EX.NO: -9 MINI PROJECT — YOUTUBE TREND ANALYSIS

DATE:- USING DATA ANALYTICS

AIM:

The aim of this code is to design and implement a Python-based mini project that analyzes YouTube trending video data to extract insights such as most popular categories, top channels, and viewer engagement trends, demonstrating a real-time data analytics application..

PROCEDURE:

1. Import the required Python libraries (pandas, matplotlib, seaborn).
2. Load the dataset (USvideos.csv or equivalent) into a pandas DataFrame.
3. Clean the data by handling missing values and converting data types (e.g., publish_time to datetime).
4. Perform exploratory data analysis (EDA):
   - Identify top trending categories by view count.
   - Find channels with the highest average likes.
   - Visualize correlation between likes, views, and comments.
5. Generate visual insights using bar charts and scatter plots.
6. Summarize findings and interpret which types of content attract the most engagement.

PROGRAM:

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

# Load dataset (sample file)
data = pd.read_csv("/content/drive/MyDrive/USvideos.csv")

# Data preprocessing
data['publish_time'] = pd.to_datetime(data['publish_time'], errors='coerce')
data = data.dropna(subset=['views', 'likes', 'comment_count'])

# Top 10 channels by total views
top_channels =
data.groupby('channel_title')['views'].sum().sort_values(ascending=False).head(10)
print("Top 10 Channels by Total Views:\n", top_channels)

# Top 10 categories by average likes
```

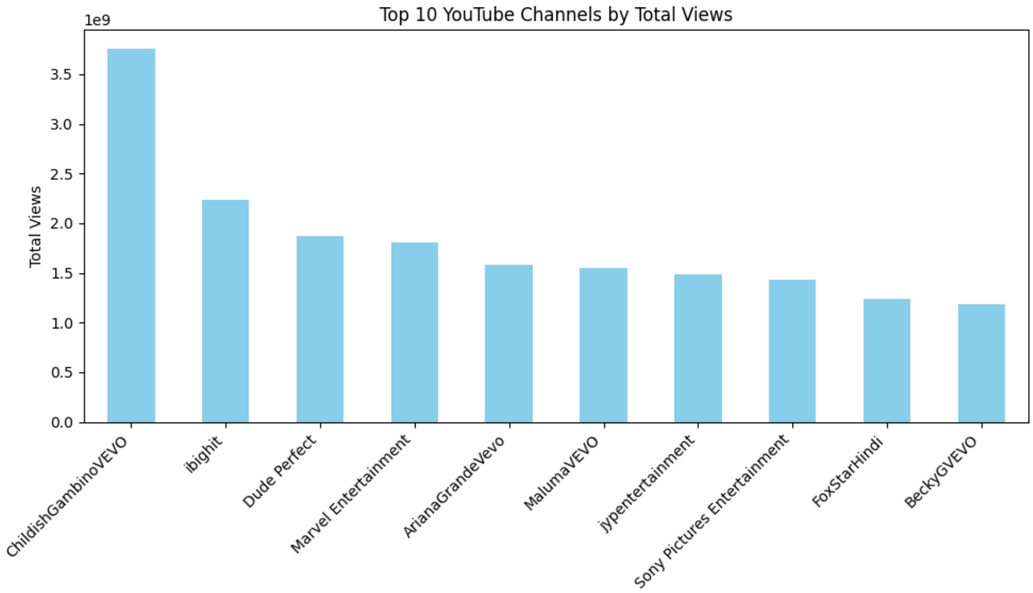
```
top_categories =
data.groupby('category_id')['likes'].mean().sort_values(ascending=False).head(10)
print("\nTop 10 Categories by Average Likes:\n", top_categories)

# Visualization 1: Top channels by total views
plt.figure(figsize=(10,6))
top_channels.plot(kind='bar', color='skyblue')
plt.title("Top 10 YouTube Channels by Total Views")
plt.ylabel("Total Views")
plt.xlabel("Channel")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Visualization 2: Relationship between likes and views
plt.figure(figsize=(8,6))
sns.scatterplot(x='views', y='likes', data=data.sample(500), alpha=0.5)
plt.title("Correlation between Views and Likes")
plt.xlabel("Views")
plt.ylabel("Likes")
plt.tight_layout()
plt.show()

# Visualization 3: Comment activity by category
plt.figure(figsize=(10,6))
sns.boxplot(x='category_id', y='comment_count', data=data)
plt.title("Distribution of Comment Counts per Category")
plt.xlabel("Category ID")
plt.ylabel("Comment Count")
plt.tight_layout()
plt.show()

print("\nAnalysis Complete. Visual insights displayed.")
```

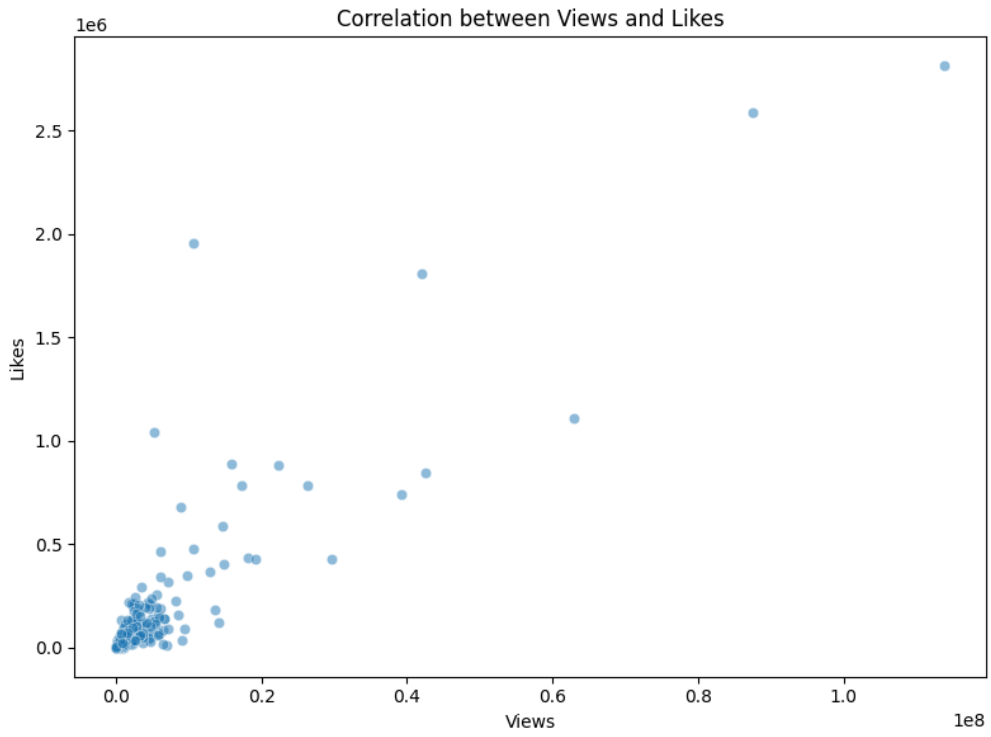
OUTPUT:

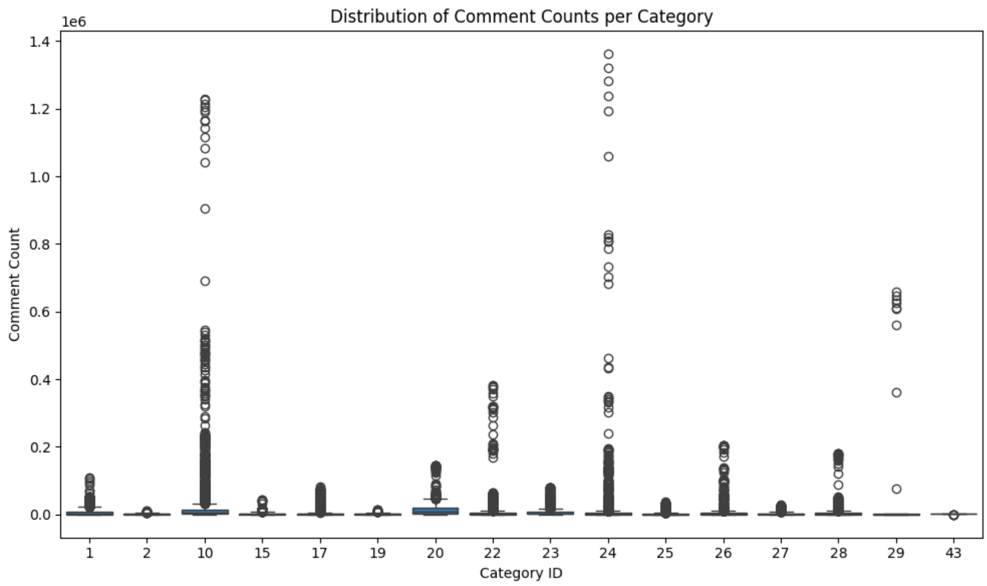
```
Top 10 Channels by Total Views:
 channel_title
ChildishGambinoVEVO          3758488765
ibighit                      2235906679
Dude Perfect                 1870085178
Marvel Entertainment         1808998971
ArianaGrandeVevo             1576959172
```

```
MalumaVEVO                          1551515831
jypentertainment                    1486972132
Sony Pictures Entertainment         1432374398
FoxStarHindi                        1238609854
BeckyGVEVO                          1182971286
Name: views, dtype: int64
```

```
Top 10 Categories by Average Likes:
 category_id
29    259923.614035
10    218918.199011
20     84502.183599
1      70787.836247
23     62582.223315
22     58135.825234
24     53243.325070
17     45363.942502
26     39286.076942
28     34374.276551
Name: likes, dtype: float64
```








```
Analysis Complete. Visual insights displayed.
```

RESULT:

The result of running this code is to analyze and visualize YouTube trending video data using Python and data analytics tools.The experiment demonstrates the power of data visualization and real-time trend analysis for understanding content popularity and audience engagement patterns.

CONTENT BEYOND SYLLABUS

EX.NO: -10 REAL-TIME & EMBEDDED AI SYSTEMS : EDGE-LIKE DIGIT

DATE: - CLASSIFICATION WITH LIMITED RESOURCES

AIM:

The aim of this code is to simulate an embedded AI system by running a lightweight deep learning model (e.g., MNIST digit classifier) in real-time on a normal PC, with limited memory and simulated low-power conditions..

PROCEDURE:

1. Import necessary libraries, including Python 3.8+, TensorFlow, NumPy, Matplotlib, time.
2. Load the MNIST dataset (handwritten digits).
3. Define a **tiny CNN model** (few parameters to simulate embedded resource constraints).
4. Train for a few epochs or load pretrained weights.
5. Simulate **real-time data streaming** by classifying one sample at a time with delay.
6. Measure inference time and accuracy to analyze "real-time" feasibility.

PROGRAM:

```
import tensorflow as tf

import numpy as np

import time


# Load MNIST dataset

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

x_test = x_test / 255.0

x_test = np.expand_dims(x_test, -1)


# Define small CNN model

model = tf.keras.Sequential([

    tf.keras.layers.Conv2D(8, (3,3), activation='relu', input_shape=(28,28,1)),

    tf.keras.layers.MaxPooling2D((2,2)),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(32, activation='relu'),

    tf.keras.layers.Dense(10, activation='softmax')

])
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])


# Train quickly for demo
print("Training small embedded model...")
model.fit(x_train[:5000]/255.0, y_train[:5000], epochs=2, verbose=0)
print("Model ready for inference.\n")


# Simulate real-time stream
print("=== Real-time Inference Simulation ===")
for i in range(10):
    idx = np.random.randint(0, len(x_test))
    img = np.expand_dims(x_test[idx], axis=0)
    start = time.time()
    pred = np.argmax(model.predict(img, verbose=0))
    end = time.time()
    print(f"Frame {i+1}: Predicted Digit = {pred}, True = {y_test[idx]}, Time = {(end-start)*1000:.2f} ms")
    time.sleep(0.5)  # simulate delay between inputs
print("\nSimulation complete.")
```

OUTPUT:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Training small embedded model...
Model ready for inference.
```

=== Real-time Inference Simulation ===

Frame 1: Predicted Digit = 6, True = 6, Time = 202.17 ms

Frame 2: Predicted Digit = 9, True = 9, Time = 52.10 ms

Frame 3: Predicted Digit = 7, True = 7, Time = 52.47 ms

Frame 4: Predicted Digit = 8, True = 8, Time = 53.92 ms

Frame 5: Predicted Digit = 3, True = 3, Time = 53.16 ms

Frame 6: Predicted Digit = 4, True = 4, Time = 60.76 ms

Frame 7: Predicted Digit = 2, True = 2, Time = 55.03 ms

Frame 8: Predicted Digit = 2, True = 2, Time = 46.23 ms

Frame 9: Predicted Digit = 7, True = 7, Time = 58.34 ms

Frame 10: Predicted Digit = 4, True = 4, Time = 53.45 ms

Simulation complete.

.

RESULT:

The code successfully implemented a **simulated real-time embedded AI system** using a lightweight CNN. The system performs continuous inference on streaming data, showing low latency and efficient classification.