In [ ]:
```
from googleapiclient.discovery import build
```

In [ ]:
```
import pandas as pd
import requests
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')
```

In [ ]:
```
from wordcloud import WordCloud
import re
import nltk
from nltk.corpus import stopwords, words
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer
import string
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import keras
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
```

In [ ]:
```
import cv2
from sklearn.metrics import mean_squared_error, r2_score
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input,Dense, Dropout, Flatten, LSTM, Embedding,cor
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import MaxPooling2D
```

In [ ]:
```
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('words')
```

In [ ]:
```
dict_ChannelSubs = {}
```

Extracting data from the YouTube API

Steps to gather the Video details

- Collect the channel ids of different channels
- Extract the playlist id and the subscriber count of each channel
- Extract all the video id from the given playlist id of each channel
- Finally extract all the information from the video and save it in a DataFrame

In [ ]:
```
# to gather the playlist ids and subscriber count of each channel
def get_channel_data(youtube, channel_ids):
```

```
    list_playlist_ids = []
    request = youtube.channels().list(
        part='snippet,contentDetails,statistics',
        id=','.join(channel_ids)
    )
    response = request.execute()

    for i in range(len(response['items'])):
      playlist_id = response['items'][i]['contentDetails']['relatedPlaylists']['uploa
      channel_id = response['items'][i]['id']
      subscribers = response['items'][i]['statistics']['subscriberCount']
      list_playlist_ids.append(playlist_id)
      dict_ChannelSubs[channel_id] = subscribers

    return list_playlist_ids
```

In [ ]:

```
# extract the video ids from each channel
def get_playlist_videoIds(youtube, playlist_id):
    video_ids = []

    request = youtube.playlistItems().list(
        part='contentDetails',
        playlistId=playlist_id,
        maxResults=50
    )

    response = request.execute()

    for i in range(len(response['items'])):
        video_ids.append(response['items'][i]['contentDetails']['videoId'])

    next_page_token = response.get('nextPageToken')
    more_pages = True

    while more_pages:
        if next_page_token is None:
            more_pages=False
        else:
            request = youtube.playlistItems().list(
                part='contentDetails',
                playlistId=playlist_id,
                maxResults=50,
                pageToken=next_page_token
            )

            response = request.execute()

            for i in range(len(response['items'])):
                video_ids.append(response['items'][i]['contentDetails']['videoId'])

            next_page_token = response.get('nextPageToken')
    return video_ids
```

In [ ]:

```
# extract the video details and
def get_video_details(youtube, video_ids, dict_ChannelSubs):

    all_video_stats = []

    for i in range(0, len(video_ids), 50):
```

```
            request = youtube.videos().list(
                part='snippet,statistics',
                id= ','.join(video_ids[i:i+50])
            )
            response = request.execute()

            for video in response['items']:
                try:
                    video_stats = dict(Id= video['id'],
                                       Title = video['snippet']['title'],
                                       Published_Date = video['snippet']['publishedAt']
                                       ThumbnailUrl= video['snippet']['thumbnails']['de
                                       LikesCount= video['statistics']['likeCount'],
                                       ViewsCount= video['statistics']['viewCount'],
                                       CommentCount= video['statistics']['commentCount'
                                       Subscribers= dict_ChannelSubs[video['snippet']['
                                       )
                    all_video_stats.append(video_stats)
                except:
                      pass

    return all_video_stats
```

In [ ]:
```
api_key = ''
channel_id = []
```

In [ ]:
```
youtube = build('youtube', 'v3', developerKey=api_key)
playlist_ids = get_channel_data(youtube, channel_id)
list_video_ids = []

for id in playlist_ids:
    video_ids = get_playlist_videoIds(youtube, id)
    list_video_ids.extend(video_ids)

video_details = get_video_details(youtube, list_video_ids, dict_ChannelSubs)
```

In [ ]:
```
data = pd.DataFrame(video_details)
```

Text Processing

In [ ]:
```
emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002500-\U00002BEF"  # chinese char
                           u"\U00002702-\U000027B0"
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           u"\U0001f926-\U0001f937"
                           u"\U00010000-\U0010ffff"
                           u"\u2640-\u2642"
                           u"\u2600-\u2B55"
                           u"\u200d"
                           u"\u23cf"
                           u"\u23e9"
```

```
                                u"\u231a"
                                u"\ufe0f"  # dingbats
                                u"\u3030"
                                "]+", flags=re.UNICODE)
```

In [ ]:
```
regular_punct = list(string.punctuation)
stop_words = set(stopwords.words('english'))
w_tokenizer = nltk.tokenize.WhitespaceTokenizer()
lemmatizer = nltk.stem.WordNetLemmatizer()
english_words = set(words.words())

def clean_text(text):

  #remove emoji
  sentence = emoji_pattern.sub(r'', text)

  #removing non english words and converting to lower
  sentence = ' '.join([word.lower() for word in sentence.split() if word not in (engli

  #remove stop words
  sentence =  ' '.join([word for word in sentence.split() if word not in (stop_words)

  # remove punctuation
  for punc in regular_punct:
        if punc in sentence:
            sentence = sentence.replace(punc, ' ')

  #lemmatize the words
  lemmatized_sentence = ""
  for w in w_tokenizer.tokenize(sentence):
        lemmatized_sentence = lemmatized_sentence + lemmatizer.lemmatize(w) + " "

  return lemmatized_sentence.strip()
```

In [ ]:
```
def bin_subs(text):

  subs = int(text)/1000000
  bin_subs = 0

  if(subs < 1):
    bin_subs = 0
  elif (1 < subs < 5):
    bin_subs = 1
  elif (5 < subs < 10):
    bin_subs = 2
  elif (10 < subs < 15):
    bin_subs = 3
  elif (15 < subs < 20):
    bin_subs = 4
  else:
    bin_subs = 5

  return bin_subs
```

In [ ]:
```
data['clean_text']=data['Title'].apply(lambda x : clean_text(x))
data['bin_subs'] = data['Subscribers'].apply(lambda x: bin_subs(x))
```

In [ ]:
```
# tokenize sentences
tokenizer = Tokenizer(1000)
tokenizer.fit_on_texts(data['clean_text'])
word_index = tokenizer.word_index
# convert train dataset to sequence and pad sequences
clean_text = tokenizer.texts_to_sequences(data['clean_text'])
clean_text = pad_sequences(clean_text, padding='pre', truncating= 'pre', maxlen=10)
```

Image Processing

In [ ]:
```
arr_images = []

for i in range(data.shape[0]):
  id = data.iloc[i]['Id']
  viewcount = data.iloc[i]['ViewsCount']
  img_path = f'drive//MyDrive//Thumbnails//{id}.jpg'

  image = cv2.imread(img_path)
  image = cv2.resize(image, (90,90), interpolation= cv2.INTER_LINEAR)
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  # Append the image and its corresponding label to the output
  arr_images.append(image)

arr_images = np.array(arr_images, dtype = 'float32') / 255.0
```

Numercial Data Processing

In [ ]:
```
numerical_data = data[['LikesCount', 'CommentCount', 'bin_subs']]
```

Model Building

In [ ]:
```
def linear_model(input_layer, input_shape):

  x = Dense(16, input_dim = input_shape, activation = 'relu')(input_layer)
  x = Dense(2, activation = 'relu')(x)

  return x
```

In [ ]:
```
def title_model(input_layer, input_shape):

  x = Embedding(1000, input_shape, input_length=input_shape)(input_layer)
  x = LSTM(128, return_sequences=True, input_shape= (input_shape, 1))(x)
  x = LSTM(64, return_sequences=False)(x)
  x = Flatten()(x)
  x = Dense(64, activation = 'relu')(x)
  x = Dropout(0.2)(x)
  x = Dense(16, activation = 'relu')(x)

  return x
```

In [ ]:
```
def thumbnail_model(input_layer, input_shape):
```

```
  x = Conv2D(64, kernel_size = (2,2), strides=2, padding="same", activation = 'relu',
  x = Conv2D(64, kernel_size = (2,2), strides=2, padding="same", activation  = 'relu')
  x = MaxPooling2D(pool_size = (2,2), padding="same")(x)
  x = Dropout(0.2)(x)
  x = Conv2D(32, kernel_size = (2,2), strides=2, padding="same", activation = 'relu')(
  x = MaxPooling2D(pool_size = (2,2), padding="same")(x)
  x = Conv2D(32, kernel_size = (2,2), strides=2, padding="same", activation = 'relu')(
  x = MaxPooling2D(pool_size = (2,2), padding="same")(x)
  x = Dropout(0.2)(x)
  x = Flatten()(x)
  x = Dense(1024, activation = 'relu')(x)

  return x
```

In [ ]:
```
Y = data['ViewsCount']

train_len = int(data.shape[0] * 0.8)
xtrain_txt = clean_text[:train_len]
xtest_txt = clean_text[train_len:]
xtrain_img = arr_images[:train_len]
xtest_img = arr_images[train_len:]
xtrain_num = numerical_data[:train_len]
xtest_num = numerical_data[train_len:]
ytrain = Y[:train_len]
ytest = Y[train_len:]
```

In [ ]:
```
num_input = Input(shape=3)
text_input = Input(shape=10)
image_input = Input(shape=(90,90,3))

num_layers = linear_model(num_input, 3)
text_layers = title_model(text_input, 10)
image_layers = thumbnail_model(image_input, (90,90,3))

out=concatenate([num_layers,text_layers,image_layers], axis=-1)
output=Dense(1, activation='relu')(out)
model = Model(inputs=[num_input,text_input, image_input], outputs=output)
```

In [ ]:
```
loss = tf.keras.losses.MeanSquaredError()
metric = [tf.keras.metrics.RootMeanSquaredError()]
optimizer = tf.keras.optimizers.Adam()
early_stopping = [tf.keras.callbacks.EarlyStopping(monitor = 'loss', patience = 5)]

model.compile(loss = loss, metrics = metric, optimizer = optimizer)

history = model.fit([xtrain_num,xtrain_txt, xtrain_img], ytrain, epochs = 300, callbac
```

In [ ]:
```
ypred = model.predict([xtest_num, xtest_txt,xtest_img])
mse = mean_squared_error(ytest, ypred)
print(f'Mean Squarred Error: {mse}')
print(f'Root Mean Squarred Error: {np.sqrt(mse)}')
print(f'R2 score: {r2_score(ytest, ypred)}')
```