#This Python 3 environment has pre-installed libraries that will help you when you code.

import pandas as pd #Importing CSV files and data preprocessing (for example, pd.read_csv)

#Input data files are available in the dataset directory.

import os print(os.listdir())

#The result that you write to the current directory as a .csv file is considered your submission file.

In [1]:

```
import pandas as pd
```

In [2]:

```
train = pd.read_csv('dataset/train.csv')
test = pd.read_csv('dataset/test.csv')
```

The task assigned is a Natural Language Processing problem where based on the text description of the video we are going to predict the category of the youtube channel.

The description column in its raw format is not clean and contains a variety of punctuations and stopwords (words like and,or, the,is, etc.). These needs to be removed first to prepare a usable training data. The function for this task is written below:

In [19]:

```
# Let's start with importing some necessary NLP and visualization libraries
import nltk
import re
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt

# importing some libraries for the machine learning modelling part
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
```

In [6]:

```
# function for text cleaning like removing \',whitespaces and eveything other than
def clean_text(text):
    # remove backslash-apostrophe
    text = re.sub("\'", "", text)
    # remove everything except alphabets
    text = re.sub("[^a-zA-Z]"," ",text)
    # remove whitespaces
    text = ' '.join(text.split())
    # convert text to lowercase
    text = text.lower()

    return text
```

In [7]:

```
# Now, we'll apply the 'clean_text' function on the 'description' column of trainin
train['description'] = train['description'].apply(lambda x: clean_text(x))
```

Next, we are going to write a function which will calulate the most occuring words in the 'description' column of the training data. This step is not mandatory for the problem that we are solving but I'm doing this just for the sake of knowing how many stopwords are there which are most occuring in the dataset.

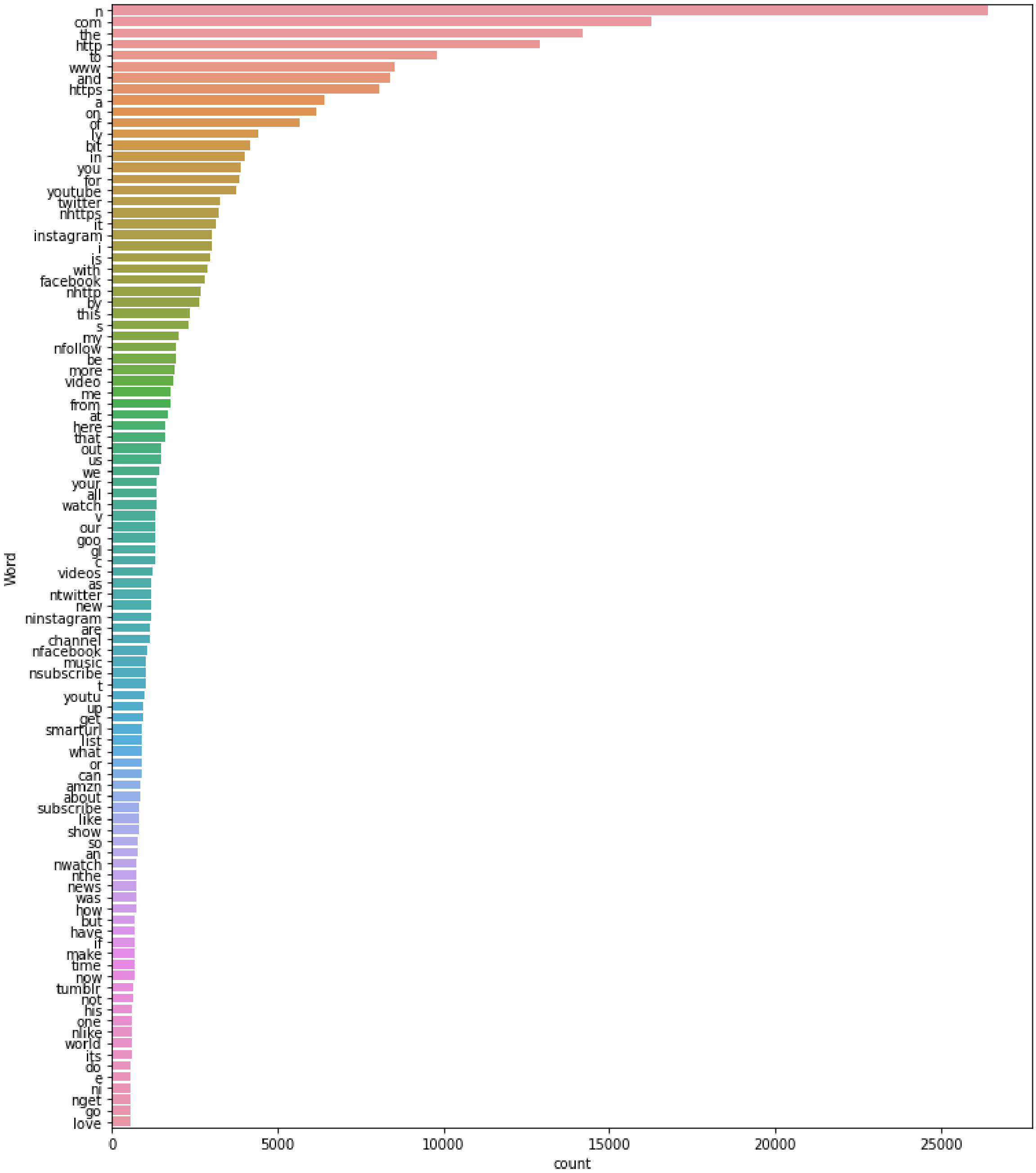
In [9]:

```
# Now let's define a function to calculate and visualize the most frequently occuri
def freq_words(x, terms = 30):
  all_words = ' '.join([text for text in x])
  all_words = all_words.split()
  fdist = nltk.FreqDist(all_words)
  words_df = pd.DataFrame({'word':list(fdist.keys()), 'count':list(fdist.values())}
  print(words_df.shape)

  # selecting top n most frequent words
  d = words_df.nlargest(columns="count", n = terms)

  # visualize words and frequencies
  plt.figure(figsize=(12,15))
  ax = sns.barplot(data=d, x= "count", y = "word")
  ax.set(ylabel = 'Word')
  plt.show()
```

In [12]:

```
# Let's have a look at the top 100 most occuring words of the 'description' column
freq_words(train['description'], 100)
```

(35231, 2)



From the graph above it is clearly visible that a majority of the most occuring words doesn't make any sense for our purpose. So let's try to remove these words.

In [14]:

```
# It is clearly visible that there are lots of stopwords in the column's text. Let'
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

# function to remove stopwords
def remove_stopwords(text):
    no_stopword_text = [w for w in text.split() if not w in stop_words]
    return ' '.join(no_stopword_text)
```

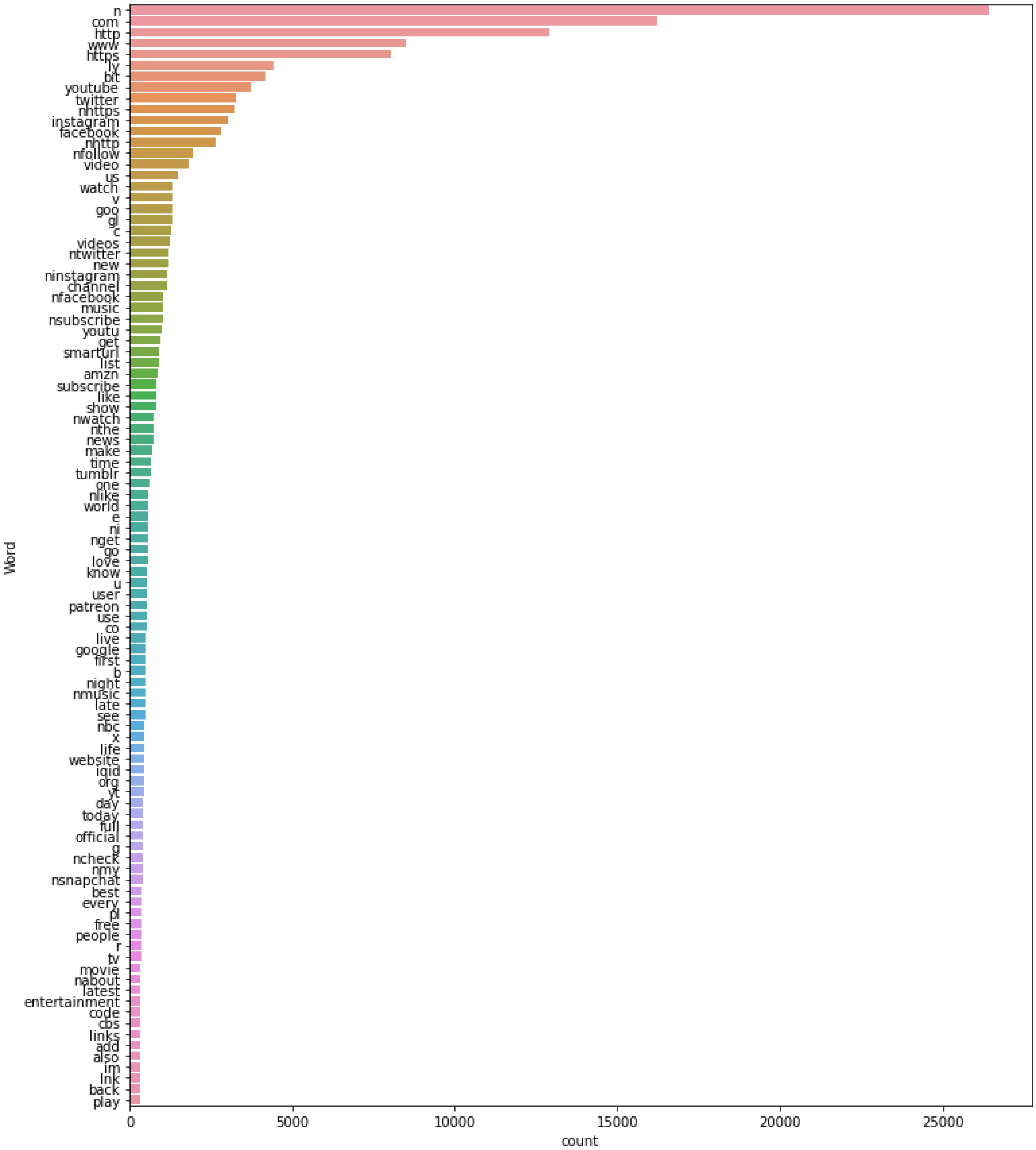
In [15]:

```
train['description'] = train['description'].apply(lambda x: remove_stopwords(x))
```

In [16]:

```
# Now again let's look at the top 100 words in the 'description' column after remov
freq_words(train['description'], 100)
```

(35085, 2)



Still, there are some unwanted words. For now, we'll move on to the next stage of modelling, to see how well the model perform on this cleaned dataset.

The very first step is to split the dataset into train and test data based on the 80-20 ratio using train_test_split function

In [20]:

```
# Let's split complete dataset in train and test data
X_train, X_test, y_train, y_test = train_test_split(train['description'], train['ca
```

The input to a ML model has to be in numeric form but the 'description' column is full of text. So, we need to vectorize the column's text and make it into numeric features using 'Tfidf Vectorizer'.

In [21]:

```
# This is a crucial step as it is the core of complete model. Now we'll convert the
tfidf_vectorizer = TfidfVectorizer()
```

In [22]:

```
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

In [23]:

```
# Its time to import and call the model
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier

lr = LogisticRegression()
clf = OneVsRestClassifier(lr)
```

In [24]:

```
# fitting model on train data
clf.fit(X_train_tfidf, y_train)
```

Out[24]:

```
OneVsRestClassifier(estimator=LogisticRegression())
```

In [25]:

```
# Let's predict the 'category_id' in the test set using the trained model
predictions = clf.predict(X_test_tfidf)
```

In [28]:

```
# Let's have a look at the accuracy score of our predictions
100*accuracy_score(y_true = y_test, y_pred = predictions)
```

Out[28]:

```
75.68345323741006
```

Wow! we have trained a baseline model which is accurately predicting the category of the youtube channel almost 76 times out of 100, which is a satisfactory result. Next, we'll calculate the evaluation criteria using f1-score

In [30]:

```
100*f1_score(y_test, predictions, average = 'weighted')
```

Out[30]:

```
74.88767985694268
```

It is satisfying to see that the accuracy and f1-score are not varying by a huge amount which indicates that our dataset is not imbalanced.

Next, let's try to experiment with other classification algorithms to see if they can perform better than logistic regression.

In [31]:

```
from sklearn.tree import DecisionTreeClassifier   #Decision Tree
from sklearn.ensemble import GradientBoostingClassifier   #Gradient Boosting Classi
from sklearn.ensemble import RandomForestClassifier   #Random Forest Classifier

dt = DecisionTreeClassifier()
gbr = GradientBoostingClassifier()
rf = RandomForestClassifier()
```

In [37]:

```
# Decision Tree Classifier

#fitting the model
dt.fit(X_train_tfidf, y_train)

#making predictions
dt_predictions = dt.predict(X_test_tfidf)

#accuracy score
print('Accuracy score:', 100*accuracy_score(y_true = y_test, y_pred = dt_prediction

#evaluation criteria
print('f-1 score:', 100*f1_score(y_test, dt_predictions, average = 'weighted'))
```

```
Accuracy score: 76.83453237410072
f-1 score: 76.93498251870551
```

In [38]:

```
# Gradient Boosting Classifier

#fitting the model
gbr.fit(X_train_tfidf, y_train)

#making predictions
gbr_predictions = gbr.predict(X_test_tfidf)

#accuracy score
print('Accuracy score:', 100*accuracy_score(y_true = y_test, y_pred = gbr_predictio

#evaluation criteria
print('f-1 score:', 100*f1_score(y_test, gbr_predictions, average = 'weighted'))
```

```
Accuracy score: 80.86330935251799
f-1 score: 80.96069886193727
```

In [39]:

```
# Random Forest Classifier

#fitting the model
rf.fit(X_train_tfidf, y_train)

#making predictions
rf_predictions = rf.predict(X_test_tfidf)

#accuracy score
print('Accuracy score:', 100*accuracy_score(y_true = y_test, y_pred = rf_prediction

#evaluation criteria
print('f-1 score:', 100*f1_score(y_test, rf_predictions, average = 'weighted'))
```

```
Accuracy score: 82.58992805755395
f-1 score: 82.74643679987344
```

Based on the comparison between above 4 algorithms; i.e Logistic regression, Decision Tree, Gradient Boosting and Random Forest. It is found that the baseline model of Random Forest Classifier performed best

with an f-1 score of 82.74

Now is the time to make the submission file of actual unlabelled test dataset. But first we have to preprocess it just like we did with train data. The steps involved are as follows

- Applying the clean_text function on the 'description' column
- Followed by the removal of stopwords
- Then, vectorizing the column using Tfidf Vectorizer
- Finally, predicting the 'category_id' using the trained Random Forest Classifier (rf) and storing them in 'submission.csv' file

In [44]:

```
test['description'] = test['description'].apply(lambda x: clean_text(x))
test['description'] = test['description'].apply(lambda x: remove_stopwords(x))
test_tfidf = tfidf_vectorizer.transform(test['description'])
test_predictions = rf.predict(test_tfidf)
```

Now, we have the predictions for the test dataset and the last step is to store it in a 'submission.csv' file in the proper format

In [53]:

```
df = test.copy()
```

In [56]:

```
df['category_id'] = test_predictions
```

In [59]:

```
del df['description']
```

In [61]:

```
df.to_csv('submission.csv', index = False)
```