**#Python script for performing LDA topic modelling**

from sklearn.model\_selection import GridSearchCV

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.decomposition import LatentDirichletAllocation

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test = train\_test\_split(df["clean\_tokens2"], test\_size=0.2, random\_state = 50)

perplexity\_scores = []

num\_topics\_range = range(2, 18, 2)

for num\_topics in num\_topics\_range:

tfidf\_vec = TfidfVectorizer()

tfidf\_matrix\_train = tfidf\_vec.fit\_transform(X\_train)

lda\_model = LatentDirichletAllocation(n\_components=num\_topics, random\_state = 50)

lda\_matrix\_train = lda\_model.fit\_transform(tfidf\_matrix\_train)

perplexity = lda\_model.perplexity(tfidf\_vec.transform(X\_test))

perplexity\_scores.append(perplexity)

font = {'family': 'serif', 'weight': 'normal', 'size': 16}

plt.rc('font', \*\*font)

plt.plot(num\_topics\_range, perplexity\_scores, marker='o')

plt.xlabel('Number of Topics')

plt.ylabel('Perplexity')

plt.gca().spines['top'].set\_visible(False)

plt.gca().spines['right'].set\_visible(False)

plt.show()

# Create the TF-IDF matrix

tfidf\_vec = TfidfVectorizer()

tfidf\_matrix = tfidf\_vec.fit\_transform(df["clean\_tokens2"])

# Define the parameter grid

param\_grid = {

'batch\_size': [10, 50, 100],

'max\_iter': [100, 100, 500],

'doc\_topic\_prior': ['auto', 0.01, 0.1], # Experiment with different alpha values

}

# Create the LDA model

lda\_model = LatentDirichletAllocation()

# Perform grid search

grid\_search = GridSearchCV(lda\_model, param\_grid, cv=5)

grid\_search.fit(tfidf\_matrix)

# Display the best parameters

print("Best Parameters:", grid\_search.best\_params\_)

lda\_model = LatentDirichletAllocation()

lda\_model.fit(tfidf\_matrix)

lda\_model = LatentDirichletAllocation(batch\_size = 50,

doc\_topic\_prior = 0.1,

max\_iter = 100)

# Apply LDA

num\_topics = 12

lda\_model = LatentDirichletAllocation(n\_components=num\_topics, random\_state=50)

lda\_matrix = lda\_model.fit\_transform(tfidf\_matrix)

# Display the Topics and Associated Words

feature\_names = tfidf\_vec.get\_feature\_names\_out()

# Function to print top words for each topic

def display\_topics(model, feature\_names, n\_top\_words):

for topic\_idx, topic in enumerate(model.components\_):

print(f"Topic #{topic\_idx + 1}:")

print([feature\_names[i] for i in topic.argsort()[:-n\_top\_words - 1:-1]])

print()

# Print the top words for each topic

n\_top\_words = 5

display\_topics(lda\_model, feature\_names, n\_top\_words)

# Assign Topics to tweets

df["Topic"] = lda\_matrix.argmax(axis=1)

# Display the DataFrame with assigned topics

print(df[["Text", "Topic"]])

topic\_counts = df["Topic"].value\_counts().sort\_index()

topic\_counts.index += 1

font = {'family': 'serif', 'weight': 'normal', 'size': 16}

plt.rc('font', \*\*font)

plt.figure(figsize=(10, 6))

sns.barplot(x=topic\_counts.index, y=topic\_counts.values, color="teal")

plt.xlabel("Topic")

plt.ylabel("Number of Tweets")

plt.gca().spines['top'].set\_visible(False)

plt.gca().spines['right'].set\_visible(False)

plt.xticks()

plt.show()

plt.figure(figsize=(15, 8))

for topic\_idx in range(num\_topics):

top\_words = [feature\_names[i] for i in lda\_model.components\_[topic\_idx].argsort()[:-n\_top\_words - 1:-1]]

plt.bar(top\_words, lda\_matrix[:, topic\_idx].sum(axis=0), label=f"Topic {topic\_idx + 1}")

# Set the font family and size

font = {'family': 'serif', 'weight': 'normal', 'size': 14}

plt.rc('font', \*\*font)

plt.xlabel("Top Words from Each Topic")

plt.ylabel("Number of Tweets")

plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left')

plt.gca().spines['top'].set\_visible(False)

plt.gca().spines['right'].set\_visible(False)

plt.xticks(rotation=45, ha='right')

plt.tight\_layout()

plt.show()