In [1]:
```
import nltk
nltk.download('punkt')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
from nltk import stem
stemmer = stem.PorterStemmer()
from nltk import word_tokenize
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
import string
punct = list(string.punctuation)
punctuations = string.punctuation
from collections import Counter
import requests
import pandas as pd
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt
!pip install PRAW
import numpy as np
import praw
import datetime
import time
import os
import plotly
import plotly.express as px
!pip install jupyterlab "ipywidgets>=7.5"
from nltk.corpus import wordnet as wn
import plotly.graph_objects as go
import csv
vad = pd.read_excel('/Users/louisvsbigmac/Documents/Office 365/xlsx/vad.xlsx', index_col = 0)
import gensim
import gensim.downloader as api
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
from IPython.display import IFrame
import plotly.express as px
from sklearn.cluster import KMeans
from sklearn.cluster import AffinityPropagation
from scipy.spatial import distance
```

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.cm as cm
import spacy
from IPython.display import FileLink
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/louisvsbigmac/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/louisvsbigmac/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
Requirement already satisfied: PRAW in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (7.7.0)
Requirement already satisfied: prawcore<3,>=2.1 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (fro
m PRAW) (2.3.0)
Requirement already satisfied: update-checker>=0.18 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages
(from PRAW) (0.18.0)
Requirement already satisfied: websocket-client>=0.54.0 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packa
ges (from PRAW) (0.58.0)
Requirement already satisfied: requests<3.0,>=2.6.0 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages
(from prawcore<3,>=2.1->PRAW) (2.28.1)
Requirement already satisfied: six in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from websocket-c
lient>=0.54.0->PRAW) (1.16.0)
Requirement already satisfied: idna<4,>=2.5 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from re
quests<3.0,>=2.6.0->prawcore<3,>=2.1->PRAW) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages
(from requests<3.0,>=2.6.0->prawcore<3,>=2.1->PRAW) (1.26.11)
Requirement already satisfied: charset-normalizer<3,>=2 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packa
ges (from requests<3.0,>=2.6.0->prawcore<3,>=2.1->PRAW) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (f
rom requests<3.0,>=2.6.0->prawcore<3,>=2.1->PRAW) (2022.9.24)
Requirement already satisfied: jupyterlab in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (3.4.4)
Requirement already satisfied: ipywidgets>=7.5 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (7.6.
5)
Requirement already satisfied: ipython in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from jupyter
lab) (7.31.1)
Requirement already satisfied: nbclassic in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from jupyt
erlab) (0.3.5)
Requirement already satisfied: notebook<7 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from jupy
terlab) (6.4.12)
Requirement already satisfied: jupyterlab-server~=2.10 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packag
es (from jupyterlab) (2.10.3)
Requirement already satisfied: tornado>=6.1.0 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from
jupyterlab) (6.1)
Requirement already satisfied: jinja2>=2.1 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from jup
yterlab) (2.11.3)
Requirement already satisfied: jupyter-core in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from ju
pyterlab) (4.11.1)
Requirement already satisfied: packaging in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from jupyt
erlab) (21.3)
Requirement already satisfied: jupyter-server~=1.16 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages
(from jupyterlab) (1.18.1)
Requirement already satisfied: ipython-genutils~=0.2.0 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packag
es (from ipywidgets>=7.5) (0.2.0)
Requirement already satisfied: ipykernel>=4.5.1 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (fro
m ipywidgets>=7.5) (6.15.2)
```

```
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-pack
ages (from ipywidgets>=7.5) (1.0.0)
Requirement already satisfied: traitlets>=4.3.1 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (fro
m ipywidgets>=7.5) (5.1.1)
Requirement already satisfied: nbformat>=4.2.0 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from
ipywidgets>=7.5) (5.5.0)
Requirement already satisfied: widgetsnbextension~=3.5.0 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-pack
ages (from ipywidgets>=7.5) (3.5.2)
Requirement already satisfied: psutil in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from ipykerne
l>=4.5.1->ipywidgets>=7.5) (5.9.0)
Requirement already satisfied: jupyter-client>=6.1.12 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-package
s (from ipykernel>=4.5.1->ipywidgets>=7.5) (7.3.4)
Requirement already satisfied: nest-asyncio in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from ip
ykernel>=4.5.1->ipywidgets>=7.5) (1.5.5)
Requirement already satisfied: appnope in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from ipykern
el>=4.5.1->ipywidgets>=7.5) (0.1.2)
Requirement already satisfied: debugpy>=1.0 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from ip
ykernel>=4.5.1->ipywidgets>=7.5) (1.5.1)
Requirement already satisfied: matplotlib-inline>=0.1 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-package
s (from ipykernel>=4.5.1->ipywidgets>=7.5) (0.1.6)
Requirement already satisfied: pyzmq>=17 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from ipyke
rnel>=4.5.1->ipywidgets>=7.5) (23.2.0)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /Users/louisvsbigmac/opt/anaconda3/lib/
python3.9/site-packages (from ipython->jupyterlab) (3.0.20)
Requirement already satisfied: pygments in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from ipytho
n->jupyterlab) (2.11.2)
Requirement already satisfied: jedi>=0.16 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from ipyt
hon->jupyterlab) (0.18.1)
Requirement already satisfied: pexpect>4.3 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from ipy
thon->jupyterlab) (4.8.0)
Requirement already satisfied: pickleshare in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from ipy
thon->jupyterlab) (0.7.5)
Requirement already satisfied: decorator in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from ipyth
on->jupyterlab) (5.1.1)
Requirement already satisfied: setuptools>=18.5 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (fro
m ipython->jupyterlab) (63.4.1)
Requirement already satisfied: backcall in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from ipytho
n->jupyterlab) (0.2.0)
Requirement already satisfied: MarkupSafe>=0.23 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (fro
m jinja2>=2.1->jupyterlab) (2.0.1)
Requirement already satisfied: Send2Trash in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from jupy
ter-server~=1.16->jupyterlab) (1.8.0)
Requirement already satisfied: anyio<4,>=3.1.0 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from
jupyter-server~=1.16->jupyterlab) (3.5.0)
```

```
Requirement already satisfied: terminado>=0.8.3 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (fro
m jupyter-server~=1.16->jupyterlab) (0.13.1)
Requirement already satisfied: argon2-cffi in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from jup
yter-server~=1.16->jupyterlab) (21.3.0)
Requirement already satisfied: websocket-client in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (fro
m jupyter-server~=1.16->jupyterlab) (0.58.0)
Requirement already satisfied: nbconvert>=6.4.4 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (fro
m jupyter-server~=1.16->jupyterlab) (6.4.4)
Requirement already satisfied: prometheus-client in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (fr
om jupyter-server~=1.16->jupyterlab) (0.14.1)
Requirement already satisfied: requests in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from jupyte
rlab-server~=2.10->jupyterlab) (2.28.1)
Requirement already satisfied: babel in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from jupyterla
b-server~=2.10->jupyterlab) (2.9.1)
Requirement already satisfied: json5 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from jupyterla
b-server~=2.10->jupyterlab) (0.9.6)
Requirement already satisfied: jsonschema>=3.0.1 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (fr
om jupyterlab-server~=2.10->jupyterlab) (4.16.0)
Requirement already satisfied: entrypoints>=0.2.2 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (f
rom jupyterlab-server~=2.10->jupyterlab) (0.4)
Requirement already satisfied: fastjsonschema in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from
 nbformat>=4.2.0->ipywidgets>=7.5) (2.16.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packa
ges (from packaging->jupyterlab) (3.0.9)
Requirement already satisfied: idna>=2.8 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from anyio
<4,>=3.1.0->jupyter-server~=1.16->jupyterlab) (3.3)
Requirement already satisfied: sniffio>=1.1 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from an
yio<4,>=3.1.0->jupyter-server~=1.16->jupyterlab) (1.2.0)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages
(from jedi>=0.16->ipython->jupyterlab) (0.8.3)
Requirement already satisfied: attrs>=17.4.0 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from j
sonschema>=3.0.1->jupyterlab-server~=2.10->jupyterlab) (21.4.0)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in /Users/louisvsbigmac/opt/anaconda3/li
b/python3.9/site-packages (from jsonschema>=3.0.1->jupyterlab-server~=2.10->jupyterlab) (0.18.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-package
s (from jupyter-client>=6.1.12->ipykernel>=4.5.1->ipywidgets>=7.5) (2.8.2)
Requirement already satisfied: mistune<2,>=0.8.1 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (fr
om nbconvert>=6.4.4->jupyter-server~=1.16->jupyterlab) (0.8.4)
Requirement already satisfied: beautifulsoup4 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from
 nbconvert>=6.4.4->jupyter-server~=1.16->jupyterlab) (4.11.1)
Requirement already satisfied: pandocfilters>=1.4.1 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages
(from nbconvert>=6.4.4->jupyter-server~=1.16->jupyterlab) (1.5.0)
Requirement already satisfied: testpath in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from nbconv
ert>=6.4.4->jupyter-server~=1.16->jupyterlab) (0.6.0)
```

```
Requirement already satisfied: defusedxml in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from nbco
nvert>=6.4.4->jupyter-server~=1.16->jupyterlab) (0.7.1)
Requirement already satisfied: jupyterlab-pygments in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages
(from nbconvert>=6.4.4->jupyter-server~=1.16->jupyterlab) (0.1.2)
Requirement already satisfied: bleach in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from nbconver
t>=6.4.4->jupyter-server~=1.16->jupyterlab) (4.1.0)
Requirement already satisfied: nbclient<0.6.0,>=0.5.0 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-package
s (from nbconvert>=6.4.4->jupyter-server~=1.16->jupyterlab) (0.5.13)
Requirement already satisfied: ptyprocess>=0.5 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from
pexpect>4.3->ipython->jupyterlab) (0.7.0)
Requirement already satisfied: wcwidth in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from prompt-
toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython->jupyterlab) (0.2.5)
Requirement already satisfied: argon2-cffi-bindings in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages
(from argon2-cffi->jupyter-server~=1.16->jupyterlab) (21.2.0)
Requirement already satisfied: pytz>=2015.7 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from ba
bel->jupyterlab-server~=2.10->jupyterlab) (2022.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages
(from requests->jupyterlab-server~=2.10->jupyterlab) (1.26.11)
Requirement already satisfied: certifi>=2017.4.17 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (f
rom requests->jupyterlab-server~=2.10->jupyterlab) (2022.9.24)
Requirement already satisfied: charset-normalizer<3,>=2 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packa
ges (from requests->jupyterlab-server~=2.10->jupyterlab) (2.0.4)
Requirement already satisfied: six in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from websocket-c
lient->jupyter-server~=1.16->jupyterlab) (1.16.0)
Requirement already satisfied: cffi>=1.0.1 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from arg
on2-cffi-bindings->argon2-cffi->jupyter-server~=1.16->jupyterlab) (1.15.1)
Requirement already satisfied: soupsieve>1.2 in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from b
eautifulsoup4->nbconvert>=6.4.4->jupyter-server~=1.16->jupyterlab) (2.3.1)
Requirement already satisfied: webencodings in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from bl
each->nbconvert>=6.4.4->jupyter-server~=1.16->jupyterlab) (0.5.1)
Requirement already satisfied: pycparser in /Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages (from cffi>
=1.0.1->argon2-cffi-bindings->argon2-cffi->jupyter-server~=1.16->jupyterlab) (2.21)
```

In [2]:

```
#Admin for YouTube API
API_KEY = "AIzaSyBKb9YXkQrDixv7uB7BGfKKSoi0Nk4A1ag"

#Right Wing
DM_CHANNEL_ID = "UCw3fku0sH3qA3c3pZeJwdAw"
DE_CHANNEL_ID = "UCz8omIXCPabFEgblrC_VIfA"

#Centrist
BBC_CHANNEL_ID = "UC16niRr50-MSBwiO3YDb3RA"
FT_CHANNEL_ID = "UCoUxsWakJucWg46KW5RsvPw"
```

```
#Left Wing
TG_CHANNEL_ID = "UCIRYBXDze5krPDzAEOxFGVA"
TM_CHANNEL_ID = "UC3EmxrWVl7K8xBH_UVIWY4Q"


#Film & Animation
TPP_CHANNEL_ID = "UCFeUyPY6W8qX8w2o6oSiRmw"
PP_CHANNEL_ID = "UCAOtE1V7Ots4DjM8JLlrYgg"

#Gaming
TDM_CHANNEL_ID = "UCS5Oz6CHmeoF7vSad0qqXfw"
JLY_CHANNEL_ID = "UC0DZmkupLYwc0yDsfocLh0A"

#Lifestyle/How-to
DH_CHANNEL_ID = "UC0rDDvHM7u_7aWgAojSXl1Q"
TT_CHANNEL_ID = "UCGLDtG2tl0uG8P0eNstbLUA"

#Music
NC_CHANNEL_ID = "UC_aEa8K-EOJ3D6gOs7HcyNg"
CP_CHANNEL_ID = "UCDPM_n1atn2ijUwHd0NNRQw"

#Non-profit
TP_CHANNEL_ID = "UCW4MQYT4fh6yZbmUPhLFvqg"
EE_CHANNEL_ID = "UCVRrGAcUc7cblUzOhI1KfFg"

#People and Blogs
JL_CHANNEL_ID = "UC5Y1pPVNm-odhiubuRO1D6A"
SM_CHANNEL_ID = "UCh5mLn90vUaB1PbRRx_AiaA"

#Animals
KG_CHANNEL_ID = "UC_9EXcw4PlVlaklxx12cDqA"
OM_CHANNEL_ID = "UCejVe2sNPjjvfCXg35q_EQQ"

#Science
WB_CHANNEL_ID = "UCMiJRAwDNSNzuYeN2uWa0pA"
STM_CHANNEL_ID = "UCEIwxahdLz7bap-VDs9h35A"

#Sport
FF_CHANNEL_ID = "UCKvn9VBLAiLiYL4FFJHri6g"
ICC_CHANNEL_ID = "UCt2JXOLNxqry7B_4rRZME3Q"

#Travel
BB_CHANNEL_ID = "UCxDZs_ltFFvn0FDHT6kmoXA"
DP_CHANNEL_ID = "UCKygRpISlqs5TufcT3JtRng"
```

```
#Auto
CW_CHANNEL_ID = "UCUhFaUpnq31m6TNX2VKVSVA"
CT_CHANNEL_ID = "UCNBbCOuAN1NZAuj0vPe_MkA"

#Comedy
NN_CHANNEL_ID = "UCVjlpEjEY9GpksqbEesJnNA"
WN_CHANNEL_ID = "UCaFUrR3oSxOl5Y9y6tvLTEg"

#Education
JB_CHANNEL_ID = "UChUHOEe7zgYmrv7a_LlpNng"
MK_CHANNEL_ID = "UC1QqdFv5e9QSEi0eYdfwuqw"

#Entertainment
DR_CHANNEL_ID = "UC6D1L2vxEAg_Vi0JSxMBDgA"
BGT_CHANNEL_ID = "UCUtZaxDF3hD5VK4xRYFBePQ"
```

In [3]:

```
DM_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+DM_CHANNEL_ID+"&part=snippet,id&ord
DM_response = requests.get(DM_url).json()

DE_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+DE_CHANNEL_ID+"&part=snippet,id&ord
DE_response = requests.get(DE_url).json()

BBC_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+BBC_CHANNEL_ID+"&part=snippet,id&o
BBC_response = requests.get(BBC_url).json()

FT_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+FT_CHANNEL_ID+"&part=snippet,id&ord
FT_response = requests.get(FT_url).json()

TG_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+TG_CHANNEL_ID+"&part=snippet,id&ord
TG_response = requests.get(TG_url).json()

TM_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+TM_CHANNEL_ID+"&part=snippet,id&ord
TM_response = requests.get(TM_url).json()

TPP_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+TPP_CHANNEL_ID+"&part=snippet,id&o
TPP_response = requests.get(TPP_url).json()

PP_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+PP_CHANNEL_ID+"&part=snippet,id&ord
PP_response = requests.get(PP_url).json()

TDM_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+TDM_CHANNEL_ID+"&part=snippet,id&o
TDM_response = requests.get(TDM_url).json()
```

```
JLY_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+JLY_CHANNEL_ID+"&part=snippet,id&o
JLY_response = requests.get(JLY_url).json()

SM_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+SM_CHANNEL_ID+"&part=snippet,id&ord
SM_response = requests.get(SM_url).json()

JL_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+JL_CHANNEL_ID+"&part=snippet,id&ord
JL_response = requests.get(JL_url).json()

KG_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+KG_CHANNEL_ID+"&part=snippet,id&ord
KG_response = requests.get(KG_url).json()

OM_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+OM_CHANNEL_ID+"&part=snippet,id&ord
OM_response = requests.get(OM_url).json()

WB_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+WB_CHANNEL_ID+"&part=snippet,id&ord
WB_response = requests.get(WB_url).json()

STM_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+STM_CHANNEL_ID+"&part=snippet,id&o
STM_response = requests.get(STM_url).json()

FF_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+FF_CHANNEL_ID+"&part=snippet,id&ord
FF_response = requests.get(FF_url).json()

ICC_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+ICC_CHANNEL_ID+"&part=snippet,id&o
ICC_response = requests.get(ICC_url).json()

BB_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+BB_CHANNEL_ID+"&part=snippet,id&ord
BB_response = requests.get(BB_url).json()

DP_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+DP_CHANNEL_ID+"&part=snippet,id&ord
DP_response = requests.get(DP_url).json()

CW_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+CW_CHANNEL_ID+"&part=snippet,id&ord
CW_response = requests.get(CW_url).json()

CT_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+CT_CHANNEL_ID+"&part=snippet,id&ord
CT_response = requests.get(CT_url).json()

NN_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+NN_CHANNEL_ID+"&part=snippet,id&ord
NN_response = requests.get(NN_url).json()

WN_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+WN_CHANNEL_ID+"&part=snippet,id&ord
```

```
WN_response = requests.get(WN_url).json()

JB_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+JB_CHANNEL_ID+"&part=snippet,id&ord
JB_response = requests.get(JB_url).json()

MK_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+MK_CHANNEL_ID+"&part=snippet,id&ord
MK_response = requests.get(MK_url).json()

DR_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+DR_CHANNEL_ID+"&part=snippet,id&ord
DR_response = requests.get(DR_url).json()

BGT_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+BGT_CHANNEL_ID+"&part=snippet,id&o
BGT_response = requests.get(BGT_url).json()

DH_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+DH_CHANNEL_ID+"&part=snippet,id&ord
DH_response = requests.get(DH_url).json()

TT_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+TT_CHANNEL_ID+"&part=snippet,id&ord
TT_response = requests.get(TT_url).json()

NC_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+NC_CHANNEL_ID+"&part=snippet,id&ord
NC_response = requests.get(NC_url).json()

CP_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+CP_CHANNEL_ID+"&part=snippet,id&ord
CP_response = requests.get(CP_url).json()

TP_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+TP_CHANNEL_ID+"&part=snippet,id&ord
TP_response = requests.get(TP_url).json()

EE_url = "https://www.googleapis.com/youtube/v3/search?key="+API_KEY+"&channelId="+EE_CHANNEL_ID+"&part=snippet,id&ord
EE_response = requests.get(EE_url).json()
```

In [6]:

```
#Obtaining Daily Mail's YouTube titles, joined together into long list.
DM_title_list = []
for video in DM_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        DM_titles = video['snippet']['title']
        DM_titles = str(DM_titles).replace("&#39;"," ")
        DM_titles = str(DM_titles).replace("|"," ")
        DM_titles = DM_titles.encode('ascii', 'ignore')
        DM_titles = DM_titles.decode()
        DM_title_list.append(DM_titles)
```

```
#Obtaining The Daily Express's YouTube titles, joined together into long list.
DE_title_list = []
for video in DE_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        DE_titles = video['snippet']['title']
        DE_titles = str(DE_titles).replace("&#39;"," ")
        DE_titles = str(DE_titles).replace("|"," ")
        DE_titles = str(DE_titles).replace("#shorts"," ")
        DE_titles = DE_titles.encode('ascii', 'ignore')
        DE_titles = DE_titles.decode()
        DE_title_list.append(DE_titles)

#Obtaining BBC's YouTube titles, joined together into long list.
BBC_title_list = []
for video in BBC_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        BBC_titles = video['snippet']['title']
        BBC_titles = str(BBC_titles).replace("&#39;"," ")
        BBC_titles = str(BBC_titles).replace("|"," ")
        BBC_titles = str(BBC_titles).replace("– BBC News"," ")
        BBC_titles = str(BBC_titles).replace("- BBC News"," ")
        BBC_titles = str(BBC_titles).replace("#shorts"," ")
        BBC_titles = BBC_titles.encode('ascii', 'ignore')
        BBC_titles = BBC_titles.decode()
        BBC_title_list.append(BBC_titles)

#Obtaining FT's YouTube titles, joined together into long list.
FT_title_list = []
for video in FT_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        FT_titles = video['snippet']['title']
        FT_titles = str(FT_titles).replace("&#39;"," ")
        FT_titles = str(FT_titles).replace("|"," ")
        FT_titles = str(FT_titles).replace("FT"," ")
        FT_titles = str(FT_titles).replace("#shorts"," ")
        FT_titles = FT_titles.encode('ascii', 'ignore')
        FT_titles = FT_titles.decode()
        FT_title_list.append(FT_titles)

#Obtaining The Guardian's YouTube titles, joined together into long list.
TG_title_list = []
for video in TG_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        TG_titles = video['snippet']['title']
```

```
        TG_titles = str(TG_titles).replace("&#39;"," ")
        TG_titles = str(TG_titles).replace("|"," ")
        TG_titles = str(TG_titles).replace("#shorts"," ")
        TG_titles = TG_titles.encode('ascii', 'ignore')
        TG_titles = TG_titles.decode()
        TG_title_list.append(TG_titles)

#Obtaining TM's YouTube titles, joined together into long list.
TM_title_list = []
for video in TM_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        TM_titles = video['snippet']['title']
        TM_titles = str(TM_titles).replace("&#39;"," ")
        TM_titles = str(TM_titles).replace("|"," ")
        TM_titles = str(TM_titles).replace("#shorts"," ")
        TM_titles = TM_titles.encode('ascii', 'ignore')
        TM_titles = TM_titles.decode()
        TM_title_list.append(TM_titles)

TPP_title_list = []
for video in TPP_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        TPP_titles = video['snippet']['title']
        TPP_titles = str(TPP_titles).replace("&#39;"," ")
        TPP_titles = str(TPP_titles).replace("|"," ")
        TPP_titles = str(TPP_titles).replace("Pink Panther"," ")
        TPP_titles = str(TPP_titles).replace("35-Minute Compilation"," ")
        TPP_titles = TPP_titles.encode('ascii', 'ignore')
        TPP_titles = TPP_titles.decode()
        TPP_title_list.append(TPP_titles)

PP_title_list = []
for video in PP_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        PP_titles = video['snippet']['title']
        PP_titles = str(PP_titles).replace("&#39;"," ")
        PP_titles = str(PP_titles).replace("|"," ")
        PP_titles = str(PP_titles).replace("Peppa Pig SPECIAL EPISODES"," ")
        PP_titles = str(PP_titles).replace("BRAND NEW SEASON 9 PEPPA"," ")
        PP_titles = PP_titles.encode('ascii', 'ignore')
        PP_titles = PP_titles.decode()
        PP_title_list.append(PP_titles)

TDM_title_list = []
```

```
for video in TDM_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        TDM_titles = video['snippet']['title']
        TDM_titles = str(TDM_titles).replace("&#39;"," ")
        TDM_titles = str(TDM_titles).replace("|"," ")
        TDM_titles = str(TDM_titles).replace("#shorts"," ")
        TDM_titles = TDM_titles.encode('ascii', 'ignore')
        TDM_titles = TDM_titles.decode()
        TDM_title_list.append(TDM_titles)

JLY_title_list = []
for video in JLY_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        JLY_titles = video['snippet']['title']
        JLY_titles = str(JLY_titles).replace("&#39;"," ")
        JLY_titles = str(JLY_titles).replace("|"," ")
        JLY_titles = str(JLY_titles).replace("#shorts"," ")
        JLY_titles = JLY_titles.encode('ascii', 'ignore')
        JLY_titles = JLY_titles.decode()
        JLY_title_list.append(JLY_titles)

SM_title_list = []
for video in SM_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        SM_titles = video['snippet']['title']
        SM_titles = str(SM_titles).replace("&#39;"," ")
        SM_titles = str(SM_titles).replace("|"," ")
        SM_titles = str(SM_titles).replace("#shorts"," ")
        SM_titles = SM_titles.encode('ascii', 'ignore')
        SM_titles = SM_titles.decode()
        SM_title_list.append(SM_titles)

JL_title_list = []
for video in JL_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        JL_titles = video['snippet']['title']
        JL_titles = str(JL_titles).replace("&#39;"," ")
        JL_titles = str(JL_titles).replace("|"," ")
        JL_titles = str(JL_titles).replace("#shorts"," ")
        JL_titles = JL_titles.encode('ascii', 'ignore')
        JL_titles = JL_titles.decode()
        JL_title_list.append(JL_titles)

KG_title_list = []
```

```
for video in KG_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        KG_titles = video['snippet']['title']
        KG_titles = str(KG_titles).replace("&#39;"," ")
        KG_titles = str(KG_titles).replace("|"," ")
        KG_titles = str(KG_titles).replace("#shorts"," ")
        KG_titles = KG_titles.encode('ascii', 'ignore')
        KG_titles = KG_titles.decode()
        KG_title_list.append(KG_titles)

OM_title_list = []
for video in OM_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        OM_titles = video['snippet']['title']
        OM_titles = str(OM_titles).replace("&#39;"," ")
        OM_titles = str(OM_titles).replace("|"," ")
        OM_titles = str(OM_titles).replace("1 Minute Animals"," ")
        OM_titles = OM_titles.encode('ascii', 'ignore')
        OM_titles = OM_titles.decode()
        OM_title_list.append(OM_titles)

WB_title_list = []
for video in WB_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        WB_titles = video['snippet']['title']
        WB_titles = str(WB_titles).replace("&#39;"," ")
        WB_titles = str(WB_titles).replace("|"," ")
        WB_titles = str(WB_titles).replace("#shorts"," ")
        WB_titles = WB_titles.encode('ascii', 'ignore')
        WB_titles = WB_titles.decode()
        WB_title_list.append(WB_titles)

STM_title_list = []
for video in STM_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        STM_titles = video['snippet']['title']
        STM_titles = str(STM_titles).replace("&#39;"," ")
        STM_titles = str(STM_titles).replace("|"," ")
        STM_titles = str(STM_titles).replace("#shorts"," ")
        STM_titles = STM_titles.encode('ascii', 'ignore')
        STM_titles = STM_titles.decode()
        STM_title_list.append(STM_titles)

FF_title_list = []
```

```
for video in FF_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        FF_titles = video['snippet']['title']
        FF_titles = str(FF_titles).replace("&#39;"," ")
        FF_titles = str(FF_titles).replace("|"," ")
        FF_titles = str(FF_titles).replace("#Shorts"," ")
        FF_titles = FF_titles.encode('ascii', 'ignore')
        FF_titles = FF_titles.decode()
        FF_title_list.append(FF_titles)

ICC_title_list = []
for video in ICC_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        ICC_titles = video['snippet']['title']
        ICC_titles = str(ICC_titles).replace("&#39;"," ")
        ICC_titles = str(ICC_titles).replace("|"," ")
        ICC_titles = str(ICC_titles).replace("ICC Mens T20 World Cup Trophy Tour 2022"," ")
        ICC_titles = str(ICC_titles).replace("ICC Men s T20 World Cup Trophy Tour"," ")
        ICC_titles = str(ICC_titles).replace("ICC Men s T20 World Cup 2022"," ")
        ICC_titles = str(ICC_titles).replace("ICC Men s T20WC 2022"," ")
        ICC_titles = str(ICC_titles).replace("T20WC 2022"," ")
        ICC_titles = str(ICC_titles).replace("T20 World Cup 2022"," ")
        ICC_titles = str(ICC_titles).replace("The ICC Review"," ")
        ICC_titles = ICC_titles.encode('ascii', 'ignore')
        ICC_titles = ICC_titles.decode()
        ICC_title_list.append(ICC_titles)

BB_title_list = []
for video in BB_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        BB_titles = video['snippet']['title']
        BB_titles = str(BB_titles).replace("&#39;"," ")
        BB_titles = str(BB_titles).replace("|"," ")
        BB_titles = str(BB_titles).replace("#shorts"," ")
        BB_titles = BB_titles.encode('ascii', 'ignore')
        BB_titles = BB_titles.decode()
        BB_title_list.append(BB_titles)

DP_title_list = []
for video in DP_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        DP_titles = video['snippet']['title']
        DP_titles = str(DP_titles).replace("&#39;"," ")
        DP_titles = str(DP_titles).replace("|"," ")
```

```
        DP_titles = str(DP_titles).replace("#shorts"," ")
        DP_titles = DP_titles.encode('ascii', 'ignore')
        DP_titles = DP_titles.decode()
        DP_title_list.append(DP_titles)

CW_title_list = []
for video in CW_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        CW_titles = video['snippet']['title']
        CW_titles = str(CW_titles).replace("&#39;"," ")
        CW_titles = str(CW_titles).replace("|"," ")
        CW_titles = str(CW_titles).replace("#shorts"," ")
        CW_titles = CW_titles.encode('ascii', 'ignore')
        CW_titles = CW_titles.decode()
        CW_title_list.append(CW_titles)

CT_title_list = []
for video in CT_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        CT_titles = video['snippet']['title']
        CT_titles = str(CT_titles).replace("&#39;"," ")
        CT_titles = str(CT_titles).replace("|"," ")
        CT_titles = str(CT_titles).replace("#shorts"," ")
        CT_titles = CT_titles.encode('ascii', 'ignore')
        CT_titles = CT_titles.decode()
        CT_title_list.append(CT_titles)

NN_title_list = []
for video in NN_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        NN_titles = video['snippet']['title']
        NN_titles = str(NN_titles).replace("&#39;"," ")
        NN_titles = str(NN_titles).replace("|"," ")
        NN_titles = str(NN_titles).replace("Uncle Roger"," ")
        NN_titles = str(NN_titles).replace("UNCLE ROGER"," ")
        NN_titles = NN_titles.encode('ascii', 'ignore')
        NN_titles = NN_titles.decode()
        NN_title_list.append(NN_titles)

WN_title_list = []
for video in WN_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        WN_titles = video['snippet']['title']
        WN_titles = str(WN_titles).replace("&#39;"," ")
```

```
        WN_titles = str(WN_titles).replace("|"," ")
        WN_titles = str(WN_titles).replace("#shorts"," ")
        WN_titles = WN_titles.encode('ascii', 'ignore')
        WN_titles = WN_titles.decode()
        WN_title_list.append(WN_titles)

JB_title_list = []
for video in JB_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        JB_titles = video['snippet']['title']
        JB_titles = str(JB_titles).replace("Jungle Beat: Munki &amp; Trunk"," ")
        JB_titles = str(JB_titles).replace("Jungle Beat: Munki and Trunk"," ")
        JB_titles = str(JB_titles).replace("|"," ")
        JB_titles = str(JB_titles).replace("Kids Animation 2023"," ")
        JB_titles = str(JB_titles).replace("Kids Animation 2022"," ")
        JB_titles = str(JB_titles).replace("Jungle Beat: Munki and Trunk"," ")
        JB_titles = JB_titles.encode('ascii', 'ignore')
        JB_titles = JB_titles.decode()
        JB_title_list.append(JB_titles)

BGT_title_list = []
for video in BGT_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        BGT_titles = video['snippet']['title']
        BGT_titles = str(BGT_titles).replace("&#39;"," ")
        BGT_titles = str(BGT_titles).replace("|"," ")
        BGT_titles = str(BGT_titles).replace("Britain s Got Talent"," ")
        BGT_titles = BGT_titles.encode('ascii', 'ignore')
        BGT_titles = BGT_titles.decode()
        BGT_title_list.append(BGT_titles)

MK_title_list = []
for video in MK_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        MK_titles = video['snippet']['title']
        MK_titles = str(MK_titles).replace("&#39;"," ")
        MK_titles = str(MK_titles).replace("|"," ")
        MK_titles = str(MK_titles).replace("Nursery Rhymes"," ")
        MK_titles = str(MK_titles).replace("Sing Along"," ")
        MK_titles = str(MK_titles).replace("Kids Learn!"," ")
        MK_titles = MK_titles.encode('ascii', 'ignore')
        MK_titles = MK_titles.decode()
        MK_title_list.append(MK_titles)
```

```
DR_title_list = []
for video in DR_response ['items']:
    if video ['id']['kind'] == "youtube#video":
        DR_titles = video['snippet']['title']
        DR_titles = str(DR_titles).replace("&#39;"," ")
        DR_titles = str(DR_titles).replace("|"," ")
        DR_titles = str(DR_titles).replace("#shorts"," ")
        DR_titles = DR_titles.encode('ascii', 'ignore')
        DR_titles = DR_titles.decode()
        DR_title_list.append(DR_titles)

DH_title_list = []
for video in DH_response['items']:
    if video['id']['kind'] == "youtube#video":
        DH_titles = video['snippet']['title']
        DH_titles = str(DH_titles).replace("'", " ")
        DH_titles = str(DH_titles).replace("|", " ")
        DH_titles = str(DH_titles).replace("#shorts", " ")
        DH_titles = DH_titles.encode('ascii', 'ignore')
        DH_titles = DH_titles.decode()
        DH_title_list.append(DH_titles)

TT_title_list = []
for video in TT_response['items']:
    if video['id']['kind'] == "youtube#video":
        TT_titles = video['snippet']['title']
        TT_titles = str(TT_titles).replace("&#39;", " ")
        TT_titles = str(TT_titles).replace("|", " ")
        TT_titles = str(TT_titles).replace("#shorts", " ")
        TT_titles = TT_titles.encode('ascii', 'ignore')
        TT_titles = TT_titles.decode()
        TT_title_list.append(TT_titles)

NC_title_list = []
for video in NC_response['items']:
    if video['id']['kind'] == "youtube#video":
        NC_titles = video['snippet']['title']
        NC_titles = str(NC_titles).replace("&#39;", " ")
        NC_titles = str(NC_titles).replace("|", " ")
        NC_titles = str(NC_titles).replace("#shorts", " ")
        NC_titles = NC_titles.encode('ascii', 'ignore')
        NC_titles = NC_titles.decode()
        NC_title_list.append(NC_titles)
```

```
CP_title_list = []
for video in CP_response['items']:
    if video['id']['kind'] == "youtube#video":
        CP_titles = video['snippet']['title']
        CP_titles = str(CP_titles).replace("&#39;", " ")
        CP_titles = str(CP_titles).replace("|", " ")
        CP_titles = str(CP_titles).replace("#shorts", " ")
        CP_titles = CP_titles.encode('ascii', 'ignore')
        CP_titles = CP_titles.decode()
        CP_title_list.append(CP_titles)

TP_title_list = []
for video in TP_response['items']:
    if video['id']['kind'] == "youtube#video":
        TP_titles = video['snippet']['title']
        TP_titles = str(TP_titles).replace("&#39;", " ")
        TP_titles = str(TP_titles).replace("|", " ")
        TP_titles = str(TP_titles).replace("#shorts", " ")
        TP_titles = TP_titles.encode('ascii', 'ignore')
        TP_titles = TP_titles.decode()
        TP_title_list.append(TP_titles)

EE_title_list = []
for video in EE_response['items']:
    if video['id']['kind'] == "youtube#video":
        EE_titles = video['snippet']['title']
        EE_titles = str(EE_titles).replace("&#39;", " ")
        EE_titles = str(EE_titles).replace("|", " ")
        EE_titles = str(EE_titles).replace("#shorts", " ")
        EE_titles = EE_titles.encode('ascii', 'ignore')
        EE_titles = EE_titles.decode()
        EE_title_list.append(EE_titles)
```

In [7]:

```
#Translating list of titles into string
DM_title_long_string = " ".join(DM_title_list)
DM_title_string_complete = DM_title_long_string.replace('Jungle Confidential', '')
DM_num_punctuation = len([token for token in DM_title_string_complete if token in punctuations])
DM_num_uppercase = len([word for word in DM_title_string_complete if word.isupper()])
DM_PROP_UPPER = DM_num_uppercase / len(DM_title_string_complete)
DM_tokenised_words = word_tokenize(DM_title_string_complete)
DM_lc_tokens = [i.lower() for i in DM_tokenised_words]
DM_lc_filtered_tokens = [word for word in DM_lc_tokens if word not in punctuations]
DM_lc_fully_filtered_tokens = [word for word in DM_lc_filtered_tokens if not word in stop_words]
```

```
DM_lemmatized_tokens = []
for token in DM_lc_fully_filtered_tokens:
    DM_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
DE_title_string_complete = " ".join(DE_title_list)
DE_num_punctuation = len([token for token in DE_title_string_complete if token in punctuations])
DE_num_uppercase = len([word for word in DE_title_string_complete if word.isupper()])
DE_PROP_UPPER = DE_num_uppercase / len(DE_title_string_complete)
DE_tokenised_words = word_tokenize(DE_title_string_complete)
DE_lc_tokens = [i.lower() for i in DE_tokenised_words]
DE_lc_filtered_tokens = [word for word in DE_lc_tokens if word not in punctuations]
DE_lc_fully_filtered_tokens = [word for word in DE_lc_filtered_tokens if not word in stop_words]
DE_lemmatized_tokens = []
for token in DE_lc_fully_filtered_tokens:
    DE_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
BBC_title_string_complete = " ".join(BBC_title_list)
BBC_num_punctuation = len([token for token in BBC_title_string_complete if token in punctuations])
BBC_num_uppercase = len([word for word in BBC_title_string_complete if word.isupper()])
BBC_PROP_UPPER = BBC_num_uppercase / len(BBC_title_string_complete)
BBC_tokenised_words = word_tokenize(BBC_title_string_complete)
BBC_lc_tokens = [i.lower() for i in BBC_tokenised_words]
BBC_lc_filtered_tokens = [word for word in BBC_lc_tokens if word not in punctuations]
BBC_lc_fully_filtered_tokens = [word for word in BBC_lc_filtered_tokens if not word in stop_words]
BBC_lemmatized_tokens = []
for token in BBC_lc_fully_filtered_tokens:
    BBC_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
FT_title_string_complete = " ".join(FT_title_list)
FT_num_punctuation = len([token for token in FT_title_string_complete if token in punctuations])
FT_num_uppercase = len([word for word in FT_title_string_complete if word.isupper()])
FT_PROP_UPPER = FT_num_uppercase / len(FT_title_string_complete)
FT_tokenised_words = word_tokenize(FT_title_string_complete)
FT_lc_tokens = [i.lower() for i in FT_tokenised_words]
FT_lc_filtered_tokens = [word for word in FT_lc_tokens if word not in punctuations]
FT_lc_fully_filtered_tokens = [word for word in FT_lc_filtered_tokens if not word in stop_words]
FT_lemmatized_tokens = []
for token in FT_lc_fully_filtered_tokens:
    FT_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
```

```
TG_title_long_string = " ".join(TG_title_list)
TG_num_punctuation = len([token for token in TG_title_long_string if token in punctuations])
TG_num_uppercase = len([word for word in TG_title_long_string if word.isupper()])
TG_PROP_UPPER = TG_num_uppercase / len(TG_title_long_string)
TG_tokenised_words = word_tokenize(TG_title_long_string)
TG_lc_tokens = [i.lower() for i in TG_tokenised_words]
TG_lc_filtered_tokens = [word for word in TG_lc_tokens if word not in punctuations]
TG_lc_fully_filtered_tokens = [word for word in TG_lc_filtered_tokens if not word in stop_words]
TG_lemmatized_tokens = []
for token in TG_lc_fully_filtered_tokens:
    TG_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
TM_title_long_string = " ".join(TM_title_list)
TM_num_punctuation = len([token for token in TM_title_long_string if token in punctuations])
TM_num_uppercase = len([word for word in TM_title_long_string if word.isupper()])
TM_PROP_UPPER = TM_num_uppercase / len(TM_title_long_string)
TM_tokenised_words = word_tokenize(TM_title_long_string)
TM_lc_tokens = [i.lower() for i in TM_tokenised_words]
TM_lc_filtered_tokens = [word for word in TM_lc_tokens if word not in punctuations]
TM_lc_fully_filtered_tokens = [word for word in TM_lc_filtered_tokens if not word in stop_words]
TM_lemmatized_tokens = []
for token in TM_lc_fully_filtered_tokens:
    TM_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
TPP_title_long_string = " ".join(TPP_title_list)
TPP_num_punctuation = len([token for token in TPP_title_long_string if token in punctuations])
TPP_num_uppercase = len([word for word in TPP_title_long_string if word.isupper()])
TPP_PROP_UPPER = TPP_num_uppercase / len(TPP_title_long_string)
TPP_tokenised_words = word_tokenize(TPP_title_long_string)
TPP_lc_tokens = [i.lower() for i in TPP_tokenised_words]
TPP_lc_filtered_tokens = [word for word in TPP_lc_tokens if word not in punctuations]
TPP_lc_fully_filtered_tokens = [word for word in TPP_lc_filtered_tokens if not word in stop_words]
TPP_lemmatized_tokens = []
for token in TPP_lc_fully_filtered_tokens:
    TPP_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
PP_title_long_string = " ".join(PP_title_list)
PP_num_punctuation = len([token for token in PP_title_long_string if token in punctuations])
PP_num_uppercase = len([word for word in PP_title_long_string if word.isupper()])
PP_PROP_UPPER = PP_num_uppercase / len(PP_title_long_string)
PP_tokenised_words = word_tokenize(PP_title_long_string)
```

```
PP_lc_tokens = [i.lower() for i in PP_tokenised_words]
PP_lc_filtered_tokens = [word for word in PP_lc_tokens if word not in punctuations]
PP_lc_fully_filtered_tokens = [word for word in PP_lc_filtered_tokens if not word in stop_words]
PP_lemmatized_tokens = []
for token in PP_lc_fully_filtered_tokens:
    PP_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
TDM_title_long_string = " ".join(TDM_title_list)
TDM_num_punctuation = len([token for token in TDM_title_long_string if token in punctuations])
TDM_num_uppercase = len([word for word in TDM_title_long_string if word.isupper()])
TDM_PROP_UPPER = TDM_num_uppercase / len(TDM_title_long_string)
TDM_tokenised_words = word_tokenize(TDM_title_long_string)
TDM_lc_tokens = [i.lower() for i in TDM_tokenised_words]
TDM_lc_filtered_tokens = [word for word in TDM_lc_tokens if word not in punctuations]
TDM_lc_fully_filtered_tokens = [word for word in TDM_lc_filtered_tokens if not word in stop_words]
TDM_lemmatized_tokens = []
for token in TDM_lc_fully_filtered_tokens:
    TDM_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
JLY_title_long_string = " ".join(JLY_title_list)
JLY_num_punctuation = len([token for token in JLY_title_long_string if token in punctuations])
JLY_num_uppercase = len([word for word in JLY_title_long_string if word.isupper()])
JLY_PROP_UPPER = JLY_num_uppercase / len(JLY_title_long_string)
JLY_tokenised_words = word_tokenize(JLY_title_long_string)
JLY_lc_tokens = [i.lower() for i in JLY_tokenised_words]
JLY_lc_filtered_tokens = [word for word in JLY_lc_tokens if word not in punctuations]
JLY_lc_fully_filtered_tokens = [word for word in JLY_lc_filtered_tokens if not word in stop_words]
JLY_lemmatized_tokens = []
for token in JLY_lc_fully_filtered_tokens:
    JLY_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
SM_title_long_string = " ".join(SM_title_list)
SM_num_punctuation = len([token for token in SM_title_long_string if token in punctuations])
SM_num_uppercase = len([word for word in SM_title_long_string if word.isupper()])
SM_PROP_UPPER = SM_num_uppercase / len(SM_title_long_string)
SM_tokenised_words = word_tokenize(SM_title_long_string)
SM_lc_tokens = [i.lower() for i in SM_tokenised_words]
SM_lc_filtered_tokens = [word for word in SM_lc_tokens if word not in punctuations]
SM_lc_fully_filtered_tokens = [word for word in SM_lc_filtered_tokens if not word in stop_words]
SM_lemmatized_tokens = []
for token in SM_lc_fully_filtered_tokens:
```

```
    SM_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
JL_title_long_string = " ".join(JL_title_list)
JL_num_punctuation = len([token for token in JL_title_long_string if token in punctuations])
JL_num_uppercase = len([word for word in JL_title_long_string if word.isupper()])
JL_PROP_UPPER = JL_num_uppercase / len(JL_title_long_string)
JL_tokenised_words = word_tokenize(JL_title_long_string)
JL_lc_tokens = [i.lower() for i in JL_tokenised_words]
JL_lc_filtered_tokens = [word for word in JL_lc_tokens if word not in punctuations]
JL_lc_fully_filtered_tokens = [word for word in JL_lc_filtered_tokens if not word in stop_words]
JL_lemmatized_tokens = []
for token in JL_lc_fully_filtered_tokens:
    JL_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
KG_title_long_string = " ".join(KG_title_list)
KG_num_punctuation = len([token for token in KG_title_long_string if token in punctuations])
KG_num_uppercase = len([word for word in KG_title_long_string if word.isupper()])
KG_PROP_UPPER = KG_num_uppercase / len(KG_title_long_string)
KG_tokenised_words = word_tokenize(KG_title_long_string)
KG_lc_tokens = [i.lower() for i in KG_tokenised_words]
KG_lc_filtered_tokens = [word for word in KG_lc_tokens if word not in punctuations]
KG_lc_fully_filtered_tokens = [word for word in KG_lc_filtered_tokens if not word in stop_words]
KG_lemmatized_tokens = []
for token in KG_lc_fully_filtered_tokens:
    KG_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
OM_title_long_string = " ".join(OM_title_list)
OM_num_punctuation = len([token for token in OM_title_long_string if token in punctuations])
OM_num_uppercase = len([word for word in OM_title_long_string if word.isupper()])
OM_PROP_UPPER = OM_num_uppercase / len(OM_title_long_string)
OM_tokenised_words = word_tokenize(OM_title_long_string)
OM_lc_tokens = [i.lower() for i in OM_tokenised_words]
OM_lc_filtered_tokens = [word for word in OM_lc_tokens if word not in punctuations]
OM_lc_fully_filtered_tokens = [word for word in OM_lc_filtered_tokens if not word in stop_words]
OM_lemmatized_tokens = []
for token in OM_lc_fully_filtered_tokens:
    OM_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
WB_title_long_string = " ".join(WB_title_list)
WB_num_punctuation = len([token for token in WB_title_long_string if token in punctuations])
```

```
WB_num_uppercase = len([word for word in WB_title_long_string if word.isupper()])
WB_PROP_UPPER = WB_num_uppercase / len(WB_title_long_string)
WB_tokenised_words = word_tokenize(WB_title_long_string)
WB_lc_tokens = [i.lower() for i in WB_tokenised_words]
WB_lc_filtered_tokens = [word for word in WB_lc_tokens if word not in punctuations]
WB_lc_fully_filtered_tokens = [word for word in WB_lc_filtered_tokens if not word in stop_words]
WB_lemmatized_tokens = []
for token in WB_lc_fully_filtered_tokens:
    WB_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
STM_title_long_string = " ".join(STM_title_list)
STM_num_punctuation = len([token for token in STM_title_long_string if token in punctuations])
STM_num_uppercase = len([word for word in STM_title_long_string if word.isupper()])
STM_PROP_UPPER = STM_num_uppercase / len(STM_title_long_string)
STM_tokenised_words = word_tokenize(STM_title_long_string)
STM_lc_tokens = [i.lower() for i in STM_tokenised_words]
STM_lc_filtered_tokens = [word for word in STM_lc_tokens if word not in punctuations]
STM_lc_fully_filtered_tokens = [word for word in STM_lc_filtered_tokens if not word in stop_words]
STM_lemmatized_tokens = []
for token in STM_lc_fully_filtered_tokens:
    STM_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
FF_title_long_string = " ".join(FF_title_list)
FF_num_punctuation = len([token for token in FF_title_long_string if token in punctuations])
FF_num_uppercase = len([word for word in FF_title_long_string if word.isupper()])
FF_PROP_UPPER = FF_num_uppercase / len(FF_title_long_string)
FF_tokenised_words = word_tokenize(FF_title_long_string)
FF_lc_tokens = [i.lower() for i in FF_tokenised_words]
FF_lc_filtered_tokens = [word for word in FF_lc_tokens if word not in punctuations]
FF_lc_fully_filtered_tokens = [word for word in FF_lc_filtered_tokens if not word in stop_words]
FF_lemmatized_tokens = []
for token in FF_lc_fully_filtered_tokens:
    FF_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
ICC_title_long_string = " ".join(ICC_title_list)
ICC_num_punctuation = len([token for token in ICC_title_long_string if token in punctuations])
ICC_num_uppercase = len([word for word in ICC_title_long_string if word.isupper()])
ICC_PROP_UPPER = ICC_num_uppercase / len(ICC_title_long_string)
ICC_tokenised_words = word_tokenize(ICC_title_long_string)
ICC_lc_tokens = [i.lower() for i in ICC_tokenised_words]
ICC_lc_filtered_tokens = [word for word in ICC_lc_tokens if word not in punctuations]
```

```
ICC_lc_fully_filtered_tokens = [word for word in ICC_lc_filtered_tokens if not word in stop_words]
ICC_lemmatized_tokens = []
for token in ICC_lc_fully_filtered_tokens:
    ICC_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
BB_title_long_string = " ".join(BB_title_list)
BB_num_punctuation = len([token for token in BB_title_long_string if token in punctuations])
BB_num_uppercase = len([word for word in BB_title_long_string if word.isupper()])
BB_PROP_UPPER = BB_num_uppercase / len(BB_title_long_string)
BB_tokenised_words = word_tokenize(BB_title_long_string)
BB_lc_tokens = [i.lower() for i in BB_tokenised_words]
BB_lc_filtered_tokens = [word for word in BB_lc_tokens if word not in punctuations]
BB_lc_fully_filtered_tokens = [word for word in BB_lc_filtered_tokens if not word in stop_words]
BB_lemmatized_tokens = []
for token in BB_lc_fully_filtered_tokens:
    BB_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
DP_title_long_string = " ".join(DP_title_list)
DP_num_punctuation = len([token for token in DP_title_long_string if token in punctuations])
DP_num_uppercase = len([word for word in DP_title_long_string if word.isupper()])
DP_PROP_UPPER = DP_num_uppercase / len(DP_title_long_string)
DP_tokenised_words = word_tokenize(DP_title_long_string)
DP_lc_tokens = [i.lower() for i in DP_tokenised_words]
DP_lc_filtered_tokens = [word for word in DP_lc_tokens if word not in punctuations]
DP_lc_fully_filtered_tokens = [word for word in DP_lc_filtered_tokens if not word in stop_words]
DP_lemmatized_tokens = []
for token in DP_lc_fully_filtered_tokens:
    DP_lemmatized_tokens.append(lemmatizer.lemmatize(token))

#Translating list of titles into string
CW_title_long_string = " ".join(CW_title_list)
CW_num_punctuation = len([token for token in CW_title_long_string if token in punctuations])
CW_num_uppercase = len([word for word in CW_title_long_string if word.isupper()])
CW_PROP_UPPER = CW_num_uppercase / len(CW_title_long_string)
CW_tokenised_words = word_tokenize(CW_title_long_string)
CW_lc_tokens = [i.lower() for i in CW_tokenised_words]
CW_lc_filtered_tokens = [word for word in CW_lc_tokens if word not in punctuations]
CW_lc_fully_filtered_tokens = [word for word in CW_lc_filtered_tokens if not word in stop_words]
CW_lemmatized_tokens = []
for token in CW_lc_fully_filtered_tokens:
    CW_lemmatized_tokens.append(lemmatizer.lemmatize(token))
```

```
#Translating list of titles into string
CT_title_long_string = " ".join(CT_title_list)
CT_num_punctuation = len([token for token in CT_title_long_string if token in punctuations])
CT_num_uppercase = len([word for word in CT_title_long_string if word.isupper()])
CT_PROP_UPPER = CT_num_uppercase / len(CT_title_long_string)
CT_tokenised_words = word_tokenize(CT_title_long_string)
CT_lc_tokens = [i.lower() for i in CT_tokenised_words]
CT_lc_filtered_tokens = [word for word in CT_lc_tokens if word not in punctuations]
CT_lc_fully_filtered_tokens = [word for word in CT_lc_filtered_tokens if not word in stop_words]
CT_lemmatized_tokens = []
for token in CT_lc_fully_filtered_tokens:
    CT_lemmatized_tokens.append(lemmatizer.lemmatize(token))

NN_title_long_string = " ".join(NN_title_list)
NN_num_punctuation = len([token for token in NN_title_long_string if token in punctuations])
NN_num_uppercase = len([word for word in NN_title_long_string if word.isupper()])
NN_PROP_UPPER = NN_num_uppercase / len(NN_title_long_string)
NN_tokenised_words = word_tokenize(NN_title_long_string)
NN_lc_tokens = [i.lower() for i in NN_tokenised_words]
NN_lc_filtered_tokens = [word for word in NN_lc_tokens if word not in punctuations]
NN_lc_fully_filtered_tokens = [word for word in NN_lc_filtered_tokens if not word in stop_words]
NN_lemmatized_tokens = []
for token in NN_lc_fully_filtered_tokens:
    NN_lemmatized_tokens.append(lemmatizer.lemmatize(token))

WN_title_long_string = " ".join(WN_title_list)
WN_num_punctuation = len([token for token in WN_title_long_string if token in punctuations])
WN_num_uppercase = len([word for word in WN_title_long_string if word.isupper()])
WN_PROP_UPPER = WN_num_uppercase / len(WN_title_long_string)
WN_tokenised_words = word_tokenize(WN_title_long_string)
WN_lc_tokens = [i.lower() for i in WN_tokenised_words]
WN_lc_filtered_tokens = [word for word in WN_lc_tokens if word not in punctuations]
WN_lc_fully_filtered_tokens = [word for word in WN_lc_filtered_tokens if not word in stop_words]
WN_lemmatized_tokens = []
for token in WN_lc_fully_filtered_tokens:
    WN_lemmatized_tokens.append(lemmatizer.lemmatize(token))

JB_title_long_string = " ".join(JB_title_list)
JB_num_punctuation = len([token for token in JB_title_long_string if token in punctuations])
JB_num_uppercase = len([word for word in JB_title_long_string if word.isupper()])
JB_PROP_UPPER = JB_num_uppercase / len(JB_title_long_string)
JB_tokenised_words = word_tokenize(JB_title_long_string)
JB_lc_tokens = [i.lower() for i in JB_tokenised_words]
JB_lc_filtered_tokens = [word for word in JB_lc_tokens if word not in punctuations]
```

```
JB_lc_fully_filtered_tokens = [word for word in JB_lc_filtered_tokens if not word in stop_words]
JB_lemmatized_tokens = []
for token in JB_lc_fully_filtered_tokens:
    JB_lemmatized_tokens.append(lemmatizer.lemmatize(token))

MK_title_long_string = " ".join(MK_title_list)
MK_num_punctuation = len([token for token in MK_title_long_string if token in punctuations])
MK_num_uppercase = len([word for word in MK_title_long_string if word.isupper()])
MK_PROP_UPPER = MK_num_uppercase / len(MK_title_long_string)
MK_tokenised_words = word_tokenize(MK_title_long_string)
MK_lc_tokens = [i.lower() for i in MK_tokenised_words]
MK_lc_filtered_tokens = [word for word in MK_lc_tokens if word not in punctuations]
MK_lc_fully_filtered_tokens = [word for word in MK_lc_filtered_tokens if not word in stop_words]
MK_lemmatized_tokens = []
for token in MK_lc_fully_filtered_tokens:
    MK_lemmatized_tokens.append(lemmatizer.lemmatize(token))

DR_title_long_string = " ".join(DR_title_list)
DR_num_punctuation = len([token for token in DR_title_long_string if token in punctuations])
DR_num_uppercase = len([word for word in DR_title_long_string if word.isupper()])
DR_PROP_UPPER = DR_num_uppercase / len(DR_title_long_string)
DR_tokenised_words = word_tokenize(DR_title_long_string)
DR_lc_tokens = [i.lower() for i in DR_tokenised_words]
DR_lc_filtered_tokens = [word for word in DR_lc_tokens if word not in punctuations]
DR_lc_fully_filtered_tokens = [word for word in DR_lc_filtered_tokens if not word in stop_words]
DR_lemmatized_tokens = []
for token in DR_lc_fully_filtered_tokens:
    DR_lemmatized_tokens.append(lemmatizer.lemmatize(token))

BGT_title_long_string = " ".join(BGT_title_list)
BGT_num_punctuation = len([token for token in BGT_title_long_string if token in punctuations])
BGT_num_uppercase = len([word for word in BGT_title_long_string if word.isupper()])
BGT_PROP_UPPER = BGT_num_uppercase / len(BGT_title_long_string)
BGT_tokenised_words = word_tokenize(BGT_title_long_string)
BGT_lc_tokens = [i.lower() for i in BGT_tokenised_words]
BGT_lc_filtered_tokens = [word for word in BGT_lc_tokens if word not in punctuations]
BGT_lc_fully_filtered_tokens = [word for word in BGT_lc_filtered_tokens if not word in stop_words]
BGT_lemmatized_tokens = []
for token in BGT_lc_fully_filtered_tokens:
    BGT_lemmatized_tokens.append(lemmatizer.lemmatize(token))

DH_title_long_string = " ".join(DH_title_list)
DH_num_punctuation = len([token for token in DH_title_long_string if token in punctuations])
DH_num_uppercase = len([word for word in DH_title_long_string if word.isupper()])
```

```
DH_PROP_UPPER = DH_num_uppercase / len(DH_title_long_string)
DH_tokenised_words = word_tokenize(DH_title_long_string)
DH_lc_tokens = [i.lower() for i in DH_tokenised_words]
DH_lc_filtered_tokens = [word for word in DH_lc_tokens if word not in punctuations]
DH_lc_fully_filtered_tokens = [word for word in DH_lc_filtered_tokens if not word in stop_words]
DH_lemmatized_tokens = []
for token in DH_lc_fully_filtered_tokens:
    DH_lemmatized_tokens.append(lemmatizer.lemmatize(token))

TT_title_long_string = " ".join(TT_title_list)
TT_num_punctuation = len([token for token in TT_title_long_string if token in punctuations])
TT_num_uppercase = len([word for word in TT_title_long_string if word.isupper()])
TT_PROP_UPPER = TT_num_uppercase / len(TT_title_long_string)
TT_tokenised_words = word_tokenize(TT_title_long_string)
TT_lc_tokens = [i.lower() for i in TT_tokenised_words]
TT_lc_filtered_tokens = [word for word in TT_lc_tokens if word not in punctuations]
TT_lc_fully_filtered_tokens = [word for word in TT_lc_filtered_tokens if not word in stop_words]
TT_lemmatized_tokens = []
for token in TT_lc_fully_filtered_tokens:
    TT_lemmatized_tokens.append(lemmatizer.lemmatize(token))

NC_title_long_string = " ".join(NC_title_list)
NC_num_punctuation = len([token for token in NC_title_long_string if token in punctuations])
NC_num_uppercase = len([word for word in NC_title_long_string if word.isupper()])
NC_PROP_UPPER = NC_num_uppercase / len(NC_title_long_string)
NC_tokenised_words = word_tokenize(NC_title_long_string)
NC_lc_tokens = [i.lower() for i in NC_tokenised_words]
NC_lc_filtered_tokens = [word for word in NC_lc_tokens if word not in punctuations]
NC_lc_fully_filtered_tokens = [word for word in NC_lc_filtered_tokens if not word in stop_words]
NC_lemmatized_tokens = []
for token in NC_lc_fully_filtered_tokens:
    NC_lemmatized_tokens.append(lemmatizer.lemmatize(token))

CP_title_long_string = " ".join(CP_title_list)
CP_num_punctuation = len([token for token in CP_title_long_string if token in punctuations])
CP_num_uppercase = len([word for word in CP_title_long_string if word.isupper()])
CP_PROP_UPPER = CP_num_uppercase / len(CP_title_long_string)
CP_tokenised_words = word_tokenize(CP_title_long_string)
CP_lc_tokens = [i.lower() for i in CP_tokenised_words]
CP_lc_filtered_tokens = [word for word in CP_lc_tokens if word not in punctuations]
CP_lc_fully_filtered_tokens = [word for word in CP_lc_filtered_tokens if not word in stop_words]
CP_lemmatized_tokens = []
for token in CP_lc_fully_filtered_tokens:
    CP_lemmatized_tokens.append(lemmatizer.lemmatize(token))
```

```
TP_title_long_string = " ".join(TP_title_list)
TP_num_punctuation = len([token for token in TP_title_long_string if token in punctuations])
TP_num_uppercase = len([word for word in TP_title_long_string if word.isupper()])
TP_PROP_UPPER = TP_num_uppercase / len(TP_title_long_string)
TP_tokenised_words = word_tokenize(TP_title_long_string)
TP_lc_tokens = [i.lower() for i in TP_tokenised_words]
TP_lc_filtered_tokens = [word for word in TP_lc_tokens if word not in punctuations]
TP_lc_fully_filtered_tokens = [word for word in TP_lc_filtered_tokens if not word in stop_words]
TP_lemmatized_tokens = []
for token in TP_lc_fully_filtered_tokens:
    TP_lemmatized_tokens.append(lemmatizer.lemmatize(token))

EE_title_long_string = " ".join(EE_title_list)
EE_num_punctuation = len([token for token in EE_title_long_string if token in punctuations])
EE_num_uppercase = len([word for word in EE_title_long_string if word.isupper()])
EE_PROP_UPPER = EE_num_uppercase / len(EE_title_long_string)
EE_tokenised_words = word_tokenize(EE_title_long_string)
EE_lc_tokens = [i.lower() for i in EE_tokenised_words]
EE_lc_filtered_tokens = [word for word in EE_lc_tokens if word not in punctuations]
EE_lc_fully_filtered_tokens = [word for word in EE_lc_filtered_tokens if not word in stop_words]
EE_lemmatized_tokens = []
for token in EE_lc_fully_filtered_tokens:
    EE_lemmatized_tokens.append(lemmatizer.lemmatize(token))
```

In [8]:

```
#Word counts
DM_word_counts = pd.Series(Counter(DM_lemmatized_tokens))
DM_word_counts.sort_values()
DM_df = pd.DataFrame()
DM_df['DM_Lemma_Counts'] = DM_word_counts

#Word counts
DE_word_counts = pd.Series(Counter(DE_lemmatized_tokens))
DE_word_counts.sort_values()
DE_df = pd.DataFrame()
DE_df['DE_Lemma_Counts'] = DE_word_counts

#Word counts
BBC_word_counts = pd.Series(Counter(BBC_lemmatized_tokens))
BBC_word_counts.sort_values()
BBC_df = pd.DataFrame()
BBC_df['BBC_Lemma_Counts'] = BBC_word_counts
```

```
#Word counts
FT_word_counts = pd.Series(Counter(FT_lemmatized_tokens))
FT_word_counts.sort_values()
FT_df = pd.DataFrame()
FT_df['FT_Lemma_Counts'] = FT_word_counts

#Word counts
TG_word_counts = pd.Series(Counter(TG_lemmatized_tokens))
TG_word_counts.sort_values()
TG_df = pd.DataFrame()
TG_df['TG_Lemma_Counts'] = TG_word_counts

#Word counts
TM_word_counts = pd.Series(Counter(TM_lemmatized_tokens))
TM_word_counts.sort_values()
TM_df = pd.DataFrame()
TM_df['TM_Lemma_Counts'] = TM_word_counts

TPP_word_counts = pd.Series(Counter(TPP_lemmatized_tokens))
TPP_word_counts.sort_values()
TPP_df = pd.DataFrame()
TPP_df['TPP_Lemma_Counts'] = TPP_word_counts

PP_word_counts = pd.Series(Counter(PP_lemmatized_tokens))
PP_word_counts.sort_values()
PP_df = pd.DataFrame()
PP_df['PP_Lemma_Counts'] = PP_word_counts

TDM_word_counts = pd.Series(Counter(TDM_lemmatized_tokens))
TDM_word_counts.sort_values()
TDM_df = pd.DataFrame()
TDM_df['TDM_Lemma_Counts'] = TDM_word_counts

JLY_word_counts = pd.Series(Counter(JLY_lemmatized_tokens))
JLY_word_counts.sort_values()
JLY_df = pd.DataFrame()
JLY_df['JLY_Lemma_Counts'] = JLY_word_counts

SM_word_counts = pd.Series(Counter(SM_lemmatized_tokens))
SM_word_counts.sort_values()
SM_df = pd.DataFrame()
SM_df['SM_Lemma_Counts'] = SM_word_counts

JL_word_counts = pd.Series(Counter(JL_lemmatized_tokens))
```

```
JL_word_counts.sort_values()
JL_df = pd.DataFrame()
JL_df['JL_Lemma_Counts'] = JL_word_counts

KG_word_counts = pd.Series(Counter(KG_lemmatized_tokens))
KG_word_counts.sort_values()
KG_df = pd.DataFrame()
KG_df['KG_Lemma_Counts'] = KG_word_counts

OM_word_counts = pd.Series(Counter(OM_lemmatized_tokens))
OM_word_counts.sort_values()
OM_df = pd.DataFrame()
OM_df['OM_Lemma_Counts'] = OM_word_counts

WB_word_counts = pd.Series(Counter(WB_lemmatized_tokens))
WB_word_counts.sort_values()
WB_df = pd.DataFrame()
WB_df['WB_Lemma_Counts'] = WB_word_counts

STM_word_counts = pd.Series(Counter(STM_lemmatized_tokens))
STM_word_counts.sort_values()
STM_df = pd.DataFrame()
STM_df['STM_Lemma_Counts'] = STM_word_counts

FF_word_counts = pd.Series(Counter(FF_lemmatized_tokens))
FF_word_counts.sort_values()
FF_df = pd.DataFrame()
FF_df['FF_Lemma_Counts'] = FF_word_counts

ICC_word_counts = pd.Series(Counter(ICC_lemmatized_tokens))
ICC_word_counts.sort_values()
ICC_df = pd.DataFrame()
ICC_df['ICC_Lemma_Counts'] = ICC_word_counts

BB_word_counts = pd.Series(Counter(BB_lemmatized_tokens))
BB_word_counts.sort_values()
BB_df = pd.DataFrame()
BB_df['BB_Lemma_Counts'] = BB_word_counts

DP_word_counts = pd.Series(Counter(DP_lemmatized_tokens))
DP_word_counts.sort_values()
DP_df = pd.DataFrame()
DP_df['DP_Lemma_Counts'] = DP_word_counts
```

```
CW_word_counts = pd.Series(Counter(CW_lemmatized_tokens))
CW_word_counts.sort_values()
CW_df = pd.DataFrame()
CW_df['CW_Lemma_Counts'] = CW_word_counts

CT_word_counts = pd.Series(Counter(CT_lemmatized_tokens))
CT_word_counts.sort_values()
CT_df = pd.DataFrame()
CT_df['CT_Lemma_Counts'] = CT_word_counts

NN_word_counts = pd.Series(Counter(NN_lemmatized_tokens))
NN_word_counts.sort_values()
NN_df = pd.DataFrame()
NN_df['NN_Lemma_Counts'] = NN_word_counts

WN_word_counts = pd.Series(Counter(WN_lemmatized_tokens))
WN_word_counts.sort_values()
WN_df = pd.DataFrame()
WN_df['WN_Lemma_Counts'] = WN_word_counts

JB_word_counts = pd.Series(Counter(JB_lemmatized_tokens))
JB_word_counts.sort_values()
JB_df = pd.DataFrame()
JB_df['JB_Lemma_Counts'] = JB_word_counts

MK_word_counts = pd.Series(Counter(MK_lemmatized_tokens))
MK_word_counts.sort_values()
MK_df = pd.DataFrame()
MK_df['MK_Lemma_Counts'] = MK_word_counts

DR_word_counts = pd.Series(Counter(DR_lemmatized_tokens))
DR_word_counts.sort_values()
DR_df = pd.DataFrame()
DR_df['DR_Lemma_Counts'] = DR_word_counts

BGT_word_counts = pd.Series(Counter(BGT_lemmatized_tokens))
BGT_word_counts.sort_values()
BGT_df = pd.DataFrame()
BGT_df['BGT_Lemma_Counts'] = BGT_word_counts

DH_word_counts = pd.Series(Counter(DH_lemmatized_tokens))
DH_word_counts.sort_values()
DH_df = pd.DataFrame()
DH_df['DH_Lemma_Counts'] = DH_word_counts
```

```
TT_word_counts = pd.Series(Counter(TT_lemmatized_tokens))
TT_word_counts.sort_values()
TT_df = pd.DataFrame()
TT_df['TT_Lemma_Counts'] = TT_word_counts

NC_word_counts = pd.Series(Counter(NC_lemmatized_tokens))
NC_word_counts.sort_values()
NC_df = pd.DataFrame()
NC_df['NC_Lemma_Counts'] = NC_word_counts

CP_word_counts = pd.Series(Counter(CP_lemmatized_tokens))
CP_word_counts.sort_values()
CP_df = pd.DataFrame()
CP_df['CP_Lemma_Counts'] = CP_word_counts

TP_word_counts = pd.Series(Counter(TP_lemmatized_tokens))
TP_word_counts.sort_values()
TP_df = pd.DataFrame()
TP_df['TP_Lemma_Counts'] = TP_word_counts

EE_word_counts = pd.Series(Counter(EE_lemmatized_tokens))
EE_word_counts.sort_values()
EE_df = pd.DataFrame()
EE_df['EE_Lemma_Counts'] = EE_word_counts
```

In [9]:

```
#Gather VAD data
DM_words = []
DM_emo = []
for i in DM_lemmatized_tokens:
    if i in vad.index:
        DM_emo.append(vad.loc[i])
        DM_words.append(i)
    else:
        pass
DM_emo_df = pd.DataFrame(DM_emo, index = DM_words)

#Gather VAD data
DE_words = []
DE_emo = []
for i in DE_lemmatized_tokens:
    if i in vad.index:
        DE_emo.append(vad.loc[i])
```

```
        DE_words.append(i)
    else:
        pass
DE_emo_df = pd.DataFrame(DE_emo, index = DE_words)

#Gather VAD data
BBC_words = []
BBC_emo = []
for i in BBC_lemmatized_tokens:
    if i in vad.index:
        BBC_emo.append(vad.loc[i])
        BBC_words.append(i)
    else:
        pass
BBC_emo_df = pd.DataFrame(BBC_emo, index = BBC_words)

#Gather VAD data
FT_words = []
FT_emo = []
for i in FT_lemmatized_tokens:
    if i in vad.index:
        FT_emo.append(vad.loc[i])
        FT_words.append(i)
    else:
        pass
FT_emo_df = pd.DataFrame(FT_emo, index = FT_words)

#Gather VAD data
TG_words = []
TG_emo = []
for i in TG_lemmatized_tokens:
    if i in vad.index:
        TG_emo.append(vad.loc[i])
        TG_words.append(i)
    else:
        pass
TG_emo_df = pd.DataFrame(TG_emo, index = TG_words)

#Gather VAD data
TM_words = []
TM_emo = []
for i in TM_lemmatized_tokens:
    if i in vad.index:
        TM_emo.append(vad.loc[i])
```

```
        TM_words.append(i)
    else:
        pass
TM_emo_df = pd.DataFrame(TM_emo, index = TM_words)

TPP_words = []
TPP_emo = []
for i in TPP_lemmatized_tokens:
    if i in vad.index:
        TPP_emo.append(vad.loc[i])
        TPP_words.append(i)
    else:
        pass
TPP_emo_df = pd.DataFrame(TPP_emo, index=TPP_words)

PP_words = []
PP_emo = []
for i in PP_lemmatized_tokens:
    if i in vad.index:
        PP_emo.append(vad.loc[i])
        PP_words.append(i)
    else:
        pass
PP_emo_df = pd.DataFrame(PP_emo, index=PP_words)

TDM_words = []
TDM_emo = []
for i in TDM_lemmatized_tokens:
    if i in vad.index:
        TDM_emo.append(vad.loc[i])
        TDM_words.append(i)
    else:
        pass
TDM_emo_df = pd.DataFrame(TDM_emo, index=TDM_words)

JLY_words = []
JLY_emo = []
for i in JLY_lemmatized_tokens:
    if i in vad.index:
        JLY_emo.append(vad.loc[i])
        JLY_words.append(i)
    else:
        pass
JLY_emo_df = pd.DataFrame(JLY_emo, index=JLY_words)
```

```
SM_words = []
SM_emo = []
for i in SM_lemmatized_tokens:
    if i in vad.index:
        SM_emo.append(vad.loc[i])
        SM_words.append(i)
    else:
        pass
SM_emo_df = pd.DataFrame(SM_emo, index=SM_words)

JL_words = []
JL_emo = []
for i in JL_lemmatized_tokens:
    if i in vad.index:
        JL_emo.append(vad.loc[i])
        JL_words.append(i)
    else:
        pass
JL_emo_df = pd.DataFrame(JL_emo, index=JL_words)

KG_words = []
KG_emo = []
for i in KG_lemmatized_tokens:
    if i in vad.index:
        KG_emo.append(vad.loc[i])
        KG_words.append(i)
    else:
        pass
KG_emo_df = pd.DataFrame(KG_emo, index=KG_words)

OM_words = []
OM_emo = []
for i in OM_lemmatized_tokens:
    if i in vad.index:
        OM_emo.append(vad.loc[i])
        OM_words.append(i)
    else:
        pass
OM_emo_df = pd.DataFrame(OM_emo, index=OM_words)

WB_words = []
WB_emo = []
for i in WB_lemmatized_tokens:
```

```
    if i in vad.index:
        WB_emo.append(vad.loc[i])
        WB_words.append(i)
    else:
        pass
WB_emo_df = pd.DataFrame(WB_emo, index=WB_words)

STM_words = []
STM_emo = []
for i in STM_lemmatized_tokens:
    if i in vad.index:
        STM_emo.append(vad.loc[i])
        STM_words.append(i)
    else:
        pass
STM_emo_df = pd.DataFrame(STM_emo, index=STM_words)

FF_words = []
FF_emo = []
for i in FF_lemmatized_tokens:
    if i in vad.index:
        FF_emo.append(vad.loc[i])
        FF_words.append(i)
    else:
        pass
FF_emo_df = pd.DataFrame(FF_emo, index=FF_words)

ICC_words = []
ICC_emo = []
for i in ICC_lemmatized_tokens:
    if i in vad.index:
        ICC_emo.append(vad.loc[i])
        ICC_words.append(i)
    else:
        pass
ICC_emo_df = pd.DataFrame(ICC_emo, index=ICC_words)

BB_words = []
BB_emo = []
for i in BB_lemmatized_tokens:
    if i in vad.index:
        BB_emo.append(vad.loc[i])
        BB_words.append(i)
    else:
```

```
        pass
BB_emo_df = pd.DataFrame(BB_emo, index=BB_words)

DP_words = []
DP_emo = []
for i in DP_lemmatized_tokens:
    if i in vad.index:
        DP_emo.append(vad.loc[i])
        DP_words.append(i)
    else:
        pass
DP_emo_df = pd.DataFrame(DP_emo, index=DP_words)

CW_words = []
CW_emo = []
for i in CW_lemmatized_tokens:
    if i in vad.index:
        CW_emo.append(vad.loc[i])
        CW_words.append(i)
    else:
        pass
CW_emo_df = pd.DataFrame(CW_emo, index=CW_words)

CT_words = []
CT_emo = []
for i in CT_lemmatized_tokens:
    if i in vad.index:
        CT_emo.append(vad.loc[i])
        CT_words.append(i)
    else:
        pass
CT_emo_df = pd.DataFrame(CT_emo, index=CT_words)

NN_words = []
NN_emo = []
for i in NN_lemmatized_tokens:
    if i in vad.index:
        NN_emo.append(vad.loc[i])
        NN_words.append(i)
    else:
        pass
NN_emo_df = pd.DataFrame(NN_emo, index=NN_words)

WN_words = []
```

```
WN_emo = []
for i in WN_lemmatized_tokens:
    if i in vad.index:
        WN_emo.append(vad.loc[i])
        WN_words.append(i)
    else:
        pass
WN_emo_df = pd.DataFrame(WN_emo, index=WN_words)

JB_words = []
JB_emo = []
for i in JB_lemmatized_tokens:
    if i in vad.index:
        JB_emo.append(vad.loc[i])
        JB_words.append(i)
    else:
        pass
JB_emo_df = pd.DataFrame(JB_emo, index=JB_words)

MK_words = []
MK_emo = []
for i in MK_lemmatized_tokens:
    if i in vad.index:
        MK_emo.append(vad.loc[i])
        MK_words.append(i)
    else:
        pass
MK_emo_df = pd.DataFrame(MK_emo, index=MK_words)

DR_words = []
DR_emo = []
for i in DR_lemmatized_tokens:
    if i in vad.index:
        DR_emo.append(vad.loc[i])
        DR_words.append(i)
    else:
        pass
DR_emo_df = pd.DataFrame(DR_emo, index=DR_words)

BGT_words = []
BGT_emo = []
for i in BGT_lemmatized_tokens:
    if i in vad.index:
        BGT_emo.append(vad.loc[i])
```

```
        BGT_words.append(i)
    else:
        pass
BGT_emo_df = pd.DataFrame(BGT_emo, index=BGT_words)

DH_words = []
DH_emo = []
for i in DH_lemmatized_tokens:
    if i in vad.index:
        DH_emo.append(vad.loc[i])
        DH_words.append(i)
    else:
        pass
DH_emo_df = pd.DataFrame(DH_emo, index=DH_words)

TT_words = []
TT_emo = []
for i in TT_lemmatized_tokens:
    if i in vad.index:
        TT_emo.append(vad.loc[i])
        TT_words.append(i)
    else:
        pass
TT_emo_df = pd.DataFrame(TT_emo, index=TT_words)

NC_words = []
NC_emo = []
for i in NC_lemmatized_tokens:
    if i in vad.index:
        NC_emo.append(vad.loc[i])
        NC_words.append(i)
    else:
        pass
NC_emo_df = pd.DataFrame(NC_emo, index=NC_words)

CP_words = []
CP_emo = []
for i in CP_lemmatized_tokens:
    if i in vad.index:
        CP_emo.append(vad.loc[i])
        CP_words.append(i)
    else:
        pass
CP_emo_df = pd.DataFrame(CP_emo, index=CP_words)
```

```
TP_words = []
TP_emo = []
for i in TP_lemmatized_tokens:
    if i in vad.index:
        TP_emo.append(vad.loc[i])
        TP_words.append(i)
    else:
        pass
TP_emo_df = pd.DataFrame(TP_emo, index=TP_words)

EE_words = []
EE_emo = []
for i in EE_lemmatized_tokens:
    if i in vad.index:
        EE_emo.append(vad.loc[i])
        EE_words.append(i)
    else:
        pass
EE_emo_df = pd.DataFrame(EE_emo, index=EE_words)
```

In [10]:
```
#Shifting index across on them all
DM_emo_df = DM_emo_df.reset_index().rename(columns={'index': 'word'})
DM_emo_df['Publication'] = 'The Daily Mail'
DM_emo_df['Political Compass/Theme'] = 'Right-Wing'

DE_emo_df = DE_emo_df.reset_index().rename(columns={'index': 'word'})
DE_emo_df['Publication'] = 'The Daily Express'
DE_emo_df['Political Compass/Theme'] = 'Right-Wing'

BBC_emo_df = BBC_emo_df.reset_index().rename(columns={'index': 'word'})
BBC_emo_df['Publication'] = 'The BBC'
BBC_emo_df['Political Compass/Theme'] = 'Centrist'

FT_emo_df = FT_emo_df.reset_index().rename(columns={'index': 'word'})
FT_emo_df['Publication'] = 'The Financial Times'
FT_emo_df['Political Compass/Theme'] = 'Centrist'

TG_emo_df = TG_emo_df.reset_index().rename(columns={'index': 'word'})
TG_emo_df['Publication'] = 'The Guardian'
TG_emo_df['Political Compass/Theme'] = 'Left-Wing'

TM_emo_df = TM_emo_df.reset_index().rename(columns={'index': 'word'})
```

```
TM_emo_df['Publication'] = 'The Mirror'
TM_emo_df['Political Compass/Theme'] = 'Left-Wing'


TPP_emo_df = TPP_emo_df.reset_index().rename(columns={'index': 'word'})
TPP_emo_df['Publication'] = 'The Pink Panther'
TPP_emo_df['Political Compass/Theme'] = 'Film & Animation'

PP_emo_df = PP_emo_df.reset_index().rename(columns={'index': 'word'})
PP_emo_df['Publication'] = 'Percy The Pig'
PP_emo_df['Political Compass/Theme'] = 'Film & Animation'

TDM_emo_df = TDM_emo_df.reset_index().rename(columns={'index': 'word'})
TDM_emo_df['Publication'] = 'DanTDM'
TDM_emo_df['Political Compass/Theme'] = 'Gaming'

JLY_emo_df = JLY_emo_df.reset_index().rename(columns={'index': 'word'})
JLY_emo_df['Publication'] = 'Jelly'
JLY_emo_df['Political Compass/Theme'] = 'Gaming'

SM_emo_df = SM_emo_df.reset_index().rename(columns={'index': 'word'})
SM_emo_df['Publication'] = 'The Sidemen'
SM_emo_df['Political Compass/Theme'] = 'People & Blogs'

JL_emo_df = JL_emo_df.reset_index().rename(columns={'index': 'word'})
JL_emo_df['Publication'] = 'Jeremy Lynch'
JL_emo_df['Political Compass/Theme'] = 'People & Blogs'

KG_emo_df = KG_emo_df.reset_index().rename(columns={'index': 'word'})
KG_emo_df['Publication'] = 'K\'eyush The Stunt Dog'
KG_emo_df['Political Compass/Theme'] = 'Animals & Nature'

OM_emo_df = OM_emo_df.reset_index().rename(columns={'index': 'word'})
OM_emo_df['Publication'] = 'One Minute Animals'
OM_emo_df['Political Compass/Theme'] = 'Animals & Nature'

WB_emo_df = WB_emo_df.reset_index().rename(columns={'index': 'word'})
WB_emo_df['Publication'] = 'Mr. Who\'s Boss'
WB_emo_df['Political Compass/Theme'] = 'Science & Tech'

STM_emo_df = STM_emo_df.reset_index().rename(columns={'index': 'word'})
STM_emo_df['Publication'] = 'Steve Mould'
STM_emo_df['Political Compass/Theme'] = 'Science & Tech'
```

```
FF_emo_df = FF_emo_df.reset_index().rename(columns={'index': 'word'})
FF_emo_df['Publication'] = 'F2Freestyle'
FF_emo_df['Political Compass/Theme'] = 'Sport'

ICC_emo_df = ICC_emo_df.reset_index().rename(columns={'index': 'word'})
ICC_emo_df['Publication'] = 'F2Freestyle'
ICC_emo_df['Political Compass/Theme'] = 'Sport'

BB_emo_df = BB_emo_df.reset_index().rename(columns={'index': 'word'})
BB_emo_df['Publication'] = 'Bald & Bankrupt'
BB_emo_df['Political Compass/Theme'] = 'Travel'

DP_emo_df = DP_emo_df.reset_index().rename(columns={'index': 'word'})
DP_emo_df['Publication'] = 'Dale Philip'
DP_emo_df['Political Compass/Theme'] = 'Travel'

CW_emo_df = CW_emo_df.reset_index().rename(columns={'index': 'word'})
CW_emo_df['Publication'] = 'Carwow'
CW_emo_df['Political Compass/Theme'] = 'Automotive'

CT_emo_df = CT_emo_df.reset_index().rename(columns={'index': 'word'})
CT_emo_df['Publication'] = 'Car Throttle'
CT_emo_df['Political Compass/Theme'] = 'Automotive'

NN_emo_df = NN_emo_df.reset_index().rename(columns={'index': 'word'})
NN_emo_df['Publication'] = 'Mr. Nigel Ng'
NN_emo_df['Political Compass/Theme'] = 'Comedy'

WN_emo_df = WN_emo_df.reset_index().rename(columns={'index': 'word'})
WN_emo_df['Publication'] = 'WillNE'
WN_emo_df['Political Compass/Theme'] = 'Comedy'

JB_emo_df = JB_emo_df.reset_index().rename(columns={'index': 'word'})
JB_emo_df['Publication'] = 'Jungle Beat'
JB_emo_df['Political Compass/Theme'] = 'Education'

MK_emo_df = MK_emo_df.reset_index().rename(columns={'index': 'word'})
MK_emo_df['Publication'] = 'Moonbug Kid'
MK_emo_df['Political Compass/Theme'] = 'Education'

DR_emo_df = DR_emo_df.reset_index().rename(columns={'index': 'word'})
DR_emo_df['Publication'] = 'Dan Rhoades'
DR_emo_df['Political Compass/Theme'] = 'Entertainment'
```

```
BGT_emo_df = BGT_emo_df.reset_index().rename(columns={'index': 'word'})
BGT_emo_df['Publication'] = 'Britain\'s Got Talent'
BGT_emo_df['Political Compass/Theme'] = 'Entertainment'

DH_emo_df = DH_emo_df.reset_index().rename(columns={'index': 'word'})
DH_emo_df['Publication'] = 'DaveHax'
DH_emo_df['Political Compass/Theme'] = 'Lifestyle/How-To'

TT_emo_df = TT_emo_df.reset_index().rename(columns={'index': 'word'})
TT_emo_df['Publication'] = 'Tool Tips'
TT_emo_df['Political Compass/Theme'] = 'Lifestyle/How-To'

NC_emo_df = NC_emo_df.reset_index().rename(columns={'index': 'word'})
NC_emo_df['Publication'] = 'No Copyright Sounds'
NC_emo_df['Political Compass/Theme'] = 'Music'

CP_emo_df = CP_emo_df.reset_index().rename(columns={'index': 'word'})
CP_emo_df['Publication'] = 'Coldplay'
CP_emo_df['Political Compass/Theme'] = 'Music'

TP_emo_df = TP_emo_df.reset_index().rename(columns={'index': 'word'})
TP_emo_df['Publication'] = 'The Two Preachers'
TP_emo_df['Political Compass/Theme'] = 'Non-Profit & Charity'

EE_emo_df = EE_emo_df.reset_index().rename(columns={'index': 'word'})
EE_emo_df['Publication'] = 'The Two Preachers'
EE_emo_df['Political Compass/Theme'] = 'Non-Profit & Charity'

#Finally bringing them together
Mega_VAD_Df = pd.concat([DM_emo_df, DE_emo_df, BBC_emo_df, FT_emo_df, TG_emo_df, TM_emo_df, TPP_emo_df, PP_emo_df, TDM
Mega_VAD_Df = Mega_VAD_Df.reset_index(drop=True)
```

In [60]:
```
Mega_VAD_Df.to_csv('All_words_publishers_categories_and_VAD_scores.csv', index=False)
```

In [11]:
```
#Calculating means of VAD scores
DM_avg_v = DM_emo_df['valence'].mean()
DM_avg_a = DM_emo_df['arousal'].mean()
DM_avg_d = DM_emo_df['dominance'].mean()

DE_avg_v = DE_emo_df['valence'].mean()
DE_avg_a = DE_emo_df['arousal'].mean()
DE_avg_d = DE_emo_df['dominance'].mean()
```

```
BBC_avg_v = BBC_emo_df['valence'].mean()
BBC_avg_a = BBC_emo_df['arousal'].mean()
BBC_avg_d = BBC_emo_df['dominance'].mean()

FT_avg_v = FT_emo_df['valence'].mean()
FT_avg_a = FT_emo_df['arousal'].mean()
FT_avg_d = FT_emo_df['dominance'].mean()

TG_avg_v = TG_emo_df['valence'].mean()
TG_avg_a = TG_emo_df['arousal'].mean()
TG_avg_d = TG_emo_df['dominance'].mean()

TM_avg_v = TM_emo_df['valence'].mean()
TM_avg_a = TM_emo_df['arousal'].mean()
TM_avg_d = TM_emo_df['dominance'].mean()

TPP_avg_v = TPP_emo_df['valence'].mean()
TPP_avg_a = TPP_emo_df['arousal'].mean()
TPP_avg_d = TPP_emo_df['dominance'].mean()

PP_avg_v = PP_emo_df['valence'].mean()
PP_avg_a = PP_emo_df['arousal'].mean()
PP_avg_d = PP_emo_df['dominance'].mean()

TDM_avg_v = TDM_emo_df['valence'].mean()
TDM_avg_a = TDM_emo_df['arousal'].mean()
TDM_avg_d = TDM_emo_df['dominance'].mean()

JLY_avg_v = JLY_emo_df['valence'].mean()
JLY_avg_a = JLY_emo_df['arousal'].mean()
JLY_avg_d = JLY_emo_df['dominance'].mean()

JL_avg_v = JL_emo_df['valence'].mean()
JL_avg_a = JL_emo_df['arousal'].mean()
JL_avg_d = JL_emo_df['dominance'].mean()

SM_avg_v = SM_emo_df['valence'].mean()
SM_avg_a = SM_emo_df['arousal'].mean()
SM_avg_d = SM_emo_df['dominance'].mean()

KG_avg_v = KG_emo_df['valence'].mean()
KG_avg_a = KG_emo_df['arousal'].mean()
KG_avg_d = KG_emo_df['dominance'].mean()
```

```
OM_avg_v = OM_emo_df['valence'].mean()
OM_avg_a = OM_emo_df['arousal'].mean()
OM_avg_d = OM_emo_df['dominance'].mean()

WB_avg_v = WB_emo_df['valence'].mean()
WB_avg_a = WB_emo_df['arousal'].mean()
WB_avg_d = WB_emo_df['dominance'].mean()

STM_avg_v = STM_emo_df['valence'].mean()
STM_avg_a = STM_emo_df['arousal'].mean()
STM_avg_d = STM_emo_df['dominance'].mean()

FF_avg_v = FF_emo_df['valence'].mean()
FF_avg_a = FF_emo_df['arousal'].mean()
FF_avg_d = FF_emo_df['dominance'].mean()

ICC_avg_v = ICC_emo_df['valence'].mean()
ICC_avg_a = ICC_emo_df['arousal'].mean()
ICC_avg_d = ICC_emo_df['dominance'].mean()

BB_avg_v = BB_emo_df['valence'].mean()
BB_avg_a = BB_emo_df['arousal'].mean()
BB_avg_d = BB_emo_df['dominance'].mean()

DP_avg_v = DP_emo_df['valence'].mean()
DP_avg_a = DP_emo_df['arousal'].mean()
DP_avg_d = DP_emo_df['dominance'].mean()

CW_avg_v = CW_emo_df['valence'].mean()
CW_avg_a = CW_emo_df['arousal'].mean()
CW_avg_d = CW_emo_df['dominance'].mean()

CT_avg_v = CT_emo_df['valence'].mean()
CT_avg_a = CT_emo_df['arousal'].mean()
CT_avg_d = CT_emo_df['dominance'].mean()

NN_avg_v = NN_emo_df['valence'].mean()
NN_avg_a = NN_emo_df['arousal'].mean()
NN_avg_d = NN_emo_df['dominance'].mean()

WN_avg_v = WN_emo_df['valence'].mean()
WN_avg_a = WN_emo_df['arousal'].mean()
WN_avg_d = WN_emo_df['dominance'].mean()
```

```
JB_avg_v = JB_emo_df['valence'].mean()
JB_avg_a = JB_emo_df['arousal'].mean()
JB_avg_d = JB_emo_df['dominance'].mean()

MK_avg_v = MK_emo_df['valence'].mean()
MK_avg_a = MK_emo_df['arousal'].mean()
MK_avg_d = MK_emo_df['dominance'].mean()

DR_avg_v = DR_emo_df['valence'].mean()
DR_avg_a = DR_emo_df['arousal'].mean()
DR_avg_d = DR_emo_df['dominance'].mean()

BGT_avg_v = BGT_emo_df['valence'].mean()
BGT_avg_a = BGT_emo_df['arousal'].mean()
BGT_avg_d = BGT_emo_df['dominance'].mean()

DH_avg_v = DH_emo_df['valence'].mean()
DH_avg_a = DH_emo_df['arousal'].mean()
DH_avg_d = DH_emo_df['dominance'].mean()

TT_avg_v = TT_emo_df['valence'].mean()
TT_avg_a = TT_emo_df['arousal'].mean()
TT_avg_d = TT_emo_df['dominance'].mean()

NC_avg_v = NC_emo_df['valence'].mean()
NC_avg_a = NC_emo_df['arousal'].mean()
NC_avg_d = NC_emo_df['dominance'].mean()

CP_avg_v = CP_emo_df['valence'].mean()
CP_avg_a = CP_emo_df['arousal'].mean()
CP_avg_d = CP_emo_df['dominance'].mean()

TP_avg_v = TP_emo_df['valence'].mean()
TP_avg_a = TP_emo_df['arousal'].mean()
TP_avg_d = TP_emo_df['dominance'].mean()

EE_avg_v = EE_emo_df['valence'].mean()
EE_avg_a = EE_emo_df['arousal'].mean()
EE_avg_d = EE_emo_df['dominance'].mean()
```

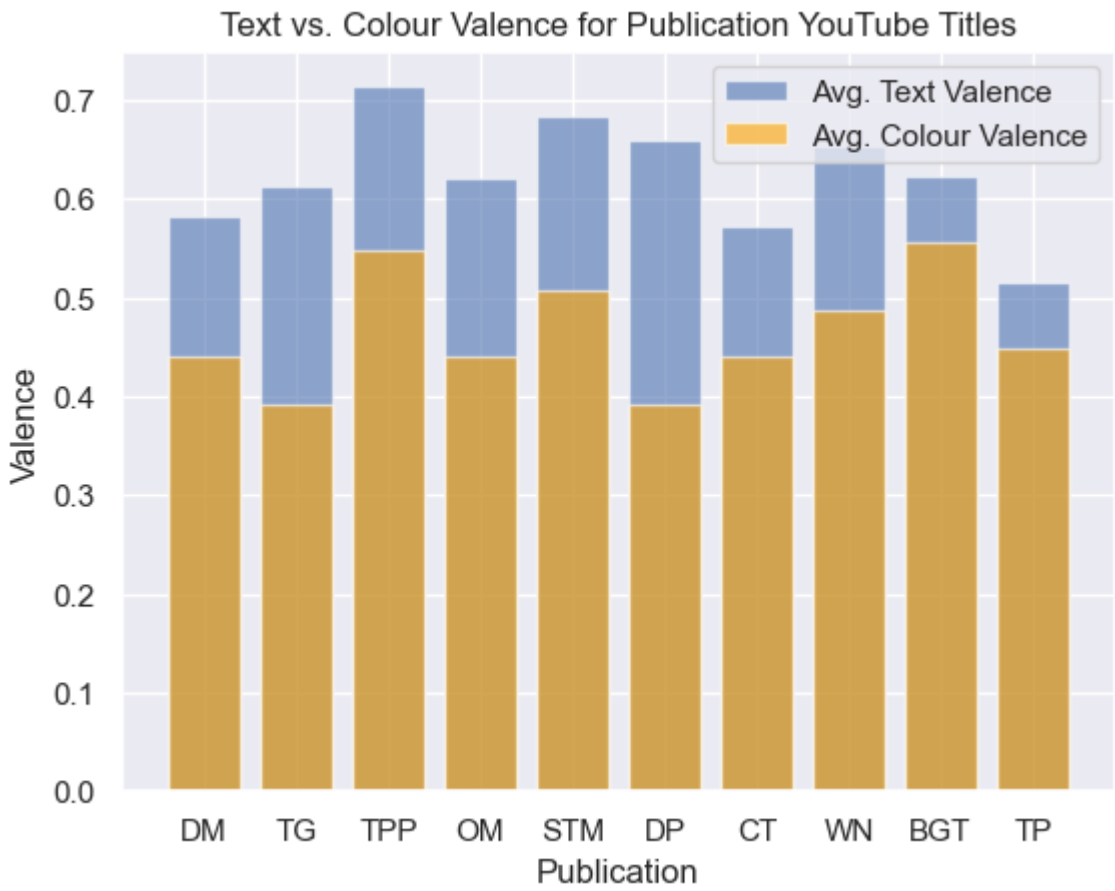
In [55]:

```
#Comparing means of VAD scores
Analysis_data = {'Publication': ['DM','TG','TPP','OM','STM', 'DP', 'CT',  'WN', 'BGT', 'TP'],
        'Avg. Text Valence': [DM_avg_v, TG_avg_v, TPP_avg_v, OM_avg_v, STM_avg_v, DP_avg_v, CT_avg_v, WN_avg_v, BGT_av
```

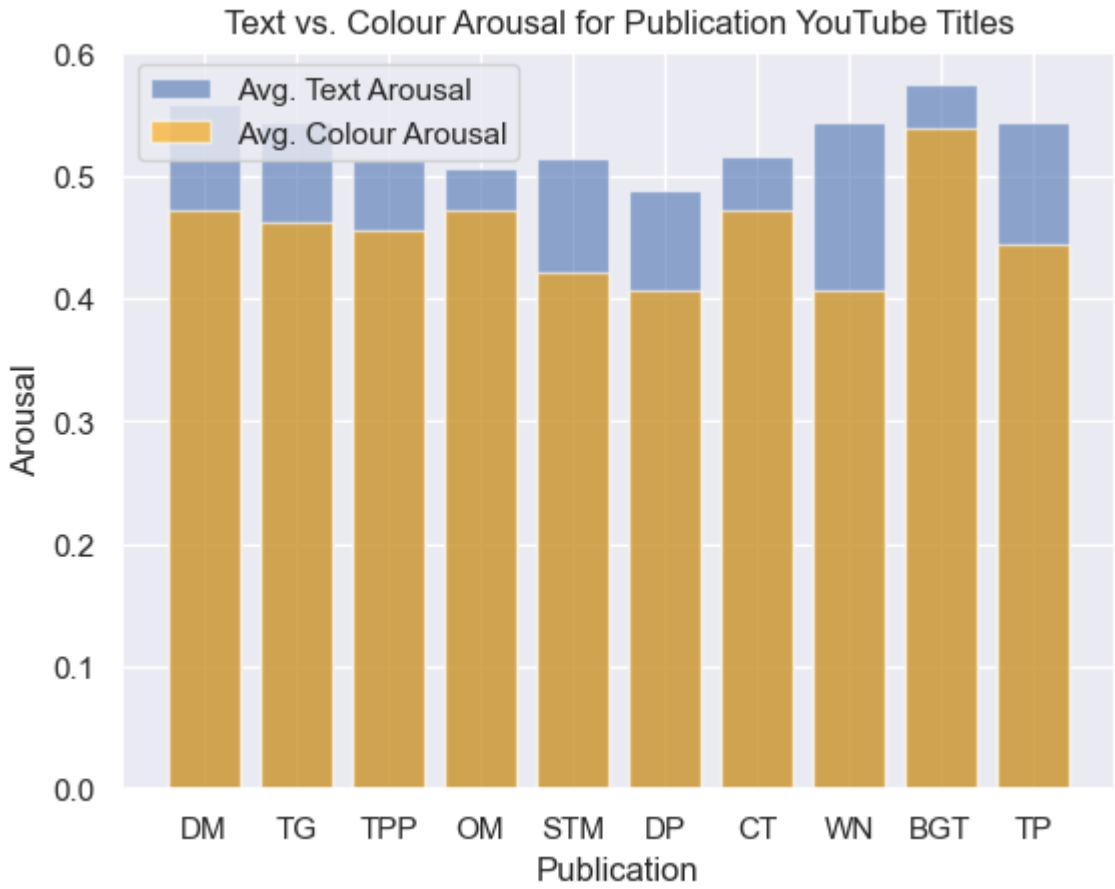
```
        'Avg. Colour Valence': [0.44, 0.392, 0.548, 0.44, 0.508, 0.392, 0.44, 0.488, 0.556, 0.448],
        'Avg. Text Arousal': [DM_avg_a, TG_avg_a, TPP_avg_a, OM_avg_a, STM_avg_a, DP_avg_a, CT_avg_a, WN_avg_a, BGT_av
        'Avg. Colour Arousal': [0.472, 0.462, 0.456, 0.472, 0.422, 0.406, 0.472, 0.406, 0.538, 0.444],
        'Avg. Text Dominance': [DM_avg_d, TG_avg_d, TPP_avg_d, OM_avg_d, STM_avg_d,  DP_avg_d,  CT_avg_d,  WN_avg_d, B
        'Avg. Colour Dominance': [0.4734, 0.54, 0.588, 0.4734, 0.566, 0.54, 0.4734, 0.566, 0.582, 0.534]}
comp_df = pd.DataFrame(Analysis_data)
```

In [56]:

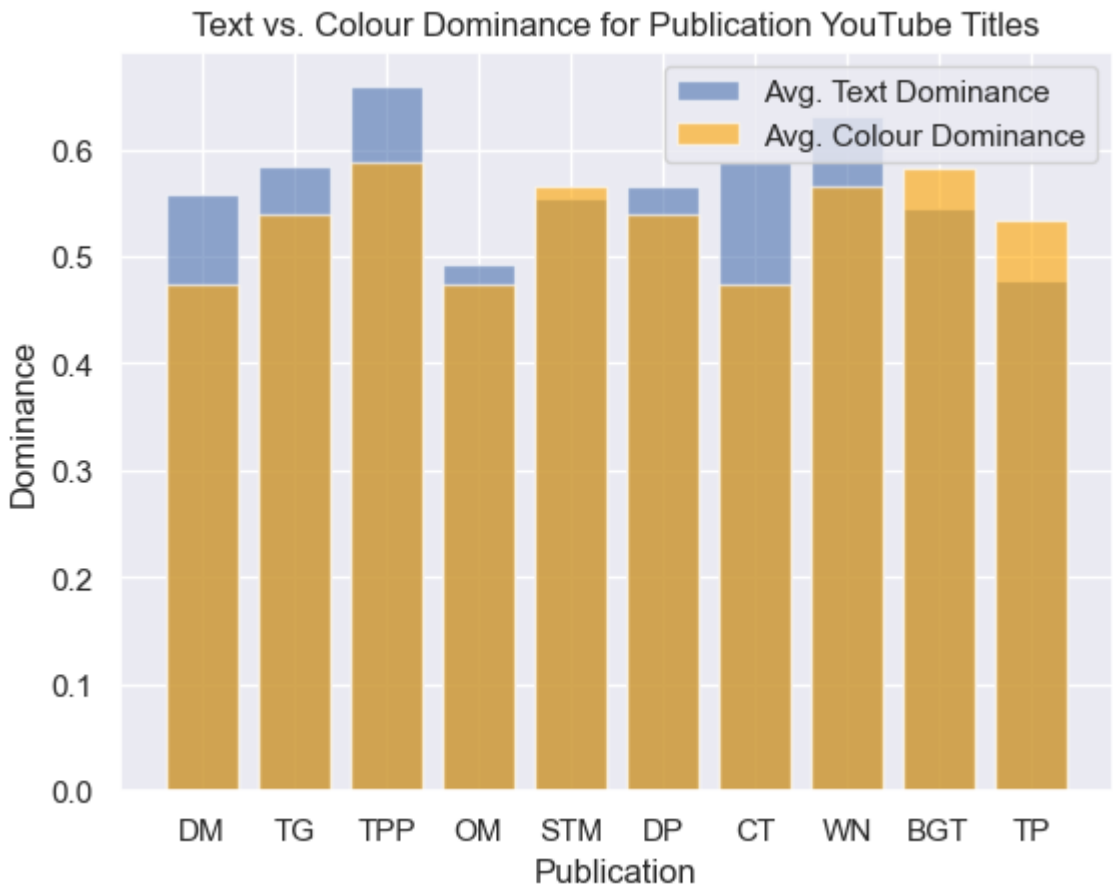
```
labels1 = comp_df['Publication']
data1 = comp_df['Avg. Text Valence']
labels2 = comp_df['Publication']
data2 = comp_df['Avg. Colour Valence']
fig, ax = plt.subplots()
ax.bar(labels1, data1, label='Avg. Text Valence', alpha=0.6)
ax.bar(labels2, data2, label='Avg. Colour Valence', color='orange', alpha=0.6)
ax.set_xlabel('Publication')
ax.set_ylabel('Valence')
ax.set_title('Text vs. Colour Valence for Publication YouTube Titles')
ax.legend()

plt.show()
```



In [57]:
```
labels1 = comp_df['Publication']
data1 = comp_df['Avg. Text Arousal']
labels2 = comp_df['Publication']
data2 = comp_df['Avg. Colour Arousal']
fig, ax = plt.subplots()
ax.bar(labels1, data1, label='Avg. Text Arousal', alpha=0.6)
ax.bar(labels2, data2, label='Avg. Colour Arousal', color='orange', alpha=0.6)
ax.set_xlabel('Publication')
ax.set_ylabel('Arousal')
ax.set_title('Text vs. Colour Arousal for Publication YouTube Titles')
ax.legend()

plt.show()
```



In [58]:
```
data1 = comp_df['Avg. Text Dominance']  # Update to 'Avg. Text Dominance'

labels2 = comp_df['Publication']
data2 = comp_df['Avg. Colour Dominance']  # Update to 'Avg. Colour Dominance'

fig, ax = plt.subplots()
ax.bar(labels1, data1, label='Avg. Text Dominance', alpha=0.6)  # Update label to 'Avg. Text Dominance'
ax.bar(labels2, data2, label='Avg. Colour Dominance', color='orange', alpha=0.6)  # Update label to 'Avg. Colour Domin
ax.set_xlabel('Publication')
ax.set_ylabel('Dominance')  # Update y-axis label to 'Dominance'
ax.set_title('Text vs. Colour Dominance for Publication YouTube Titles')  # Update title
ax.legend()

plt.show()
```



In [13]:
```
#Creating mega dataframe logging titles, political compass, and publisher.
tdf = pd.DataFrame()
tdf['Titles'] = DM_title_list
tdf.loc[:, "Publisher"] = "The Daily Mail"

ttdf = pd.DataFrame()
ttdf['Titles'] = DE_title_list
ttdf.loc[:, "Publisher"] = "The Daily Express"

Tdf = pd.concat([tdf, ttdf], axis=0)
Tdf.loc[:, "Political Compass/Genre"] = "Right-Wing"

tttdf = pd.DataFrame()
tttdf['Titles'] = BBC_title_list
```

```
tttdf.loc[:, "Publisher"] = "The BBC"

ttttdf = pd.DataFrame()
ttttdf['Titles'] = FT_title_list
ttttdf.loc[:, "Publisher"] = "The Financial Times"

TTdf = pd.concat([tttdf, ttttdf], axis=0)
TTdf.loc[:, "Political Compass/Genre"] = "Centrist"

tttttdf = pd.DataFrame()
tttttdf['Titles'] = TG_title_list
tttttdf.loc[:, "Publisher"] = "The Guardian"

ttttttdf = pd.DataFrame()
ttttttdf['Titles'] = TM_title_list
ttttttdf.loc[:, "Publisher"] = "The Mirror"

TTTdf = pd.concat([tttttdf, ttttttdf], axis=0)
TTTdf.loc[:, "Political Compass/Genre"] = "Left-Wing"

adf = pd.DataFrame()
adf['Titles'] = TPP_title_list
adf.loc[:, "Publisher"] = "The Pink Panther"

aadf = pd.DataFrame()
aadf['Titles'] = PP_title_list
aadf.loc[:, "Publisher"] = "Peppa The Pig"

Adf = pd.concat([adf, aadf], axis=0)
Adf.loc[:, "Political Compass/Genre"] = "Film & Animation"

bdf = pd.DataFrame()
bdf['Titles'] = JLY_title_list
bdf.loc[:, "Publisher"] = "Jelly"

bbdf = pd.DataFrame()
bbdf['Titles'] = TDM_title_list
bbdf.loc[:, "Publisher"] = "DanTDM"

Bdf = pd.concat([bdf, bbdf], axis=0)
Bdf.loc[:, "Political Compass/Genre"] = "Gaming"

cdf = pd.DataFrame()
cdf['Titles'] = DH_title_list
```

```
cdf.loc[:, "Publisher"] = "DaveHax"

ccdf = pd.DataFrame()
ccdf['Titles'] = TT_title_list
ccdf.loc[:, "Publisher"] = "Tool Tips"

Cdf = pd.concat([cdf, ccdf], axis=0)
Cdf.loc[:, "Political Compass/Genre"] = "Lifestyle/How-To"


ddf = pd.DataFrame()
ddf['Titles'] = NC_title_list
ddf.loc[:, "Publisher"] = "No Copyright Sounds"

dddf = pd.DataFrame()
dddf['Titles'] = CP_title_list
dddf.loc[:, "Publisher"] = "Coldplay"

Ddf = pd.concat([ddf, dddf], axis=0)
Ddf.loc[:, "Political Compass/Genre"] = "Music"


edf = pd.DataFrame()
edf['Titles'] = TP_title_list
edf.loc[:, "Publisher"] = "The Two Preachers"

eedf = pd.DataFrame()
eedf['Titles'] = EE_title_list
eedf.loc[:, "Publisher"] = "Earthling Ed"

Edf = pd.concat([edf, eedf], axis=0)
Edf.loc[:, "Political Compass/Genre"] = "Non-Profit"


fdf = pd.DataFrame()
fdf['Titles'] = JL_title_list
fdf.loc[:, "Publisher"] = "Jeremy Lynch"

ffdf = pd.DataFrame()
ffdf['Titles'] = SM_title_list
ffdf.loc[:, "Publisher"] = "The Sidemen"

Fdf = pd.concat([fdf, ffdf], axis=0)
Fdf.loc[:, "Political Compass/Genre"] = "People & Blogs"
```

```
gdf = pd.DataFrame()
gdf['Titles'] = KG_title_list
gdf.loc[:, "Publisher"] = "K\'eyush The Stunt Dog"

ggdf = pd.DataFrame()
ggdf['Titles'] = OM_title_list
ggdf.loc[:, "Publisher"] = "One Minute Animals"

Gdf = pd.concat([gdf, ggdf], axis=0)
Gdf.loc[:, "Political Compass/Genre"] = "Animals/Nature"

hdf = pd.DataFrame()
hdf['Titles'] = WB_title_list
hdf.loc[:, "Publisher"] = "Mr. Whose The Boss"

hhdf = pd.DataFrame()
hhdf['Titles'] = STM_title_list
hhdf.loc[:, "Publisher"] = "Steve Mould"

Hdf = pd.concat([hdf, hhdf], axis=0)
Hdf.loc[:, "Political Compass/Genre"] = "Science/Technology"

idf = pd.DataFrame()
idf['Titles'] = FF_title_list
idf.loc[:, "Publisher"] = "F2Freestyle"

iidf = pd.DataFrame()
iidf['Titles'] = ICC_title_list
iidf.loc[:, "Publisher"] = "ICC"

Idf = pd.concat([idf, iidf], axis=0)
Idf.loc[:, "Political Compass/Genre"] = "Sport"

jdf = pd.DataFrame()
jdf['Titles'] = BB_title_list
jdf.loc[:, "Publisher"] = "Bald & Bankrupt"

jjdf = pd.DataFrame()
jjdf['Titles'] = DP_title_list
jjdf.loc[:, "Publisher"] = "Dale Philip"

Jdf = pd.concat([jdf, jjdf], axis=0)
Jdf.loc[:, "Political Compass/Genre"] = "Travel"
```

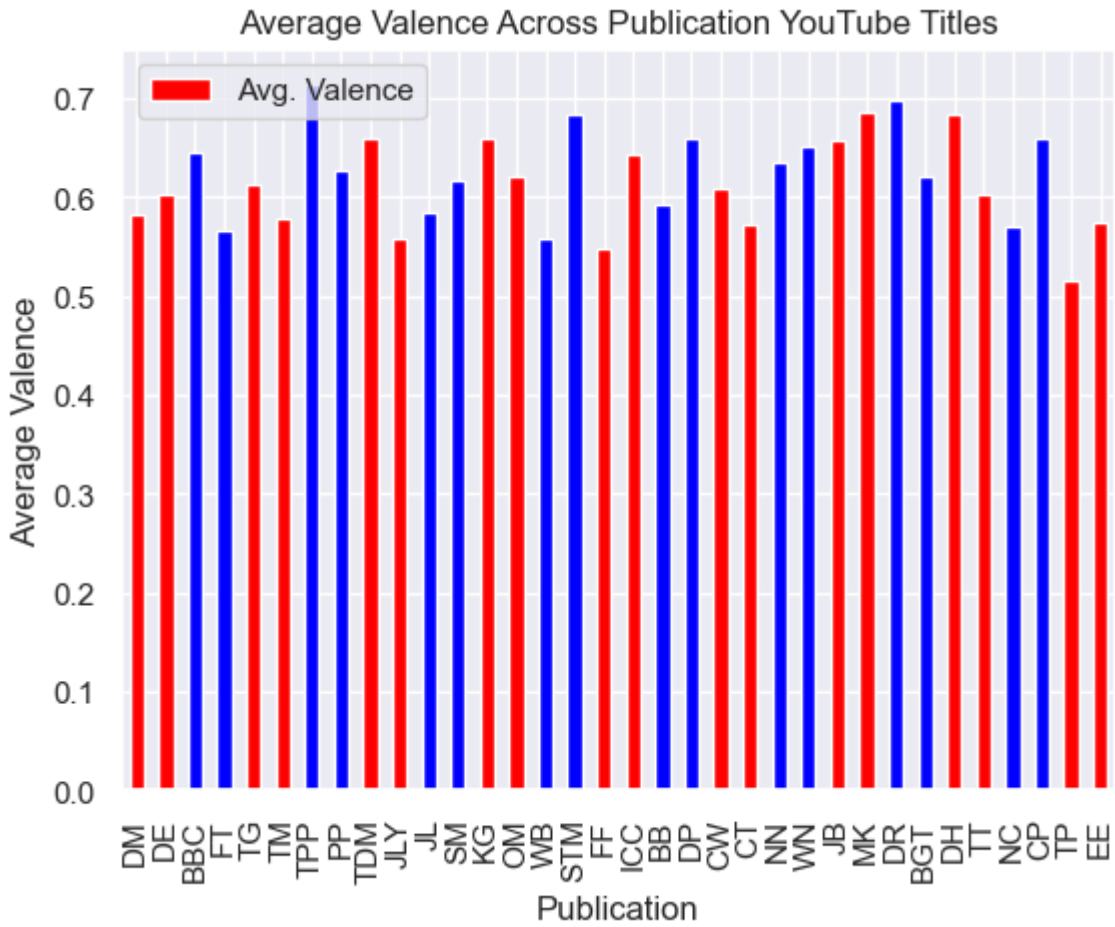
```
hdf = pd.DataFrame()
hdf['Titles'] = CW_title_list
hdf.loc[:, "Publisher"] = "Carwow"

hhdf = pd.DataFrame()
hhdf['Titles'] = CT_title_list
hhdf.loc[:, "Publisher"] = "Car Throttle"

Hdf = pd.concat([hdf, hhdf], axis=0)
Hdf.loc[:, "Political Compass/Genre"] = "Automotive"

idf = pd.DataFrame()
idf['Titles'] = DR_title_list
idf.loc[:, "Publisher"] = "Dan Rhoades"

iidf = pd.DataFrame()
iidf['Titles'] = BGT_title_list
iidf.loc[:, "Publisher"] = "Britain's Got Talent"

Idf = pd.concat([idf, iidf], axis=0)
Idf.loc[:, "Political Compass/Genre"] = "Entertainment"

kdf = pd.DataFrame()
kdf['Titles'] = NN_title_list
kdf.loc[:, "Publisher"] = "Mr. Nigel Ng"

kkdf = pd.DataFrame()
kkdf['Titles'] = WN_title_list
kkdf.loc[:, "Publisher"] = "WillNE"

Kdf = pd.concat([kdf, kkdf], axis=0)
Kdf.loc[:, "Political Compass/Genre"] = "Comedy"

ldf = pd.DataFrame()
ldf['Titles'] = JB_title_list
ldf.loc[:, "Publisher"] = "Jungle Beat"

lldf = pd.DataFrame()
lldf['Titles'] = MK_title_list
lldf.loc[:, "Publisher"] = "Moonbug Kid"

Ldf = pd.concat([ldf, lldf], axis=0)
Ldf.loc[:, "Political Compass/Genre"] = "Education"
```

```
#Finally bringing them together
Mega_Df = pd.concat([Tdf, TTdf, TTTdf, Adf, Bdf, Cdf, Ddf, Edf, Fdf, Gdf, Hdf, Idf, Jdf, Kdf, Ldf], axis=0)
Combined_Titles_df = Mega_Df.reset_index(drop=True)
```

In [14]:
```
#Handy combo list
Combined_Title_List = []
Combined_Title_List.extend(DM_title_list)
Combined_Title_List.extend(DE_title_list)
Combined_Title_List.extend(BBC_title_list)
Combined_Title_List.extend(FT_title_list)
Combined_Title_List.extend(TG_title_list)
Combined_Title_List.extend(TM_title_list)
Combined_Title_List.extend(TPP_title_list)
Combined_Title_List.extend(PP_title_list)
Combined_Title_List.extend(TDM_title_list)
Combined_Title_List.extend(JLY_title_list)
Combined_Title_List.extend(JL_title_list)
Combined_Title_List.extend(SM_title_list)
Combined_Title_List.extend(KG_title_list)
Combined_Title_List.extend(OM_title_list)
Combined_Title_List.extend(WB_title_list)
Combined_Title_List.extend(STM_title_list)
Combined_Title_List.extend(FF_title_list)
Combined_Title_List.extend(ICC_title_list)
Combined_Title_List.extend(BB_title_list)
Combined_Title_List.extend(DP_title_list)
Combined_Title_List.extend(CW_title_list)
Combined_Title_List.extend(CT_title_list)
Combined_Title_List.extend(NN_title_list)
Combined_Title_List.extend(WN_title_list)
Combined_Title_List.extend(JB_title_list)
Combined_Title_List.extend(MK_title_list)
Combined_Title_List.extend(DR_title_list)
Combined_Title_List.extend(BGT_title_list)
Combined_Title_List.extend(DH_title_list)
Combined_Title_List.extend(TT_title_list)
Combined_Title_List.extend(NC_title_list)
Combined_Title_List.extend(CP_title_list)
Combined_Title_List.extend(TP_title_list)
Combined_Title_List.extend(EE_title_list)
```

In [15]:
```
#Plotting average VAD values against publications.
colours = ['red', 'red','blue', 'blue' ]
comp_df.plot(x='Publication', y='Avg. Valence', kind='bar', color = colours)
plt.title('Average Valence Across Publication YouTube Titles')
plt.xlabel('Publication')
plt.ylabel('Average Valence')
```

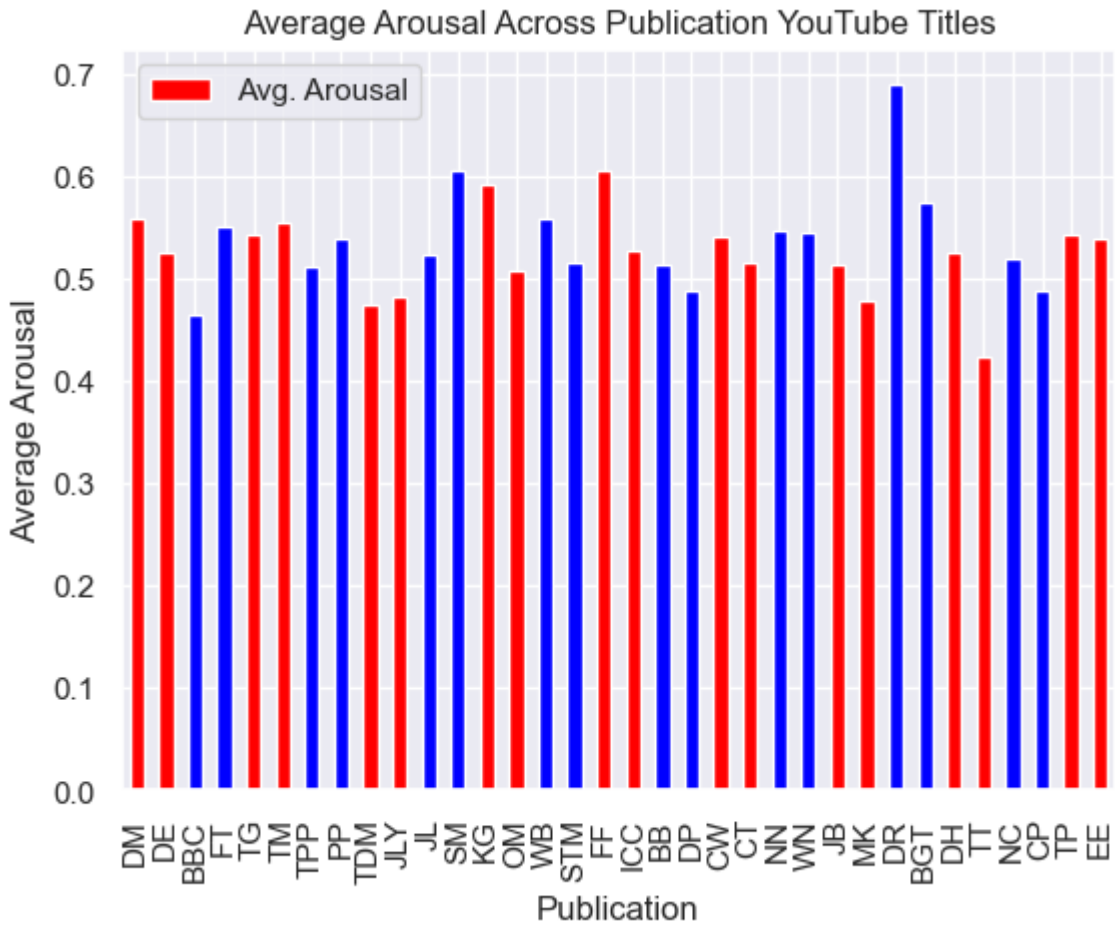
Out[15]:
```
Text(0, 0.5, 'Average Valence')
```



In [16]:
```
News_v_Average = (DM_avg_v + DE_avg_v + BBC_avg_v + FT_avg_v + TG_avg_v + TM_avg_v)/6
Others_v_Average = (TPP_avg_v + PP_avg_v + TDM_avg_v + JLY_avg_v + JL_avg_v + SM_avg_v + KG_avg_v + OM_avg_v + WB_avg_
```

In [17]:
```
#Plotting average VAD values against publications.
colours = ['red', 'red','blue', 'blue' ]
comp_df.plot(x='Publication', y='Avg. Arousal', kind='bar', color = colours)
plt.title('Average Arousal Across Publication YouTube Titles')
plt.xlabel('Publication')
plt.ylabel('Average Arousal')
```

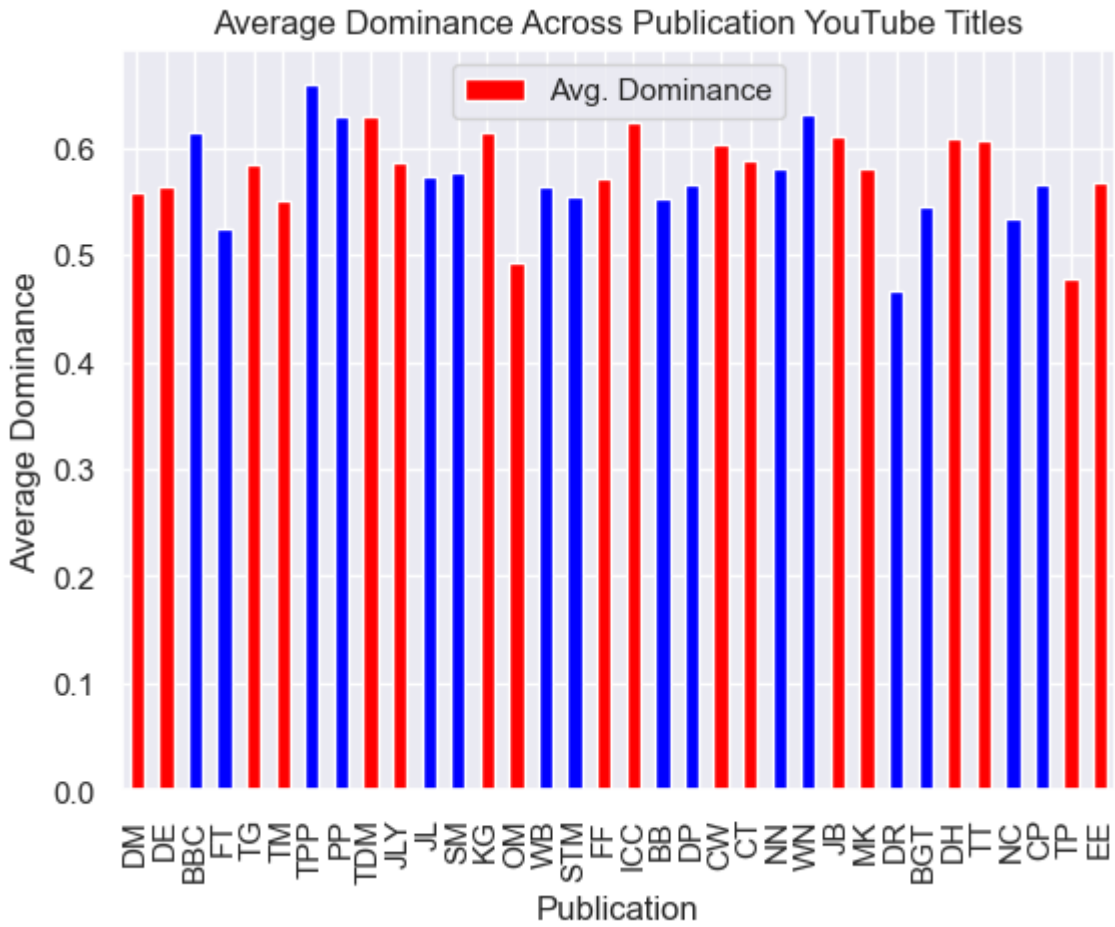
Out[17]:
```
Text(0, 0.5, 'Average Arousal')
```



In [18]:
```
News_a_Average = (DM_avg_a + DE_avg_a + BBC_avg_a + FT_avg_a + TG_avg_a + TM_avg_a)/6
Others_a_Average = (TPP_avg_a + PP_avg_a + TDM_avg_a + JLY_avg_a + JL_avg_a + SM_avg_a + KG_avg_a + OM_avg_a + WB_avg_
```

In [19]:
```
#Plotting average VAD values against publications.
colours = ['red', 'red','blue', 'blue' ]
comp_df.plot(x='Publication', y='Avg. Dominance', kind='bar', color = colours)
plt.title('Average Dominance Across Publication YouTube Titles')
plt.xlabel('Publication')
plt.ylabel('Average Dominance')
```

Out[19]:
```
Text(0, 0.5, 'Average Dominance')
```

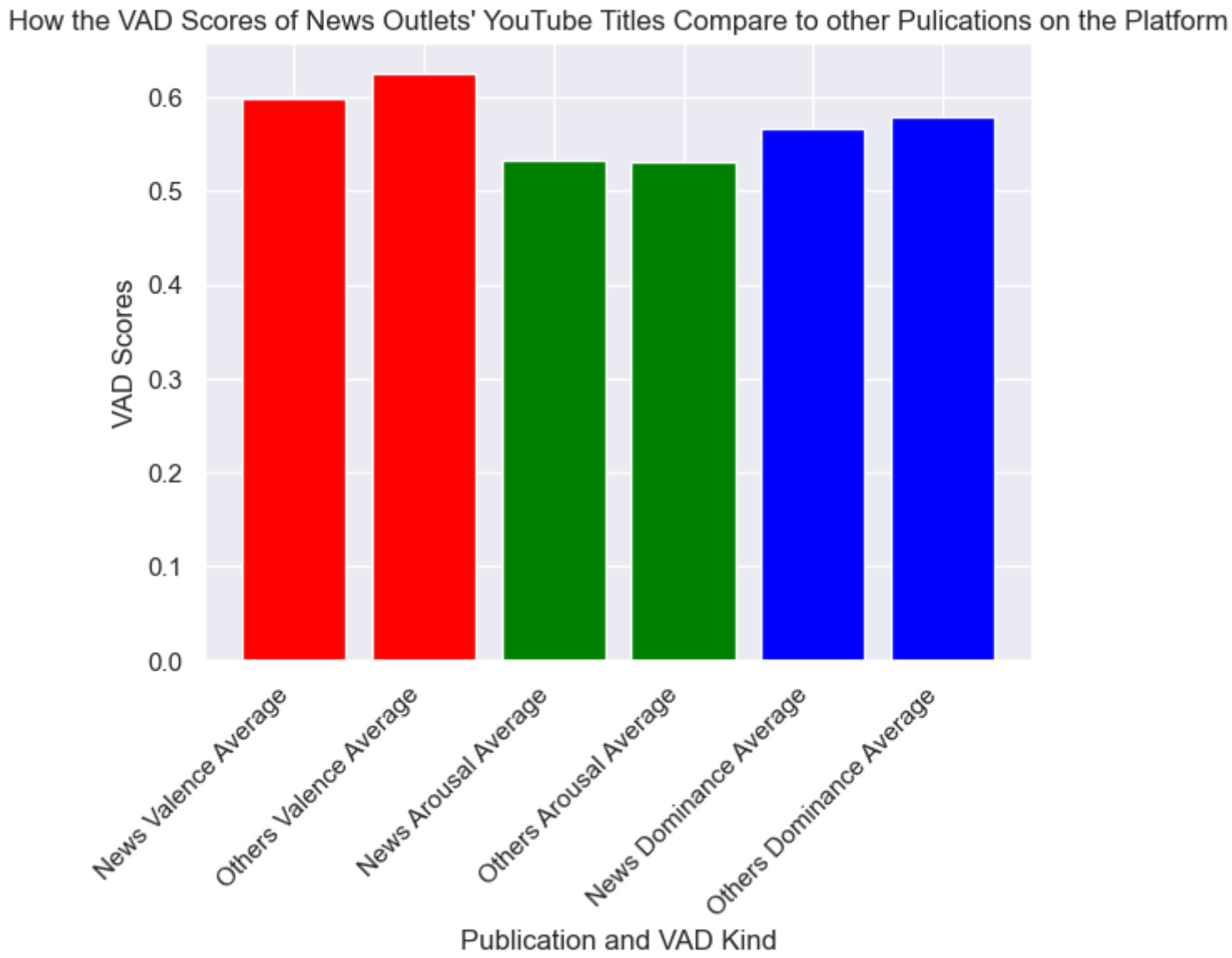


In [20]:
```
News_d_Average = (DM_avg_d + DE_avg_d + BBC_avg_d + FT_avg_d + TG_avg_d + TM_avg_d)/6
Others_d_Average = (TPP_avg_d + PP_avg_d + TDM_avg_d + JLY_avg_d + JL_avg_d + SM_avg_d + KG_avg_d + OM_avg_d + WB_avg_
```

In [21]:
```
News_v_Average,Others_v_Average,News_a_Average,Others_a_Average,News_d_Average,Others_d_Average
```

Out[21]:
```
(0.5984355067194002,
 0.6252473915077966,
 0.5325049121060835,
 0.5312229836013843,
 0.5659514522982412,
 0.5790106274175102)
```

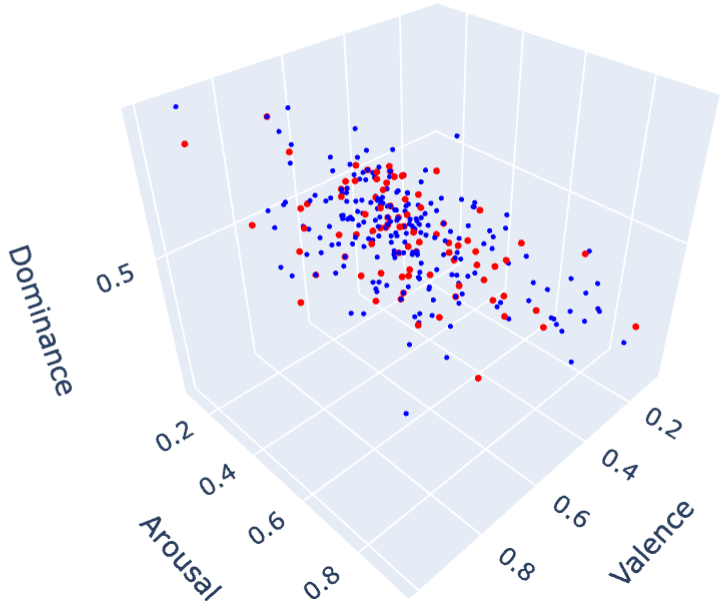
In [22]:
```
labels = ['News Valence Average', 'Others Valence Average', 'News Arousal Average', 'Others Arousal Average', 'News Do
values = [News_v_Average, Others_v_Average, News_a_Average, Others_a_Average, News_d_Average, Others_d_Average]
colours = ['red', 'red', 'green', 'green', 'blue', 'blue']
plt.bar(labels, values, color=colours)
plt.xlabel('Publication and VAD Kind')
plt.ylabel('VAD Scores')
plt.title('How the VAD Scores of News Outlets\' YouTube Titles Compare to other Pulications on the Platform')
plt.xticks(rotation=45, ha='right')
plt.show()
```



In [23]:
```
x1 = DM_emo_df['valence']
y1 = DM_emo_df['arousal']
z1 = DM_emo_df['dominance']

x2 = BBC_emo_df['valence']
y2 = BBC_emo_df['arousal']
z2 = BBC_emo_df['dominance']
```

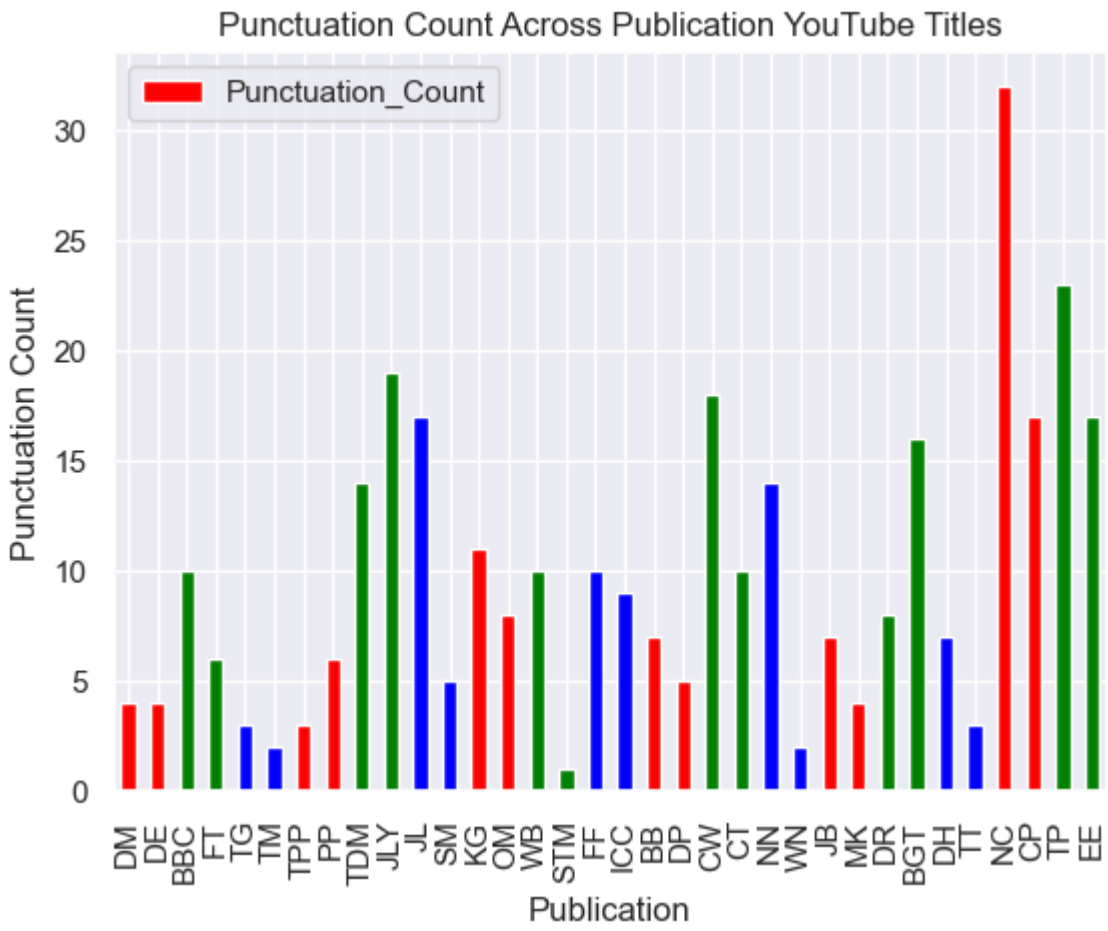
```
x3 = TG_emo_df['valence']
y3 = TG_emo_df['arousal']
z3 = TG_emo_df['dominance']

x4 = TPP_emo_df['valence']
y4 = TPP_emo_df['arousal']
z4 = TPP_emo_df['dominance']

x5 = TDM_emo_df['valence']
y5 = TDM_emo_df['arousal']
z5 = TDM_emo_df['dominance']

x6 = JL_emo_df['valence']
y6 = JL_emo_df['arousal']
z6 = JL_emo_df['dominance']

x7 = KG_emo_df['valence']
y7 = KG_emo_df['arousal']
z7 = KG_emo_df['dominance']

x8 = WB_emo_df['valence']
y8 = WB_emo_df['arousal']
z8 = WB_emo_df['dominance']

x9 = FF_emo_df['valence']
y9 = FF_emo_df['arousal']
z9 = FF_emo_df['dominance']

x10 = BB_emo_df['valence']
y10 = BB_emo_df['arousal']
z10 = BB_emo_df['dominance']

x11 = CW_emo_df['valence']
y11 = CW_emo_df['arousal']
z11 = CW_emo_df['dominance']

x12 = NN_emo_df['valence']
y12 = NN_emo_df['arousal']
z12 = NN_emo_df['dominance']

x13 = JB_emo_df['valence']
y13 = JB_emo_df['arousal']
z13 = JB_emo_df['dominance']
```

```
x14 = DR_emo_df['valence']
y14 = DR_emo_df['arousal']
z14 = DR_emo_df['dominance']

x15 = DH_emo_df['valence']
y15 = DH_emo_df['arousal']
z15 = DH_emo_df['dominance']

x16 = NC_emo_df['valence']
y16 = NC_emo_df['arousal']
z16 = NC_emo_df['dominance']

x17 = TP_emo_df['valence']
y17 = TP_emo_df['arousal']
z17 = TP_emo_df['dominance']

trace1 = go.Scatter3d(x=x1,y=y1,z=z1, mode='markers', marker=dict(size=2, color='red'), name='The Daily Mail')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2,mode='markers',marker=dict(size=2,color='red'),name='The BBC')
trace3 = go.Scatter3d(x=x3,y=y3,z=z3,mode='markers',marker=dict(size=2,color='red'),name='The Guardian')
trace4 = go.Scatter3d(x=x4,y=y4,z=z4,mode='markers',marker=dict(size=1.5,color='blue'),name='The Pink Panther')
trace5 = go.Scatter3d(x=x5,y=y5,z=z5,mode='markers',marker=dict(size=1.5,color='blue'),name='The DanTDM')
trace6 = go.Scatter3d(x=x6,y=y6,z=z6,mode='markers',marker=dict(size=1.5,color='blue'),name='The James Lynch')
trace7 = go.Scatter3d(x=x7,y=y7,z=z7,mode='markers',marker=dict(size=1.5,color='blue'),name='K\'eyush The Stunt Dog')
trace8 = go.Scatter3d(x=x8,y=y8,z=z8,mode='markers',marker=dict(size=1.5,color='blue'),name='Mr. Who\'s Boss')
trace9 = go.Scatter3d(x=x9,y=y9,z=z9,mode='markers',marker=dict(size=1.5,color='blue'),name='F2Freestyle')
trace10 = go.Scatter3d(x=x10,y=y10,z=z10,mode='markers',marker=dict(size=1.5,color='blue'),name='Bald & Bankrupt')
trace11 = go.Scatter3d(x=x11,y=y11,z=z11,mode='markers',marker=dict(size=1.5,color='blue'),name='Carwow')
trace12 = go.Scatter3d(x=x12,y=y12,z=z12,mode='markers',marker=dict(size=1.5,color='blue'),name='Mr. Nigel Ng')
trace13 = go.Scatter3d(x=x13,y=y13,z=z13,mode='markers',marker=dict(size=1.5,color='blue'),name='Jungle Beat')
trace14 = go.Scatter3d(x=x14,y=y14,z=z14,mode='markers',marker=dict(size=1.5,color='blue'),name='Dan Rhoades')
trace15 = go.Scatter3d(x=x15,y=y15,z=z15,mode='markers',marker=dict(size=1.5,color='blue'),name='DaveHax')
trace16 = go.Scatter3d(x=x16,y=y16,z=z16,mode='markers',marker=dict(size=1.5,color='blue'),name='No Copyright Sounds')
trace17 = go.Scatter3d(x=x17,y=y17,z=z17,mode='markers',marker=dict(size=1.5,color='blue'),name='The Two Preachers')


fig = go.Figure(data=[trace1, trace2, trace3, trace4, trace5, trace6, trace7, trace8, trace9, trace10, trace11, trace1
fig.update_layout(title='Comparing VAD Scores Across Right-Wing, Centrist, and Left-Wing UK Publications.\nAchieved Th
fig.show()
```



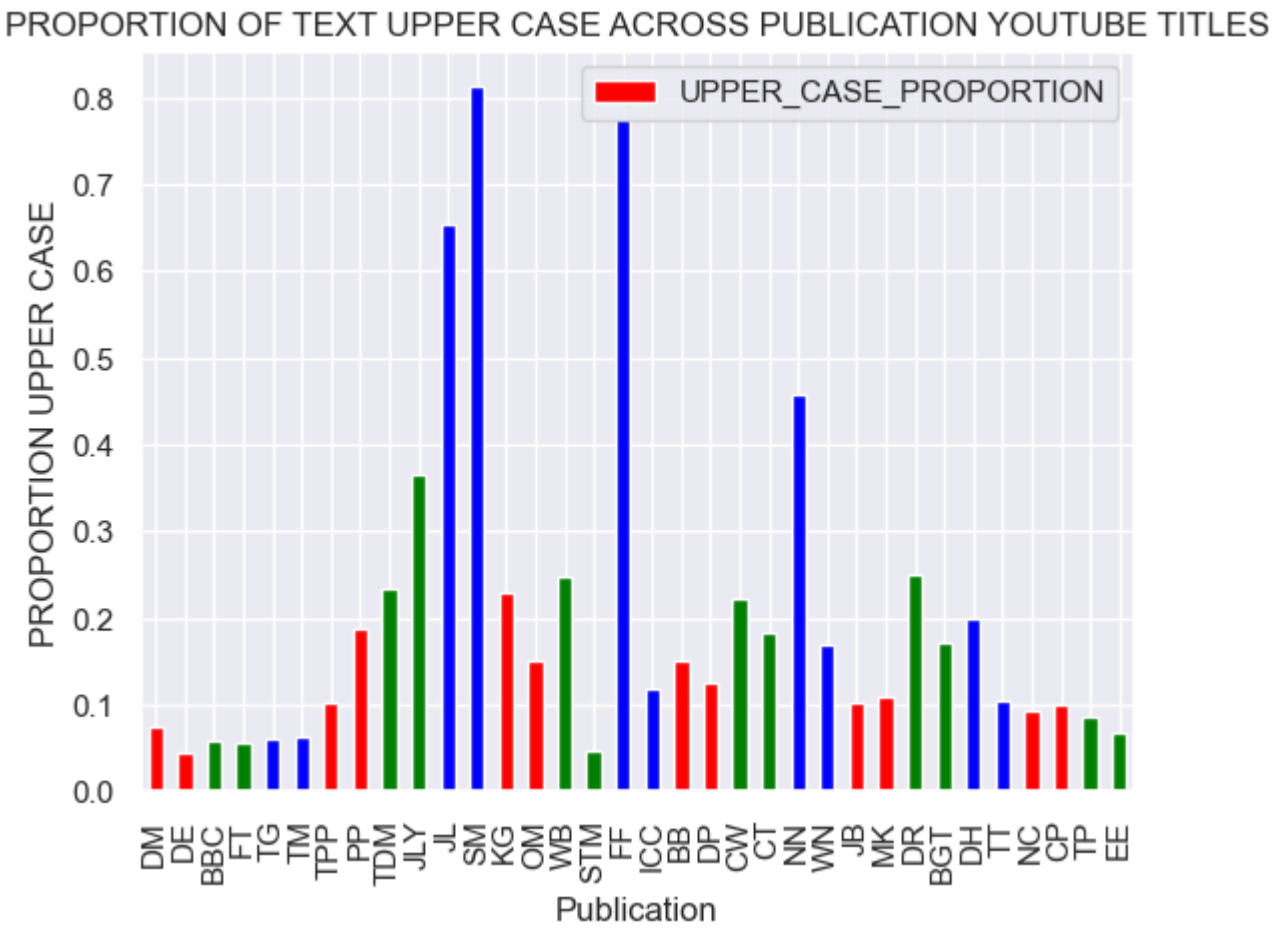
In [24]:
```
Punct_data = {'Publication': ['DM', 'DE', 'BBC', 'FT', 'TG', 'TM', 'TPP','PP','TDM','JLY','JL','SM','KG','OM','WB','ST
Punct_df = pd.DataFrame(Punct_data)
Punct_df.plot(x='Publication', y='Punctuation_Count', kind='bar', color = colours)
plt.title('Punctuation Count Across Publication YouTube Titles')
plt.xlabel('Publication')
plt.ylabel('Punctuation Count')
```

Out[24]:
```
Text(0, 0.5, 'Punctuation Count')
```



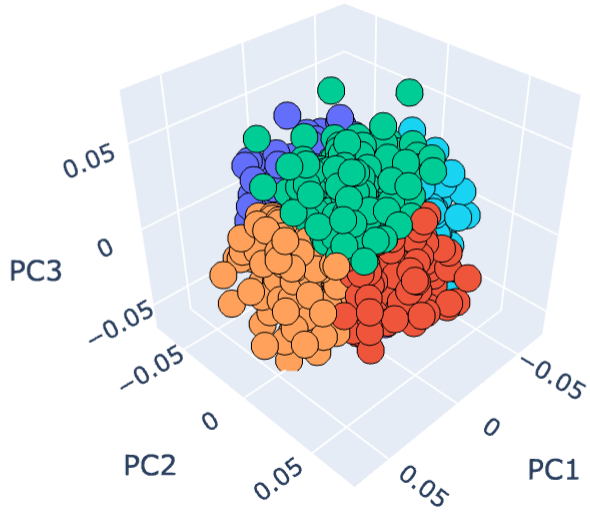
In [25]:
```
UPPER_CASE_data = {'Publication': ['DM', 'DE', 'BBC', 'FT', 'TG', 'TM', 'TPP', 'PP', 'TDM', 'JLY', 'JL', 'SM', 'KG', '
             'UPPER_CASE_PROPORTION': [DM_PROP_UPPER, DE_PROP_UPPER, BBC_PROP_UPPER, FT_PROP_UPPER, TG_PROP_UPPER, TM
UPPER_CASE_df = pd.DataFrame(UPPER_CASE_data)
UPPER_CASE_df.plot(x='Publication', y='UPPER_CASE_PROPORTION', kind='bar', color = colours)
plt.title('PROPORTION OF TEXT UPPER CASE ACROSS PUBLICATION YOUTUBE TITLES')
plt.xlabel('Publication')
plt.ylabel('PROPORTION UPPER CASE')
```

Out[25]:
```
Text(0, 0.5, 'PROPORTION UPPER CASE')
```



In [26]:
```
#Formatting for W2V
CT_w2v = [sentence.split() for sentence in Combined_Title_List]
CT_model = gensim.models.Word2Vec(CT_w2v, min_count=1, vector_size = 30)

#Sorting by most freqeunt words in vocab
CT_vocab = CT_model.wv.index_to_key

#Assiging vocab to its unique matrix values
CT_vectors = [CT_model.wv[i] for i in CT_vocab]

#Dataframe of matrix values
CT_df = pd.DataFrame(CT_vectors)
```

```
#Perform PCA analysis on dataframe
CT_pca = PCA(n_components=3)
CT_PCA_result = CT_pca.fit_transform(CT_df)
CT_PCA_df = pd.DataFrame(CT_PCA_result, columns=['PC1', 'PC2', 'PC3'])

#Callibrating first column back with original words
CT_PCA_complete_df = pd.concat([pd.Series(CT_vocab, name='word'), CT_PCA_df], axis=1)

#Clustering
kmeans = KMeans(n_clusters=6).fit(CT_PCA_complete_df[['PC1', 'PC2', 'PC3']])
CT_PCA_complete_df['cluster'] = [str(i) for i in kmeans.labels_]

#Visualising
WE_fig = px.scatter_3d(CT_PCA_complete_df, x='PC1', y='PC2', z='PC3', color='cluster',  hover_data = ['word'])
WE_fig.update_layout(title='YouTube Titles of Left/Centre/Right Wing UK news outlets – PCA analysis of Word Embedding'
WE_fig.update_traces(marker=dict(size = 8, line=dict(width=1, color='black')), selector=dict(mode='markers'))
WE_fig.show()
```



In [27]:
```
#Setting up Vectorizer
vectorizer = TfidfVectorizer(input = 'content', strip_accents = 'ascii', stop_words = 'english')

#Channeling combined publisher data through vectoriser
CT_V = vectorizer.fit_transform(Combined_Title_List)
CT_V = CT_V.todense().tolist()

#Converting to distance matrix to handy df
```

```
CT_V_Df = pd.DataFrame(CT_V, columns=vectorizer.get_feature_names())
Distances = [[] for i in range(len(CT_V_Df))]
for i in range(len(CT_V_Df)):
    for j in range(len(CT_V_Df)):
        Distances[i].append(distance.cosine(CT_V_Df.iloc[i], CT_V_Df.iloc[j]))
Dist_CT_V_Df = pd.DataFrame(Distances, columns = CT_V_Df.index, index = CT_V_Df.index)
```

```
/Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarning:

Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please us
e get_feature_names_out instead.

/Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages/scipy/spatial/distance.py:620: RuntimeWarning:

invalid value encountered in double_scalars
```

In [28]:
```
pca_1 = PCA(n_components = 3)
comps_1 = pca_1.fit_transform(Dist_CT_V_Df)
pc_df_1 = pd.DataFrame(data = comps_1, columns = ['PC1', 'PC2', 'PC3'])

#Clustering
clustering = AgglomerativeClustering(n_clusters=5, linkage='ward').fit(Dist_CT_V_Df)
kmeans = KMeans(n_clusters=5, random_state=0).fit(Dist_CT_V_Df)

#DF for visualisation
TC_DF_All = pd.concat([Dist_CT_V_Df, pc_df_1], axis = 1)
```

```
/Users/louisvsbigmac/opt/anaconda3/lib/python3.9/site-packages/scipy/cluster/hierarchy.py:834: ClusterWarning:

scipy.cluster: The symmetric non-negative hollow observation matrix looks suspiciously like an uncondensed distance ma
trix
```

In [29]:
```
TC_DF_All['clusters_ag'] = [str(i) for i in clustering.labels_]
TC_DF_All['clusters_knn'] = [str(i) for i in kmeans.labels_]
TC_DF_All['Titles'] = Combined_Titles_df["Titles"]
TC_DF_All['Publisher'] = Combined_Titles_df["Publisher"]
TC_DF_All['Political Compass'] = Combined_Titles_df["Political Compass/Genre"]
```

In [30]:
```
#Visualising
TA_fig = px.scatter_3d(TC_DF_All, x='PC1', y='PC2', z='PC3', color='clusters_ag', hover_data = ['Titles','Political Co
TA_fig.update_traces(marker=dict(size = 8, line=dict(width=1, color='black')), selector=dict(mode='markers'))
```

```
TA_fig.update_layout(title='YouTube Titles of Left/Centre/Right Wing UK news outlets – Topic analysis')
TA_fig.show()
```