youtubestreamersanalysis

May 21, 2024

1 YouTube Streamers Analysis

1.1 Importing the libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
df=pd.read_csv(r"C:\Users\AASTHA\Downloads\youtubers_df.csv")
df
```

[2]:
```
     Rank               Username            Categories   Suscribers  \
0       1                tseries         Música y baile  249500000.0
1       2                MrBeast      Videojuegos, Humor  183500000.0
2       3              CoComelon             Educación  165500000.0
3       4               SETIndia                   NaN  162600000.0
4       5           KidsDianaShow   Animación, Juguetes  113500000.0
..    …                     …                     …          …
995   996            hamzymukbang                   NaN   11700000.0
996   997             Adaahqueen                    NaN   11700000.0
997   998  LittleAngelIndonesia          Música y baile   11700000.0
998   999            PenMultiplex                   NaN   11700000.0
999  1000           OneindiaHindi  Noticias y Política   11700000.0

            Country       Visits      Likes  Comments  \
0             India      86200.0     2700.0      78.0
1    Estados Unidos  117400000.0  5300000.0   18500.0
2           Unknown    7000000.0    24700.0       0.0
3             India      15600.0      166.0       9.0
4           Unknown    3900000.0    12400.0       0.0
..              …            …          …        …
995  Estados Unidos     397400.0    14000.0     124.0
996           India    1100000.0    92500.0     164.0
997         Unknown     211400.0      745.0       0.0
998           India      14000.0       81.0       1.0
999           India       2200.0       31.0       1.0
```

```
                                                   Links
0    http://youtube.com/channel/UCq-Fj5jknLsUf-MWSy…
1    http://youtube.com/channel/UCX6OQ3DkcsbYNE6H8u…
2    http://youtube.com/channel/UCbCmjCuTUZos6Inko4…
3    http://youtube.com/channel/UCpEhnqL0y41EpW2TvW…
4    http://youtube.com/channel/UCk8GzjMOrta8yxDcKf…
..                                               …
995  http://youtube.com/channel/UCPKNKldggioffXPkSm…
996  http://youtube.com/channel/UCk3fFpqI5kDMf__mUP…
997  http://youtube.com/channel/UCdrHrQf0o0TO8YDntX…
998  http://youtube.com/channel/UCObyBrdrtQ20BU9PxH…
999  http://youtube.com/channel/UCOjgc1p2hJ4GZi6pQQ…

[1000 rows x 9 columns]
```

[3]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Rank        1000 non-null   int64
 1   Username    1000 non-null   object
 2   Categories  694 non-null    object
 3   Suscribers  1000 non-null   float64
 4   Country     1000 non-null   object
 5   Visits      1000 non-null   float64
 6   Likes       1000 non-null   float64
 7   Comments    1000 non-null   float64
 8   Links       1000 non-null   object
dtypes: float64(4), int64(1), object(4)
memory usage: 70.4+ KB
```

[4]:
```python
df.shape
```

[4]: (1000, 9)

[5]:
```python
df.describe()
```

[5]:

	Rank	Suscribers	Visits	Likes	Comments
count	1000.000000	1.000000e+03	1.000000e+03	1.000000e+03	1000.000000
mean	500.500000	2.189440e+07	1.209446e+06	5.363259e+04	1288.768000
std	288.819436	1.682775e+07	5.229942e+06	2.580457e+05	6778.188308
min	1.000000	1.170000e+07	0.000000e+00	0.000000e+00	0.000000
25%	250.750000	1.380000e+07	3.197500e+04	4.717500e+02	2.000000
50%	500.500000	1.675000e+07	1.744500e+05	3.500000e+03	67.000000

```
75%     750.250000  2.370000e+07  8.654750e+05  2.865000e+04     472.000000
max    1000.000000  2.495000e+08  1.174000e+08  5.300000e+06  154000.000000
```

[6]:
```python
df.isnull().sum()
```

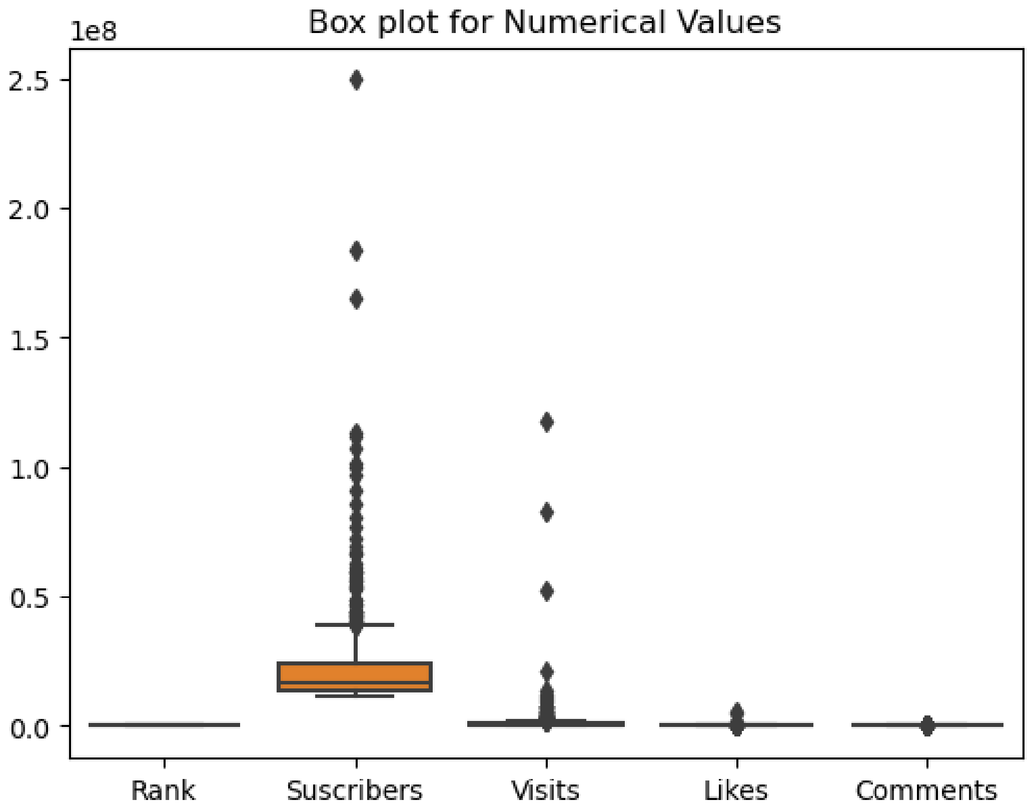
[6]:
```
Rank            0
Username        0
Categories    306
Suscribers      0
Country         0
Visits          0
Likes           0
Comments        0
Links           0
dtype: int64
```

[7]:
```python
#Drop Rows with missing values

df = df.dropna()
```

[8]:
```python
df.isnull().sum()
```

[8]:
```
Rank          0
Username      0
Categories    0
Suscribers    0
Country       0
Visits        0
Likes         0
Comments      0
Links         0
dtype: int64
```

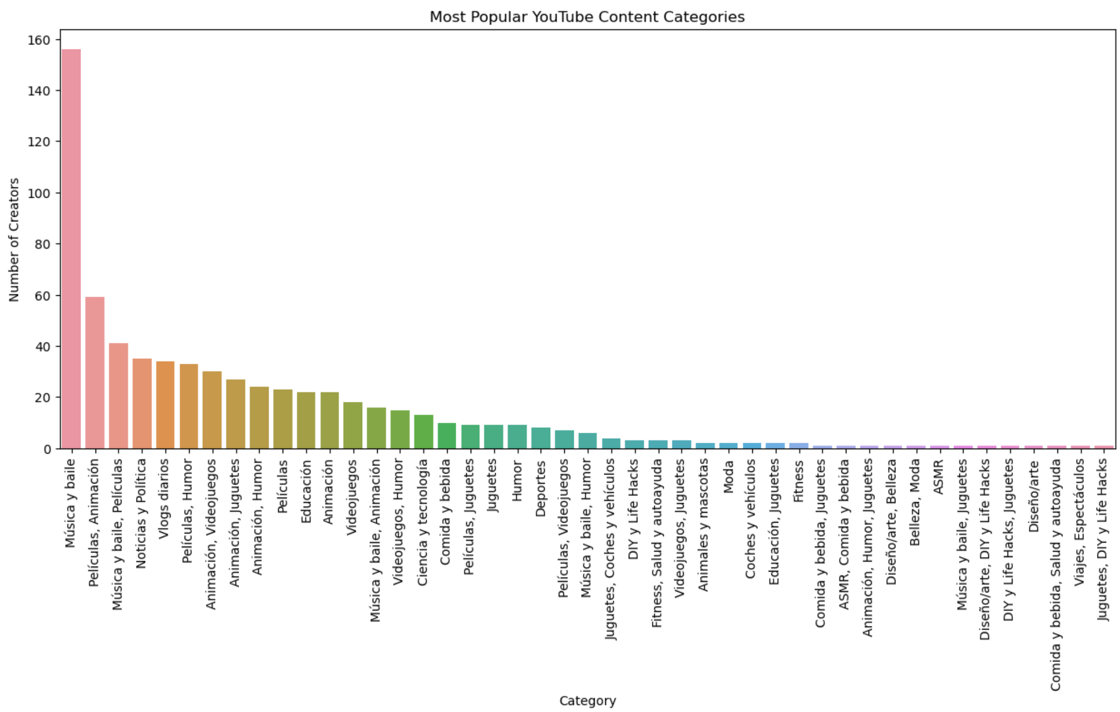
[9]:
```python
# Check for outliers using boxplots (For numerical values)

sns.boxplot(data=df)
plt.title("Box plot for Numerical Values")
plt.show()
```



[10]:
```
#Define a function to remove outliers using the z-score method for multiple␣
↪columns

from scipy.stats import zscore
def remove_outliers_zscore(dataframe, columns, threshold=3):
    df_no_outliers = dataframe.copy()
    for column in columns:
        z_scores = zscore(df_no_outliers[column])
        df_no_outliers = df_no_outliers[(z_scores < threshold) & (z_scores >␣
↪-threshold)]
    return df_no_outliers

#Choose columns to remove outliers from (e.g. Suscribers, Visits, Likes, Rank)

columns_to_clean = ['Suscribers', 'Visits', 'Likes', 'Rank']
df = remove_outliers_zscore(df, columns_to_clean)
```

[12]:
```
#To identify the most popular categories

category_counts = df['Categories'].value_counts()
plt.figure(figsize=(15,6))
```

```python
sns.barplot(x=category_counts.index, y=category_counts.values)
plt.xticks(rotation=90)
plt.title("Most Popular YouTube Content Categories")
plt.xlabel("Category")
plt.ylabel("Number of Creators")
plt.show()
```



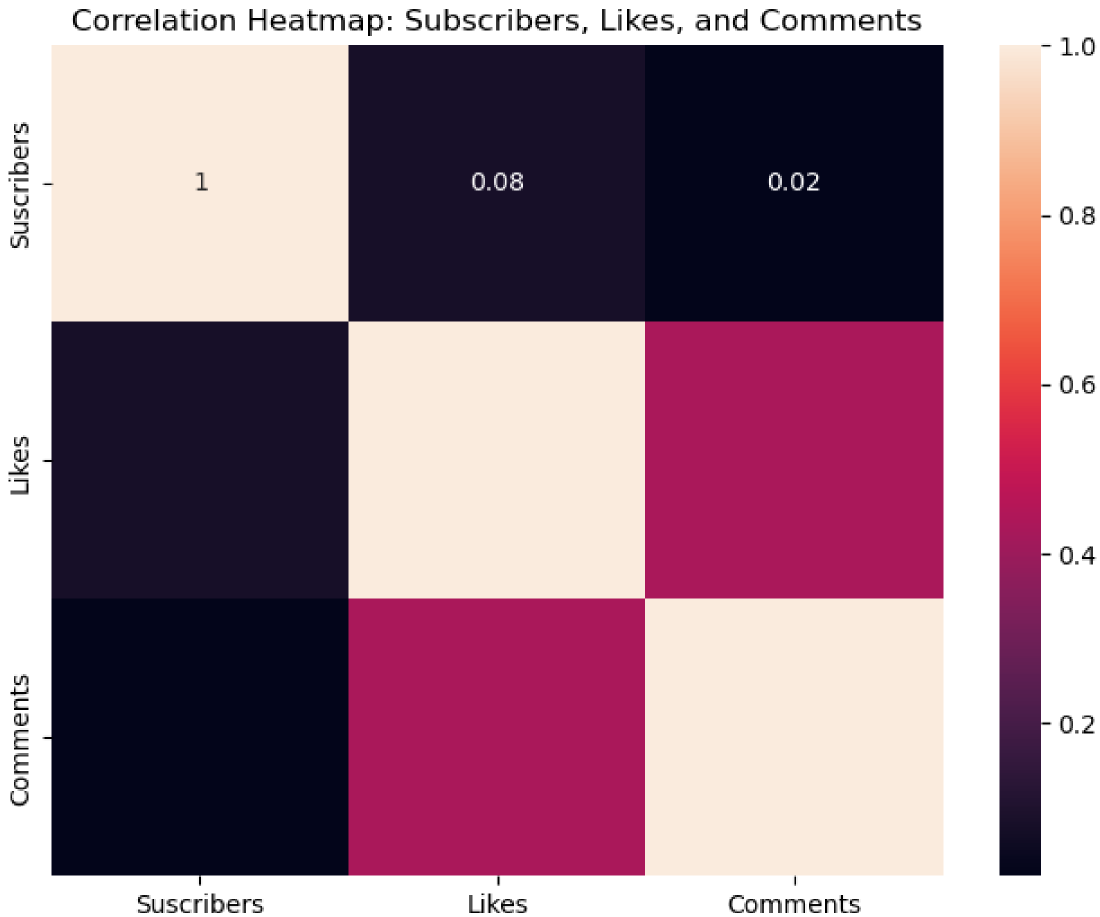
[14]:

```python
#The Correlation between Subscribers, Likes, and Comments

corr_likes = np.corrcoef(df['Suscribers'], df['Likes']) [0,1]
corr_comments = np.corrcoef(df['Suscribers'], df['Comments']) [0,1]
print(f'Correlation b/w Subscribers and Likes: {corr_likes}')
print(f'Correlation b/w Subscribers and Comments: {corr_comments}')
```

```
Correlation b/w Subscribers and Likes: 0.07958997301642448
Correlation b/w Subscribers and Comments: 0.019905291465977828
```

[18]:

```python
#The Correlation between Subscribers, Likes, and Comments

correlation = df[['Suscribers', 'Likes', 'Comments']].corr()
plt.figure(figsize=(8,6))
sns.heatmap(correlation, annot=True)
plt.title("Correlation Heatmap: Subscribers, Likes, and Comments")
plt.show()
```



1.2 Audience Study

[62]:
```
#To count the number of creators in each combination

category_country_counts = df.groupby(['Categories', 'Country'])['Categories'].
    ↪count().reset_index(name='Count')
category_country_counts
```

[62]:
```
                  Categories          Country  Count
0                       ASMR   Estados Unidos      1
1       ASMR, Comida y bebida  Estados Unidos      1
2                  Animación        Argentina      1
3                  Animación           Brasil      3
4                  Animación   Estados Unidos      4
..                         …                …      …
163            Vlogs diarios            India     12
164            Vlogs diarios        Indonesia      1
```

```
165          Vlogs diarios       Pakistán      1
166          Vlogs diarios        Turquía      2
167          Vlogs diarios        Unknown      7

[168 rows x 3 columns]
```

[63]:
```
country_visit_count = df.groupby('Country')['Visits'].sum().reset_index()
country_visit_count
```

[63]:
```
           Country       Visits
0   Arabia Saudita    3474500.0
1          Argelia     333500.0
2        Argentina    6371400.0
3       Bangladesh     100700.0
4           Brasil   18643600.0
5         Colombia    6256400.0
6           Egipto     305400.0
7           España    1984300.0
8   Estados Unidos  207221500.0
9        Filipinas    8914600.0
10         Francia    5308000.0
11           India   35783000.0
12       Indonesia   11556900.0
13            Iraq     103600.0
14           Japón    2100000.0
15         Jordania     267600.0
16       Marruecos      12000.0
17          México   31365000.0
18        Pakistán    1969600.0
19            Perú    2219400.0
20     Reino Unido    9250700.0
21           Rusia   23299600.0
22        Singapur      26400.0
23         Somalia    1900000.0
24       Tailandia    2553800.0
25         Turquía    1604900.0
26         Unknown   47211700.0
```

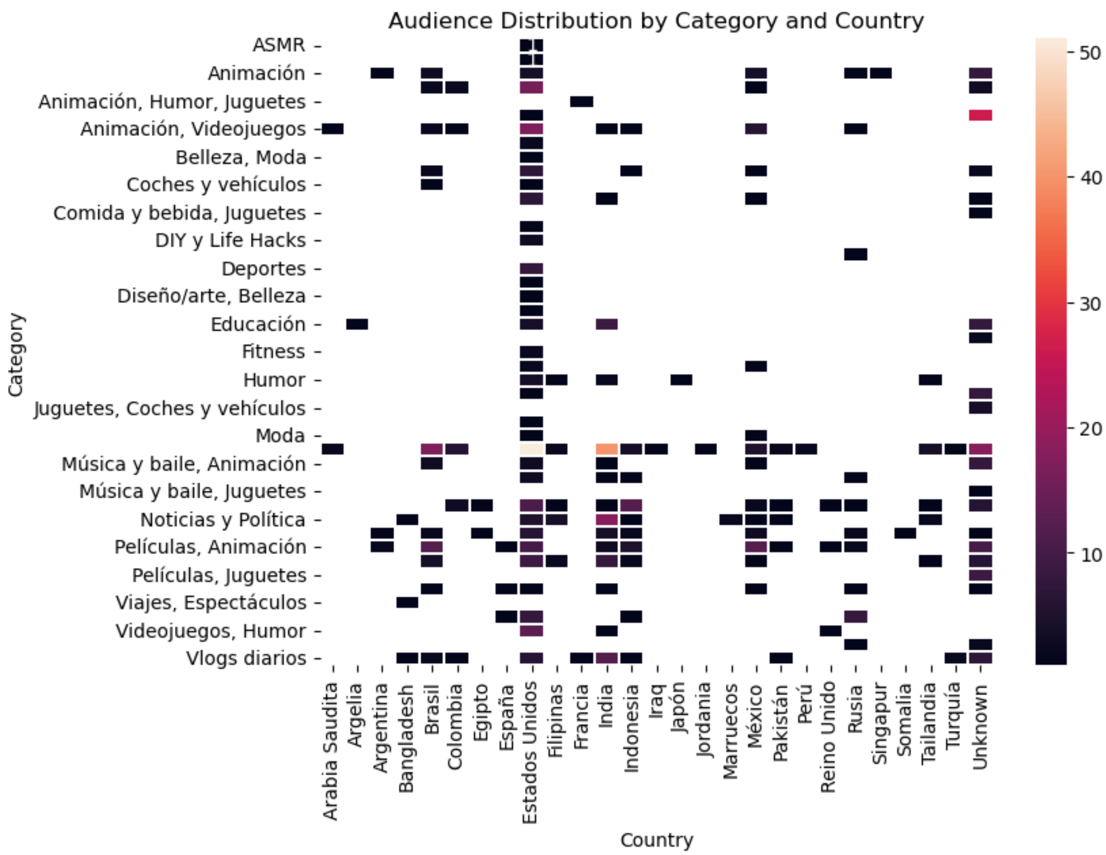
[70]:
```
#Create a pivot table for better visualization

pivot_table = category_country_counts.pivot(index='Categories',␣
  ↪columns='Country', values='Count')
```

[73]:
```
#Heatmap for streamers audiences by country and category

plt.figure(figsize=(8,6))
sns.heatmap(pivot_table, annot=True, linewidths=0.3)
```

```
plt.title("Audience Distribution by Category and Country")
plt.xlabel("Country")
plt.ylabel("Category")
```

```
C:\Users\AASTHA\anaconda3\Lib\site-packages\seaborn\matrix.py:260:
FutureWarning: Format strings passed to MaskedConstant are ignored, but in
future may error or produce different behavior
  annotation = ("{:" + self.fmt + "}").format(val)
```

[73]: Text(70.2222222222222, 0.5, 'Category')



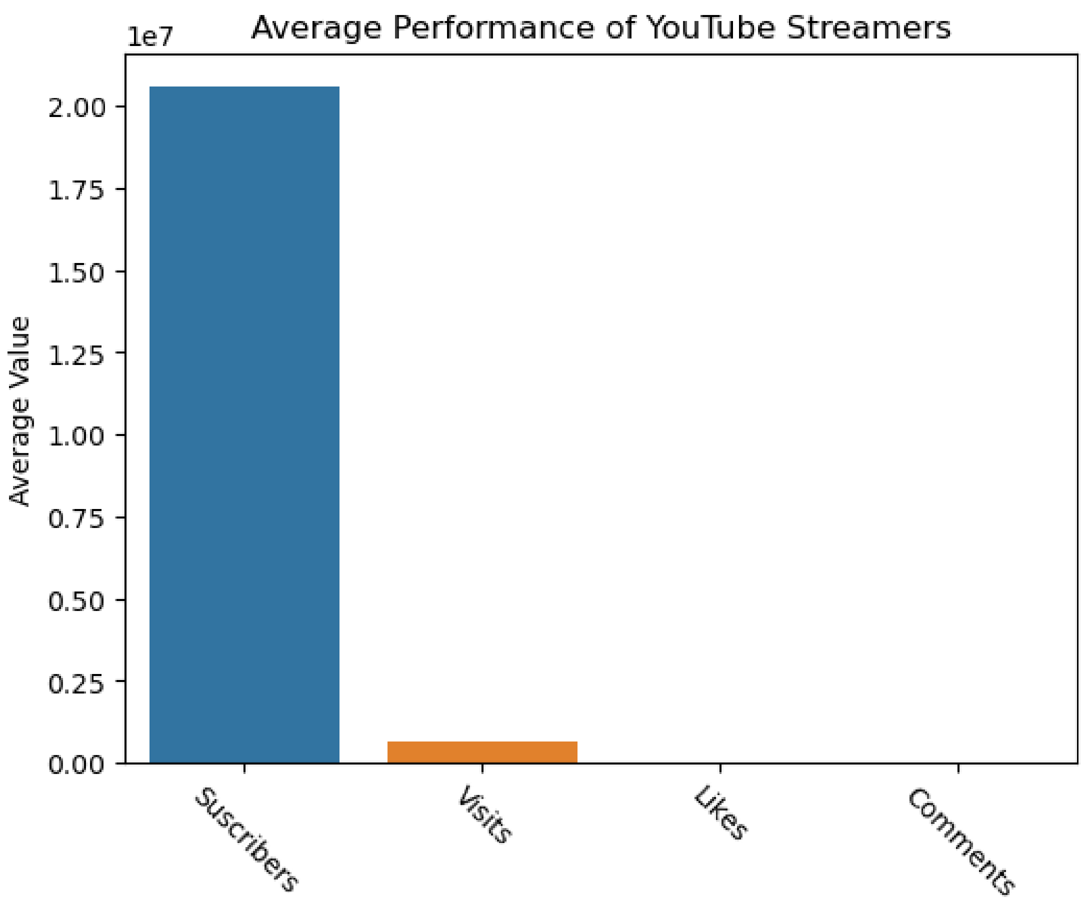
2 Performance Metrics

[74]:
```
# To caculate the average metrics

average_metrics= df[['Suscribers', 'Visits', 'Likes', 'Comments']].mean()
```

[75]:
```
#Barplot of average metrics
```

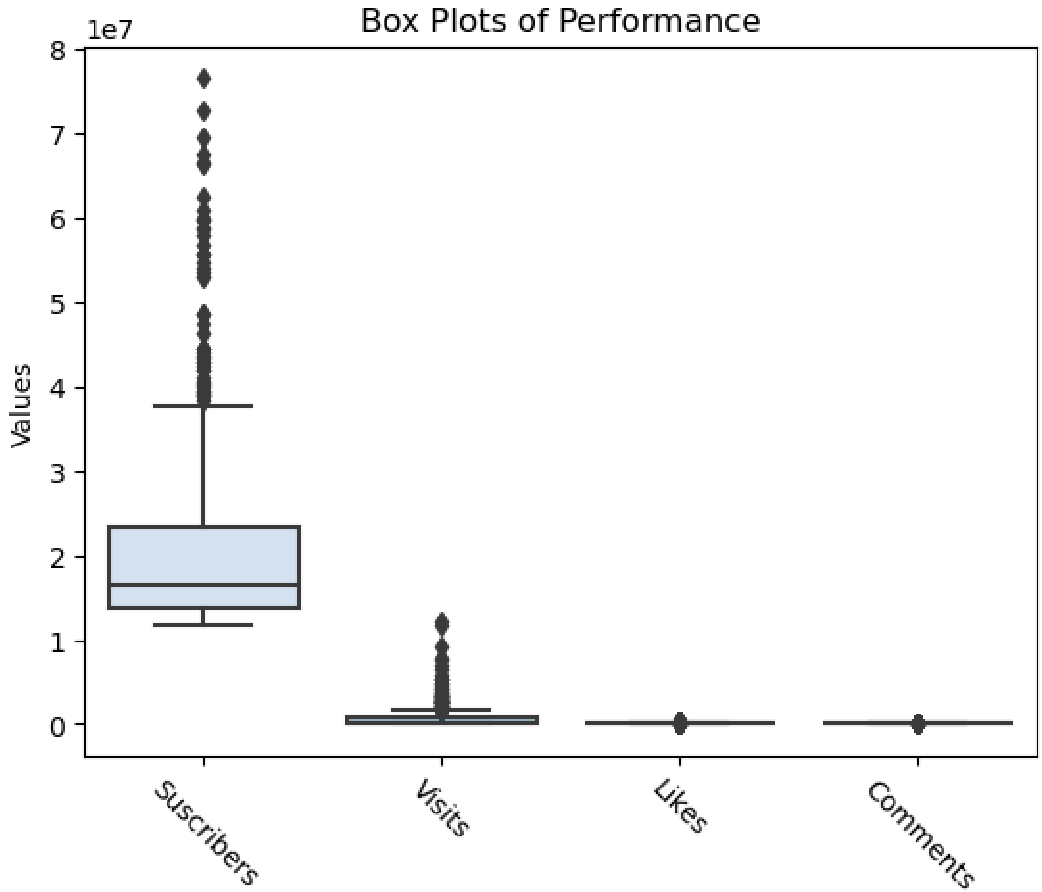
```
sns.barplot(x=average_metrics.index, y=average_metrics.values)
plt.title("Average Performance of YouTube Streamers")
plt.ylabel("Average Value")
plt.xticks(rotation=-45)
```

[75]:
```
(array([0, 1, 2, 3]),
 [Text(0, 0, 'Suscribers'),
  Text(1, 0, 'Visits'),
  Text(2, 0, 'Likes'),
  Text(3, 0, 'Comments')])
```



[79]:
```
#For patterns in the metrics

sns.boxplot(data=df[['Suscribers', 'Visits', 'Likes', 'Comments']],␣
 ↪palette="Blues")
plt.title("Box Plots of Performance")
plt.ylabel("Values")
plt.xticks(rotation=-45)
```

[79]: (array([0, 1, 2, 3]),
 [Text(0, 0, 'Suscribers'),
  Text(1, 0, 'Visits'),
  Text(2, 0, 'Likes'),
  Text(3, 0, 'Comments')])



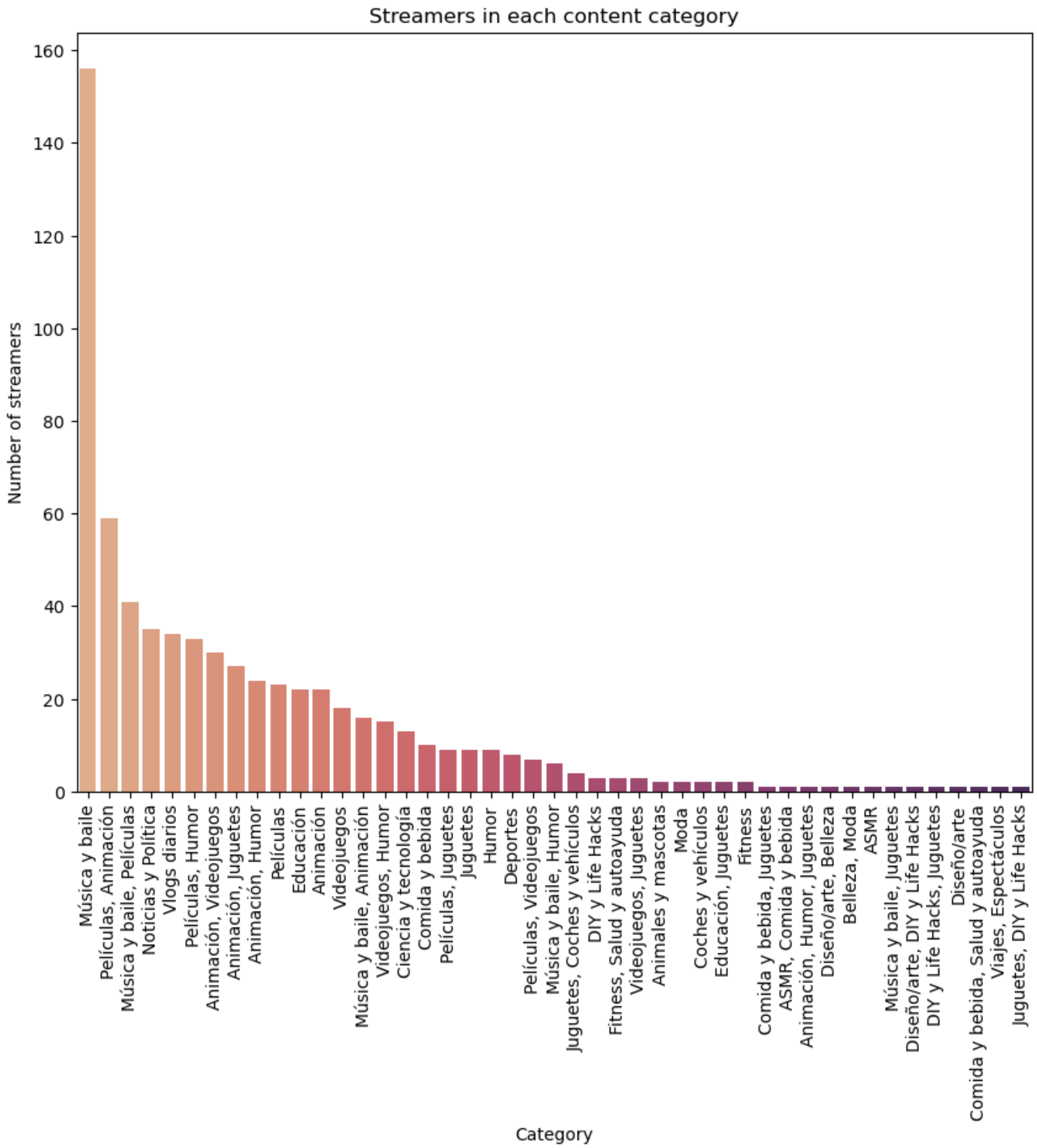
3 Distribution of content categories

[82]:
```
#To calculate category counts
category_counts = df['Categories'].value_counts()

#Create a bar plot to visualize the number of streamers in each category

plt.figure(figsize=(10,8))
sns.barplot(x=category_counts.index, y=category_counts.values, palette="flare")
plt.title("Streamers in each content category")
plt.xlabel("Category")
```

```
plt.ylabel("Number of streamers")
plt.xticks(rotation=90)
```

[82]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44]),
 [Text(0, 0, 'Música y baile'),
  Text(1, 0, 'Películas, Animación'),
  Text(2, 0, 'Música y baile, Películas'),
  Text(3, 0, 'Noticias y Política'),
  Text(4, 0, 'Vlogs diarios'),
  Text(5, 0, 'Películas, Humor'),
  Text(6, 0, 'Animación, Videojuegos'),
  Text(7, 0, 'Animación, Juguetes'),
  Text(8, 0, 'Animación, Humor'),
  Text(9, 0, 'Películas'),
  Text(10, 0, 'Educación'),
  Text(11, 0, 'Animación'),
  Text(12, 0, 'Videojuegos'),
  Text(13, 0, 'Música y baile, Animación'),
  Text(14, 0, 'Videojuegos, Humor'),
  Text(15, 0, 'Ciencia y tecnología'),
  Text(16, 0, 'Comida y bebida'),
  Text(17, 0, 'Películas, Juguetes'),
  Text(18, 0, 'Juguetes'),
  Text(19, 0, 'Humor'),
  Text(20, 0, 'Deportes'),
  Text(21, 0, 'Películas, Videojuegos'),
  Text(22, 0, 'Música y baile, Humor'),
  Text(23, 0, 'Juguetes, Coches y vehículos'),
  Text(24, 0, 'DIY y Life Hacks'),
  Text(25, 0, 'Fitness, Salud y autoayuda'),
  Text(26, 0, 'Videojuegos, Juguetes'),
  Text(27, 0, 'Animales y mascotas'),
  Text(28, 0, 'Moda'),
  Text(29, 0, 'Coches y vehículos'),
  Text(30, 0, 'Educación, Juguetes'),
  Text(31, 0, 'Fitness'),
  Text(32, 0, 'Comida y bebida, Juguetes'),
  Text(33, 0, 'ASMR, Comida y bebida'),
  Text(34, 0, 'Animación, Humor, Juguetes'),
  Text(35, 0, 'Diseño/arte, Belleza'),
  Text(36, 0, 'Belleza, Moda'),
  Text(37, 0, 'ASMR'),
  Text(38, 0, 'Música y baile, Juguetes'),
  Text(39, 0, 'Diseño/arte, DIY y Life Hacks'),
  Text(40, 0, 'DIY y Life Hacks, Juguetes'),
```

```
Text(41, 0, 'Diseño/arte'),
Text(42, 0, 'Comida y bebida, Salud y autoayuda'),
Text(43, 0, 'Viajes, Espectáculos'),
Text(44, 0, 'Juguetes, DIY y Life Hacks')])
```



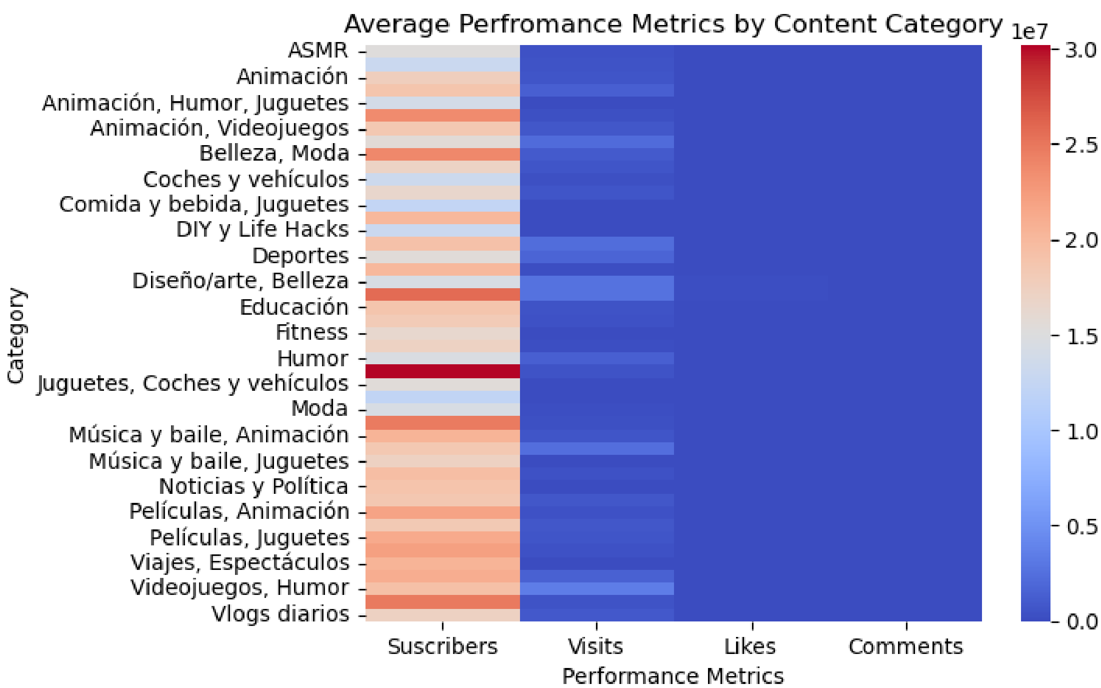
[84]:
```
#Identify the category with the highest number of streamers

max_category = category_counts.idxmax()
max_count = category_counts.max()
```

```
print(f"The category with the highest number of streamers is '{max_category}'␣
  ↪with '{max_count}' streamers")
```

```
The category with the highest number of streamers is 'Música y baile' with '156'
streamers
```

[85]:
```
#Calculate average metrics by category
ambc= df.groupby('Categories')[['Suscribers', 'Visits', 'Likes', 'Comments']].
  ↪mean()

#Visualize the average performance metrics by category

sns.heatmap(ambc, cmap="coolwarm", fmt=".2f")
plt.title("Average Perfromance Metrics by Content Category")
plt.xlabel("Performance Metrics")
plt.ylabel("Category")
```

[85]:
```
Text(50.2222222222222, 0.5, 'Category')
```



[86]:
```
df.columns
```

[86]:
```
Index(['Rank', 'Username', 'Categories', 'Suscribers', 'Country', 'Visits',
       'Likes', 'Comments', 'Links'],
      dtype='object')
```

4 BenchMarking

[87]:
```
#Calculate average values for each metric

avg_subscribers = df['Suscribers'].mean()
avg_visits = df['Visits'].mean()
avg_likes = df['Likes'].mean()
avg_comments = df['Comments'].mean()
```

[89]:
```
# Identify streamers with above-average performance

above_avg_streamers = df[
    (df['Suscribers'] > avg_subscribers) &
    (df['Visits'] > avg_visits) &
    (df['Likes'] > avg_likes) &
    (df['Comments'] > avg_comments)
]
```

[93]:
```
#Display information about the top-performing streamers

top_performing_streamers = above_avg_streamers.sort_values(by=['Suscribers'],␣
 ↪ascending=True)
print("Top Performing Streamers:")
print(top_performing_streamers[['Username', 'Suscribers', 'Visits', 'Likes',␣
 ↪'Comments']])
```

```
Top Performing Streamers:
               Username  Suscribers      Visits     Likes  Comments
319               romeo  21100000.0   3200000.0   53900.0    1600.0
315      lyricalemonade  21100000.0   2800000.0  127300.0    5800.0
318          kurzgesagt  21100000.0   4900000.0  253500.0   14000.0
304            infinite  21700000.0    884800.0   45700.0    1400.0
302          royaltyfam  21900000.0   4700000.0   67000.0    6600.0
285          BenAzelart  22500000.0   3700000.0   44900.0    2700.0
281             SSundee  22700000.0   1700000.0   59800.0    1800.0
278         StokesTwins  22700000.0  11700000.0  235000.0   10000.0
272        AmiRodrigueZZ  22900000.0   4300000.0  294400.0    1300.0
243        JamesCharles  23900000.0    964500.0   62300.0    1100.0
241  juandediospantojaa  24000000.0   3000000.0  133200.0    3600.0
234                 rug  24300000.0   3200000.0   85300.0    5100.0
207               ZHCYT  25700000.0   2600000.0  127300.0    2200.0
206            AlejoIgoa  25700000.0   5700000.0  208400.0    1700.0
202       VanossGaming  25900000.0   1300000.0   56500.0    1100.0
195          nickiminaj  26100000.0   1600000.0   98300.0    7600.0
180            NichLmao  27500000.0   1500000.0   85800.0    1600.0
179         brentrivera  27600000.0   6400000.0  154100.0    5000.0
177              DanTDM  27800000.0   3500000.0  285000.0   52500.0
```

```
171     SandeepSeminars  28000000.0  1200000.0   58500.0   4000.0
145       jacksepticeye  30400000.0  1600000.0   83400.0   2300.0
109        SSSniperWolf  34200000.0  1200000.0   34600.0   2100.0
100          markiplier  35500000.0  2100000.0  126500.0   3800.0
96       TotalGaming093  36300000.0  1500000.0  129400.0   4900.0
70          JessNoLimit  39600000.0  1300000.0   73500.0   1600.0
62       KimberlyLoaiza  42100000.0  5300000.0  271300.0  16000.0
58            Mikecrack  43400000.0  2200000.0  183400.0   1800.0
39          JuegaGerman  48600000.0  2000000.0  117100.0   3000.0
37         ArianaGrande  52900000.0  1100000.0   85800.0   3800.0
34          TaylorSwift  54100000.0  4300000.0  300400.0  15000.0
26          dudeperfect  59700000.0  5300000.0  156500.0   4200.0
14                  BTS  76500000.0   969700.0  180300.0   7400.0
```

5 Content Recommendations

[95]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
```

[96]:
```
df['user_id']=range(1,len(df['Username'])+1)
```

[97]:
```
x=df[['Rank', 'Visits', 'Comments']]
y=df['user_id']
```

[98]:
```
x_train,x_test, y_train, y_test= train_test_split(x,y,test_size=0.3)
```

[99]:
```
model = LinearRegression()
model.fit(x_train,y_train)
```

[99]:
```
LinearRegression()
```

[101]:
```
y_pred=model.predict(x_test)
y_pred
```

[101]:
```
array([5.27160366e+02, 5.03565524e+02, 2.85286090e-01, 2.07903591e+02,
       3.85044522e+02, 3.54755149e+01, 4.22149805e+01, 4.22685243e+02,
       3.46652469e+02, 1.09055785e+02, 5.85066567e+02, 1.53245897e+02,
       2.90071029e+02, 4.30163688e+02, 1.93760602e+02, 6.27516042e+02,
       5.23781535e+02, 1.04844846e+02, 2.72564306e+02, 1.25060195e+02,
       8.80147594e+01, 4.84054234e+02, 5.08190655e+02, 5.13688087e+02,
       3.02199752e+02, 5.16343414e+01, 6.26838552e+02, 4.95504278e+02,
       3.81287785e+01, 5.70940298e+02, 5.35231069e+02, 5.02202383e+02,
       1.82261150e+02, 1.35832085e+02, 1.75575313e+02, 1.33134005e+02,
       5.25776763e+02, 1.77594844e+02, 2.15309730e+02, 1.47955642e+02,
```

```
5.32537384e+02, 1.54018891e+02, 3.61535884e+01, 5.66222433e+02,
6.14046168e+02, 6.16726371e+02, 6.18761276e+02, 3.43231795e+02,
3.94468030e+02, 4.61818891e+02, 4.20741021e+02, 5.62857847e+02,
6.25476570e+02, 6.28190399e+02, 6.22795320e+02, 2.16491378e+02,
5.41304348e+02, 4.19393727e+02, 2.76582292e+02, 5.43998535e+02,
2.23396069e+02, 5.93836487e+02, 2.40907948e+02, 5.68918243e+02,
2.35519547e+02, 2.94790497e+02, 2.89283169e+02, 4.00526501e+02,
4.79315675e+02, 5.75655043e+02, 2.98829343e+02, 3.20384740e+02,
5.78343525e+02, 1.30379731e+02, 4.89504283e+01, 6.06635776e+02,
2.56354086e+02, 6.00574635e+02, 4.08585266e+02, 3.24425665e+02,
4.21405883e+02, 1.80237124e+02, 3.99851968e+02, 6.10674526e+02,
1.57357707e+02, 2.83972659e+02, 5.77675284e+02, 3.09606133e+02,
5.91819766e+02, 1.41885089e+02, 4.28478537e+01, 6.32231608e+02,
3.08934050e+02, 8.73097798e+01, 1.13585904e+02, 5.43323891e+02,
5.35895356e+02, 4.17372773e+02, 5.90472666e+02, 4.51044585e+02,
9.67624274e+01, 1.33805168e+02, 4.96074502e+02, 3.83620603e+02,
1.34483577e+02, 1.40550482e+02, 1.02159481e+02, 4.09290992e+02,
9.88425102e+00, 6.98302001e+01, 3.08261281e+02, 5.30525946e+02,
1.55296619e+02, 3.62137397e+02, 1.37842149e+02, 2.88051709e+02,
6.57142429e+02, 5.02913201e+02, 4.03229301e+02, 2.60458697e+01,
1.10577344e+00, 1.20338483e+02, 2.63804632e+02, 6.17413721e+02,
5.08972673e+02, 2.78626152e+02, 3.82909713e+02, 3.72895530e+02,
6.49069926e+02, 2.13962822e+02, 5.63443402e+02, 2.02506999e+02,
4.75905039e+01, 1.76241228e+02, 3.50693601e+02, 4.35479880e+02,
3.68225607e+01, 8.93611498e+01, 2.67175900e+02, 5.11666070e+02,
7.52154159e+01, 1.10908549e+02, 5.73620804e+02, 5.10992394e+02,
6.37670631e+01, 3.87048104e+02, 6.91525807e+01, 6.18087486e+02,
5.33886073e+02, 2.66496920e+02, 4.49660515e+02, 3.78981507e+02,
1.59409071e+02, 1.27046428e+02, 1.72128502e+01, 9.07083548e+01,
1.29693209e+02, 3.06665649e+02, 5.69582912e+02, 3.00169397e+02,
1.08894511e+02, 3.33182262e+02, 5.12034512e+02, 2.62461315e+02,
2.51007223e+02, 4.05923059e+02, 1.95743581e+02, 3.74940883e+02,
2.19343296e+02, 5.76997895e+02, 1.28427830e+02, 4.54412311e+02,
2.45622843e+02, 3.82350371e+02, 6.41660021e+02, 2.57745921e+02,
4.08621879e+01, 3.27793997e+02, 6.10005193e+02, 4.07222307e+02,
3.68201220e+02, 4.96851345e+02, 5.51407313e+02, 3.78308775e+02,
5.29177638e+02, 3.82358929e+00, 2.15987261e+02, 1.07545658e+02,
3.59384814e+02, 1.84278477e+02, 3.36549446e+02, 1.95105245e+02,
4.53026467e+02, 4.90789645e+02, 6.17479146e+01, 1.31110364e+02,
5.58135055e+02, 6.12691684e+02, 2.28108076e+02])
```

[103]:
```
#Calculate the performance metrics

mae= mean_absolute_error(y_test,y_pred)
mse= mean_squared_error(y_test,y_pred)
r2=r2_score(y_test,y_pred)
```

```
#Display the metrics

print(f"Mean Absolute Error (MAE){mae}")
print(f"Mean Squared Error (MSE){mse}")
print(f"R-Squared (R2){r2}")
```

```
Mean Absolute Error (MAE)2.1433747412581585
Mean Squared Error (MSE)7.1839602009039885
R-Squared (R2)0.9998056468217383
```

[106]:
```
#Functions to recommend streamers based on category and performance metrics

def recommendation_streamers_system(categories, performance_threshold):
    recommended_streamers = df[(df['Categories']== categories) &␣
 ↪df['performanceMetrics']]
    return recommend_streamers
print(recommendation_streamers_system)
```

```
<function recommendation_streamers_system at 0x000001EE128BC360>
```

6 Conclusion

In conclusion, our analysis of the YouTube streamers' dataset uncovered significant insights, ranging from popular content categories and audience preferences to correlations between engagement metrics. We identified top-perfroming streamers, offering benchmarks for success, and proposed a content recommendation system based on categories and performance metrics. Overall, these finding contribute to an understanding of the YouTube streaming landscape, empowering stakeholders to make informed decisions and enhance the user experience.

[ ]:
```

```