Principal Component Analysis

Source:

https://www.kaggle.com/lava18/google-play-store-apps (https://www.kaggle.com/lava18/google-play-store-apps)

Defining the Problem Statement

This dataset records the attributes of Android mobile applications in the Google Play Store. From this dataset, we would like to be able to find the best clustering results/optimum number of clusters.

Collecting the Data

In [1]:

```
# Adding Required Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
from sklearn.cluster import KMeans
```

In [2]:

```
# Read in data from the file
df = pd.read_csv('googleps_cleaned.csv')
print(df.columns)
df.describe()
```

```
Index(['Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type', 'Pric
e',
       'Content Rating', 'Genres'],
      dtype='object')
```

Out[2]:

	Category	Reviews	Size	Installs	Type	Price	C
count	7723.000000	7.723000e+03	7723.00000	7723.000000	7723.000000	7723.000000	7723.(
mean	16.551599	2.948983e+05	37.30707	11.000129	0.074712	1.128169	1.4
std	8.128757	1.863933e+06	93.54223	3.213200	0.262943	17.408036	1.(
min	0.000000	1.000000e+00	1.00000	1.000000	0.000000	0.000000	0.(
25%	11.000000	1.075000e+02	6.10000	9.000000	0.000000	0.000000	1.(
50%	14.000000	2.332000e+03	16.00000	11.000000	0.000000	0.000000	1.(
75%	24.000000	3.905300e+04	37.00000	13.000000	0.000000	0.000000	1.(
max	32.000000	4.489389e+07	994.00000	19.000000	1.000000	400.000000	5.(

Scaling the Dataset

- Target is the 'Rating' column.

In [3]:

```
from sklearn.preprocessing import StandardScaler
features = ['Category', 'Reviews', 'Size', 'Installs', 'Type', 'Price', 'Content Ratin
g', 'Genres']

x = df.loc[:, features].values
y = df.loc[:,['Rating']].values
x = StandardScaler().fit_transform(x)

pd.DataFrame(data=x, columns = features).head()
```

Out[3]:

	Category	Reviews	Size	Installs	Type	Price	Content Rating	Genres
0	-2.03631	-0.158138	-0.195722	-0.622513	-0.284156	-0.064812	-0.469001	-1.590283
1	-2.03631	-0.157704	-0.249177	0.311196	-0.284156	-0.064812	-0.469001	-1.528596
2	-2.03631	-0.111271	-0.305840	0.933669	-0.284156	-0.064812	-0.469001	-1.590283
3	-2.03631	-0.042523	-0.131576	1.556141	-0.284156	-0.064812	2.499928	-1.590283
4	-2.03631	-0.157704	-0.368917	-0.000040	-0.284156	-0.064812	-0.469001	-1.559440

Performing 2D Principal Component Analysis

In [4]:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
principalDf.head()
```

Out[4]:

	principal component 1	principal component 2
0	2.289136	-0.859828
1	2.329817	-0.278366
2	2.431526	0.115043
3	3.235964	1.048576
4	2.325821	-0.468200

In [5]:

```
finalDf = pd.concat([principalDf, df[['Rating']]], axis = 1)
finalDf.head(5)
```

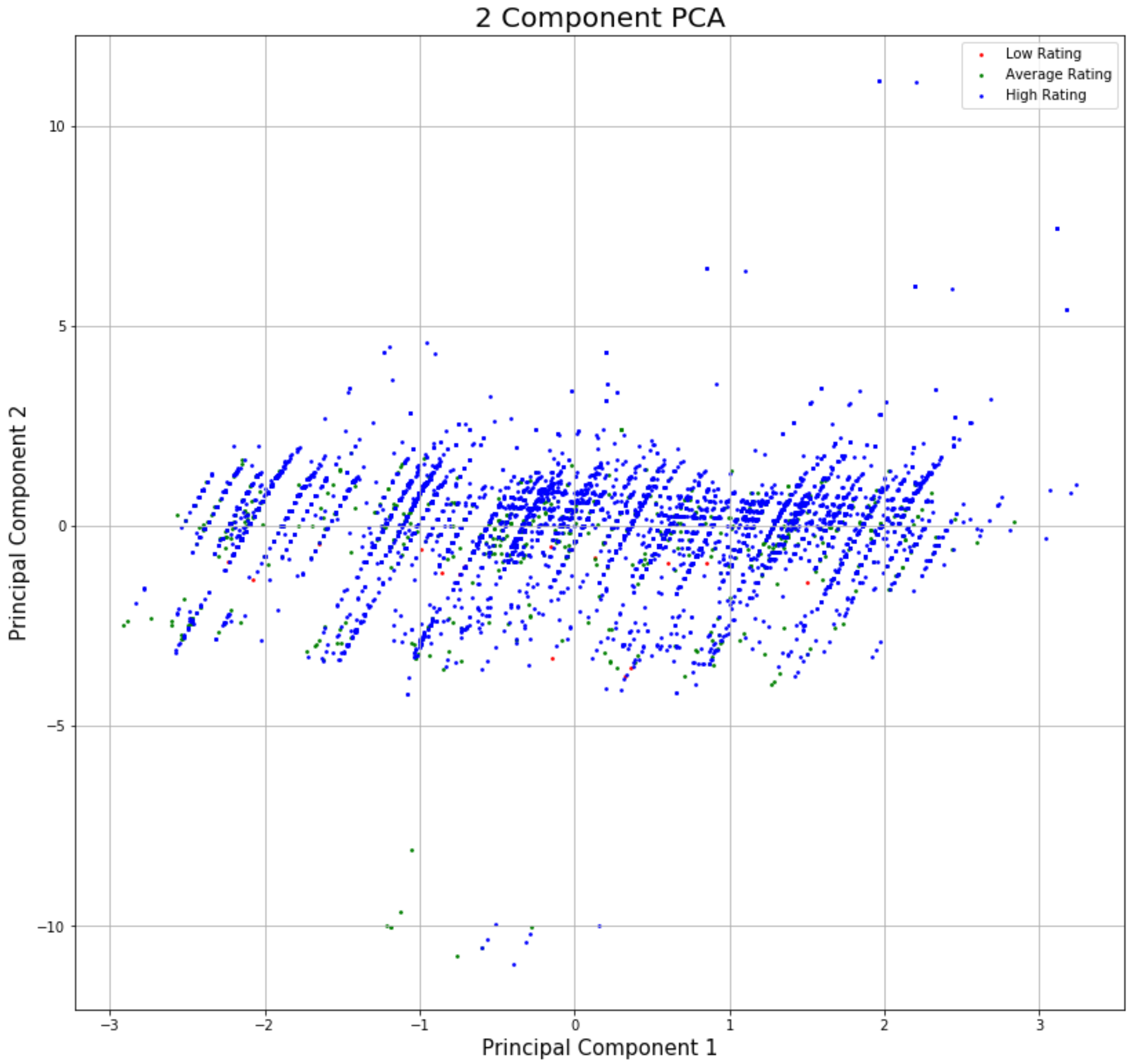
Out[5]:

	principal component 1	principal component 2	Rating
0	2.289136	-0.859828	High Rating
1	2.329817	-0.278366	Average Rating
2	2.431526	0.115043	High Rating
3	3.235964	1.048576	High Rating
4	2.325821	-0.468200	High Rating

In [6]:

```
fig = plt.figure(figsize = (30,13))
ax1 = fig.add_subplot(1,2,1)
ax1.set_xlabel('Principal Component 1', fontsize = 15)
ax1.set_ylabel('Principal Component 2', fontsize = 15)
ax1.set_title('2 Component PCA', fontsize = 20)

targets = ['Low Rating', 'Average Rating', 'High Rating']
colors = 'rgb'
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Rating'] == target
    ax1.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 3)
ax1.legend(targets)
ax1.grid()
```


2 Component PCA
Low Rating
Average Rating
High Rating
Principal Component 2
Principal Component 1
10
5
0
−5
−10
−3
−2
−1
0
1
2
3

In [7]:

```
print(pca.explained_variance_ratio_)

print(pca.explained_variance_ratio_.sum())
```

```
[0.22660842 0.18691077]
0.41351918839260504
```

Explained Variance

- The explained variance tells us how much information (variance) can be attributed to each of the principal components.
- Together, the first two principal components contain 41.35% of the information. The first principal component contains 22.66% of the variance and the second principal component contains 18.69% of the variance. The remaining principal components contained the rest of the variance of the dataset.

Fitting the PCA algorithm with our Data

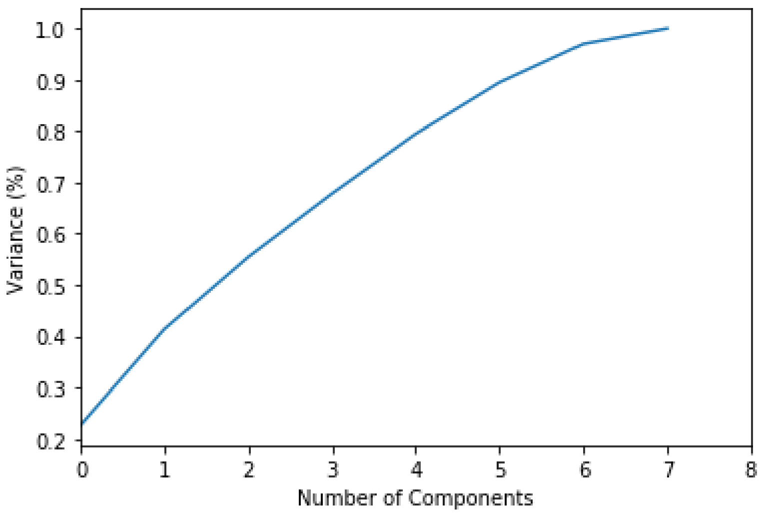
In [8]:

```
from sklearn.decomposition import PCA
pca = PCA().fit(x)
```

Plotting the Cumulative Summation of the Explained Variance

In [9]:

```
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0,8,1)
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.show()
```



The above plot shows almost 90% variance by the first 6 components. Therefore we can drop the last 2 components.

In [10]:

```
from pandas import DataFrame
DataFrame(pca.explained_variance_ratio_.round(2), index = ["P" + str(i) for i in range(
1,9)], columns=["Explained Variance Ratio"]).T
```

Out[10]:

	P1	P2	P3	P4	P5	P6	P7	P8
Explained Variance Ratio	0.23	0.19	0.14	0.12	0.12	0.1	0.08	0.03

The above information further verifies that our two-dimensional projection loses a lot of information (as measured by the explained variance of 41.35%) and that we need about 6 components to retain 90% of the variance.