int64
(29692, 5)
```

Data regarding the number of installations in the 'Installs' column are stored as string literals. Using regular expressions, we can convert the column to contain only floats

```
def convert_installs(installs_str):
    cleaned_str = re.sub(r'\D', '', installs_str)
    return float(cleaned_str)

df['Installs'] = df['Installs'].apply(convert_installs)
```

Converted 'Last Updated' column from string literal to proper date-time format

```
df["Last Updated"] = pd.to_datetime(df["Last Updated"])
```

Part II: Feature Engineering

Calculating the average sentiment polarity and subjectivity of each App under reviews

```
reviews['average_polarity'] = reviews.groupby('App')['Sentiment_Polarity'].transform('mean')
reviews['average_subjectivity'] = reviews.groupby('App')['Sentiment_Subjectivity'].transform('mean')
```

Merge the two datasets together to bring over the average_polarity and average_subjectivity ratings

```
merged_df = pd.merge(df, reviews[['App', 'average_polarity', 'average_subjectivity']], on='App', how='left')
merged_df = merged_df.drop_duplicates()
print(merged_df.columns.to_list())
```

```
['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type', 'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current V
```

Added a new metric: **Objective_Positivity**. **The higher the polarity and the lower the subjectivity, the greater the objective_positivity score.** The reason for creating this metric is because it can be argued that a highly positive (indicative from a high polarity score) and objective (indicative from a low subjectivity score) review suggests that the App is of good quality.

```
merged_df['Objective_Positivity'] = (merged_df['average_polarity'] + (1 - merged_df['average_subjectivity'])) / 2
merged_df['Objective_Positivity'] = merged_df['Objective_Positivity'].fillna(0) # Some apps do not have written reviews
```

Two of the criterias for finding a good recommendation was popularity and novelty. It was suggested that an app that is popular but not too popular was ideal.

With this said, I created a new metric called Popularity_Score. It ranges from 0 to 1. Number of installations of an app that are closest to the median are given a score of 1, while numbers that are further from the median are given a score of 0.

```
median_installs = merged_df['Installs'].median()
max_installs = merged_df['Installs'].max()
print(median_installs, max_installs)
```

```
100000.0 1000000000.0
```

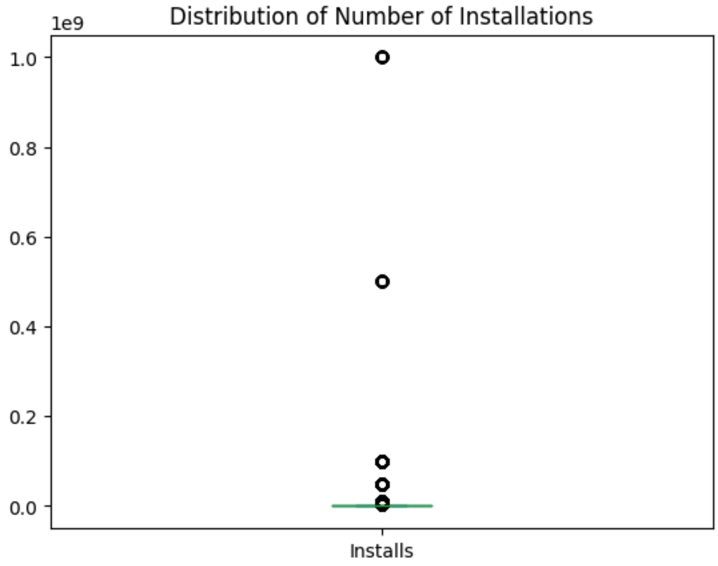
To materialize this metric, I first found the median installations and the highest number of installations. Then, the distance of the installation numbers from the median, scaled by the maximum number of installations was calculated via a function and that function is applied towards the **Installs** column to calculate the **Popularity_Score** metric.

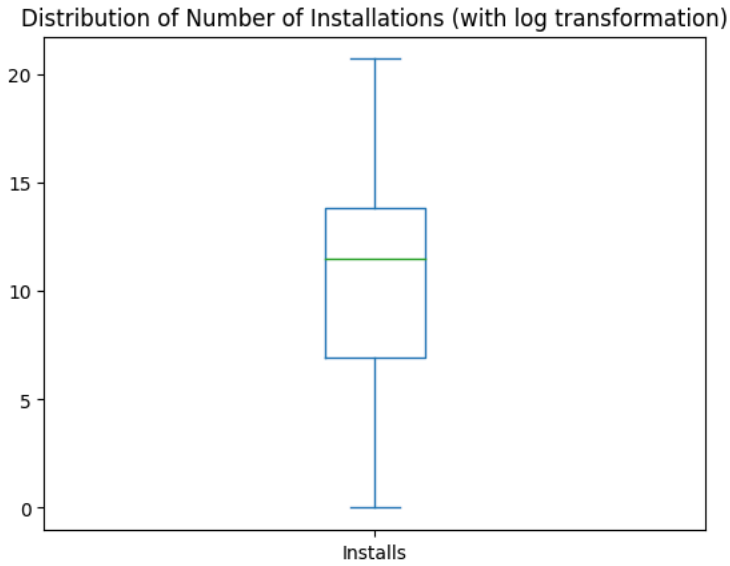
```
def popularity_score(installs):
    return 1 - (abs(installs - median_installs) / max_installs)

merged_df['Popularity_Score'] = merged_df['Installs'].apply(popularity_score)
```

> The below visualization is only for showcasing the diversity in the number of installations across all apps, thereby justifying the need to create a uniform popularity_score for ease of calculation further down the road

```
merged_df['Installs'].plot(kind='box')
plt.title("Distribution of Number of Installations")
plt.show()
np.log1p(merged_df['Installs']).plot(kind='box')
plt.title("Distribution of Number of Installations (with log transformation)")
plt.show()
```





Next, I converted **Ratings** from values ranging from 0 to 5, to values ranging from 0 to 1 for ease of calculation, through the use of MinMaxScaler.

```
min_max_scaler = MinMaxScaler()
merged_df['Ratings_Score'] = min_max_scaler.fit_transform(merged_df[['Rating']])
merged_df['Ratings_Score'] = merged_df['Ratings_Score'].fillna(0) # Some apps are unrated
```

It can be argued that the last time an application is updated could be an indicator of the app's relevancy. Therefore, I created a new metric called **Last_Update_Scaled**. This metric has a range from 0 to 1, with 1 being the closest date an app in the dataset has been updated and 0 being the further date an app in the dataset has been updated.

```
today = datetime.today()
merged_df['Days_Since_Last_Update'] = (today - merged_df["Last Updated"]).dt.days
merged_df['Last_Update_Scaled'] = (1 - min_max_scaler.fit_transform(merged_df[['Days_Since_Last_Update']]))
merged_df.drop(['Days_Since_Last_Update'], axis=1, inplace=True)
```

Part III: Obtaining Findings

I plan to create a list containing the Top 5 Apps with the highest overall score.

Recap - New metrics added include **Objective_Positivity**, **Popularity_Score**, **Ratings_Score**, and **Last_Update_Scaled**. We now find which App is most worthy to be included in the article by creating a new metric called **Overall Score**, which is the average of all the new metrics we created

```
merged_df["Overall Score"] = (merged_df['Objective_Positivity'] + merged_df['Popularity_Score'] + merged_df['Ratings_Score'] + merged_df[
```

I group all the apps into four broad categories using mapping. In each category, the app with the highest overall score makes it into the list.

```
category_mapping = {
    'ART_AND_DESIGN': 'Lifestyle & Social',
    'AUTO_AND_VEHICLES': 'Lifestyle & Social',
    'BEAUTY': 'Lifestyle & Social',
    'DATING': 'Lifestyle & Social',
    'EVENTS': 'Lifestyle & Social',
    'FOOD_AND_DRINK': 'Lifestyle & Social',
    'HEALTH_AND_FITNESS': 'Lifestyle & Social',
    'HOUSE_AND_HOME': 'Lifestyle & Social',
    'LIFESTYLE': 'Lifestyle & Social',
    'MEDICAL': 'Lifestyle & Social',
    'PARENTING': 'Lifestyle & Social',
    'SOCIAL': 'Lifestyle & Social',
    'SHOPPING': 'Lifestyle & Social',
    'SPORTS': 'Lifestyle & Social',
    'TRAVEL_AND_LOCAL': 'Lifestyle & Social',
    'WEATHER': 'Lifestyle & Social',
    'BUSINESS': 'Productivity & Tools',
    'FINANCE': 'Productivity & Tools',
    'LIBRARIES_AND_DEMO': 'Productivity & Tools',
    'TOOLS': 'Productivity & Tools',
    'PERSONALIZATION': 'Productivity & Tools',
    'PRODUCTIVITY': 'Productivity & Tools',
    'MAPS_AND_NAVIGATION': 'Productivity & Tools',
    'ENTERTAINMENT': 'Entertainment & Media',
    'COMICS': 'Entertainment & Media',
    'COMMUNICATION': 'Entertainment & Media',
    'FAMILY': 'Entertainment & Media',
    'GAME': 'Entertainment & Media',
    'PHOTOGRAPHY': 'Entertainment & Media',
    'VIDEO_PLAYERS': 'Entertainment & Media',
    'NEWS_AND_MAGAZINES': 'Entertainment & Media',
    'BOOKS_AND_REFERENCE': 'Education & Reference',
    'EDUCATION': 'Education & Reference'
}

merged_df['Broad Category'] = merged_df['Category'].map(category_mapping)
```

> Only included free apps as in general, paid apps have a lower overall score compared to free apps. However, I wanted to leave a space on the list for one paid app.

```
# Retrieve the top four free apps of each broad category\
free_apps = merged_df[merged_df['Type'] == "Free"]
idx = free_apps.groupby('Broad Category')['Overall Score'].idxmax()

highest_score_free_apps = free_apps.loc[idx]

pd.set_option('display.max_columns', None)
print(highest_score_free_apps)
```

```
                                      App            Category  Rating  \
5789                             HomeWork           EDUCATION     4.3
26614       Cameringo Lite. Filters Camera         PHOTOGRAPHY     4.2
9161   Home Workout for Men - Bodybuilding  HEALTH_AND_FITNESS     4.8
35689         GPS Speedometer and Odometer MAPS_AND_NAVIGATION     4.8

       Reviews  Size    Installs  Type Price Content Rating  \
5789     16195  5.2M   1000000.0  Free     0       Everyone
26614   140917  5.7M  10000000.0  Free     0       Everyone
9161     12705   15M   1000000.0  Free     0       Everyone
35689    15865  3.3M   1000000.0  Free     0       Everyone

                  Genres Last Updated Current Ver Android Ver  \
5789           Education   2016-09-20       8.5.2  4.0 and up
26614        Photography   2018-06-11      2.2.93  4.0 and up
9161    Health & Fitness   2018-07-10       1.0.2  4.0 and up
35689  Maps & Navigation   2018-08-03          10  4.1 and up

       average_polarity  average_subjectivity  Objective_Positivity  \
5789           1.000000              0.300000              0.850000
26614          0.770269              0.533333              0.618468
9161           0.504387              0.476908              0.513740
35689          0.650000              0.622222              0.513889

       Popularity_Score  Ratings_Score  Last_Update_Scaled  Overall Score  \
5789             0.9991          0.825            0.771076       0.861294
26614            0.9901          0.800            0.980673       0.847310
9161             0.9991          0.950            0.990337       0.863294
35689            0.9991          0.950            0.998334       0.865331

              Broad Category
```

```
5789   Education & Reference
26614  Entertainment & Media
9161      Lifestyle & Social
35689   Productivity & Tools
```

Lastly, the final spot of the list is given to a paid app with the highest overall score.

```
# Retrieve the top paid app
paid_apps = merged_df[merged_df['Type'] != "Free"]
idx = paid_apps['Overall Score'].idxmax()
highest_score_paid_app = paid_apps.loc[idx]
print(highest_score_paid_app)
```

```
App                        Diabetes & Diet Tracker
Category                                   MEDICAL
Rating                                         4.6
Reviews                                        395
Size                                           19M
Installs                                    1000.0
Type                                          Paid
Price                                        $9.99
Content Rating                            Everyone
Genres                                     Medical
Last Updated                   2018-07-16 00:00:00
Current Ver                                  6.5.1
Android Ver                             5.0 and up
average_polarity                          0.363627
average_subjectivity                      0.568469
Objective_Positivity                      0.397579
Popularity_Score                          0.999901
Ratings_Score                                  0.9
Last_Update_Scaled                        0.992336
Overall Score                             0.822454
Broad Category                  Lifestyle & Social
Name: 22789, dtype: object
```