Module 5 - Spooky Authorship Identification

Group 13

- Aidan Lonergan
- Daniel Lillard
- Radhika Garg
- Claudine Uwiragiye

Objective

1. Accurately identify the author of the sentences in the test set
2. Perform all work with Apache Spark

Stage 0 - Import Data

1. Create a code notebook called: code_6_of_10_data_mine_group13.ipynb
2. Load the dataset into Spark data objects and explore structure, size, and distribution of information

In [1]:
```
# Stage 0 Solution
from pyspark.sql import SparkSession
import pandas as pd

# Start spark session and load train and test data sets
spark = SparkSession.builder.appName("Module_5_Project").getOrCreate()
df_train = spark.read.csv('./train.csv', header=True, inferSchema=True, quote='"',
```

Summary

In [2]:
```
# Print size and descriptive statistics
print("==== DataSet Shape ====")
print(f"{len(df_train.columns)} columns\n{df_train.count()} rows\n")

print("==== DataSet Descriptive Statistics ====")
print(df_train.describe().show())

print("\n==== DataSet Unique Authors ====")
print(df_train.select('author').distinct().show())
```

```
==== DataSet Shape ====
3 columns
19579 rows

==== DataSet Descriptive Statistics ====
+-------+-------+--------------------+------+
|summary|     id|                text|author|
+-------+-------+--------------------+------+
|  count|  19579|               19579| 19579|
|   mean|   NULL|                NULL|  NULL|
| stddev|   NULL|                NULL|  NULL|
|    min|id00001|" Odenheimer, res...|   EAP|
|    max|id27971|you could not hop...|   MWS|
+-------+-------+--------------------+------+

None

==== DataSet Unique Authors ====
+-------+
|author|
+-------+
|    MWS|
|    HPL|
|    EAP|
+-------+

None
```

Stage 1 - Data Preparation (Exploratory data analysis and text mining pre-processing)

1. Perform exploratory data analysis and create visualizations and tables as needed
2. Text Preprocessing: perform tasks like tokenization and stopwords removal to clean text data
    - Tokenize - split the text into individual words aka tokens.
    - Remove stop.words - frequently used pronouns and personal references.
        - Top ten include: I, you, he, she, it, we, they, me, him, her
    - Lemmatization - convert words to their root (optional).
        - Lemmatization is a text normalization technique that reduces words to their base or dictionary form (lemma). Use to reduce inflected or derived words to their root form for better analysis and modeling outcomes

In [3]:
```python
# Step 1 - Preprocessing
import matplotlib.pyplot as plt
from pyspark.sql.functions import col, split, explode, length, lower, regexp_replac
from collections import Counter
import re
import nltk
from nltk.corpus import stopwords
```

```python
# Get stop words
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

# Clean and lowercase text, remove punctuation
df_train_cleaned = df_train.withColumn("clean_text", lower(regexp_replace(col("text

# Tokenize into words then filter out empty strings after tokenization
df_train_words = df_train_cleaned.withColumn("word", explode(split(col("clean_text"

# Remove stop words
df_train_filtered = df_train_words.filter(~col("word").isin(stop_words))

df_train_filtered.show(10)
```

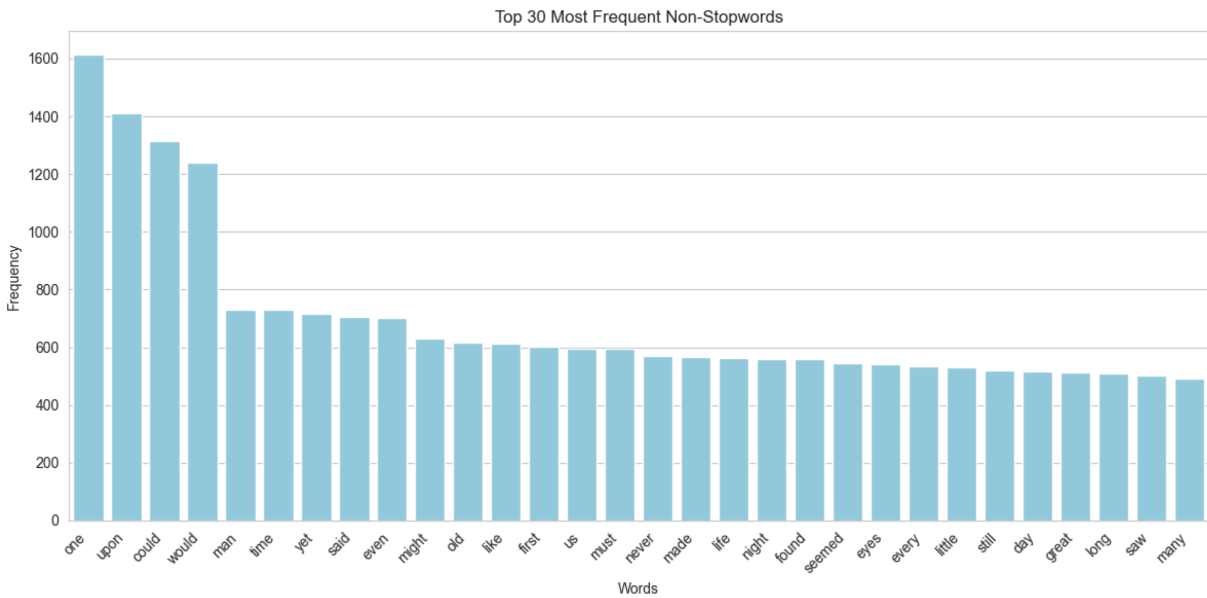
```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\aflon\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
+-------+--------------------+------+--------------------+------------+
|     id|                text|author|          clean_text|        word|
+-------+--------------------+------+--------------------+------------+
|id26305|This process, how...|   EAP|this process howe...|     process|
|id26305|This process, how...|   EAP|this process howe...|     however|
|id26305|This process, how...|   EAP|this process howe...|    afforded|
|id26305|This process, how...|   EAP|this process howe...|       means|
|id26305|This process, how...|   EAP|this process howe...|ascertaining|
|id26305|This process, how...|   EAP|this process howe...|  dimensions|
|id26305|This process, how...|   EAP|this process howe...|     dungeon|
|id26305|This process, how...|   EAP|this process howe...|       might|
|id26305|This process, how...|   EAP|this process howe...|        make|
|id26305|This process, how...|   EAP|this process howe...|     circuit|
+-------+--------------------+------+--------------------+------------+
only showing top 10 rows
```

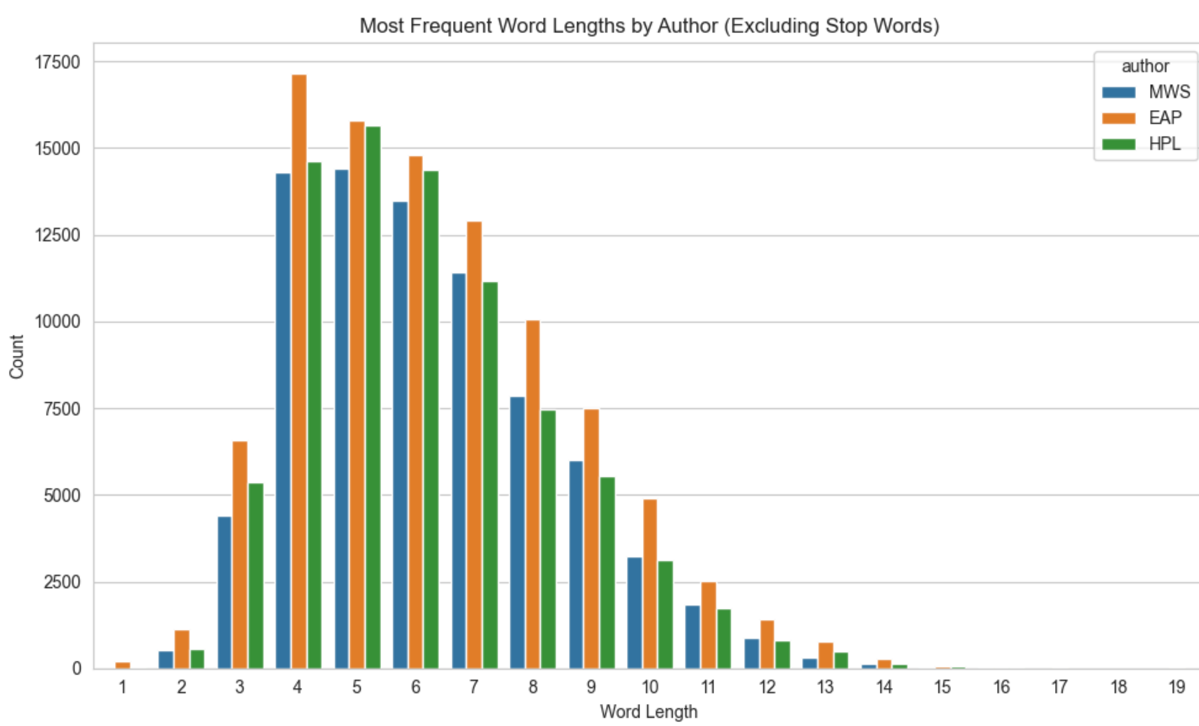
In [4]:

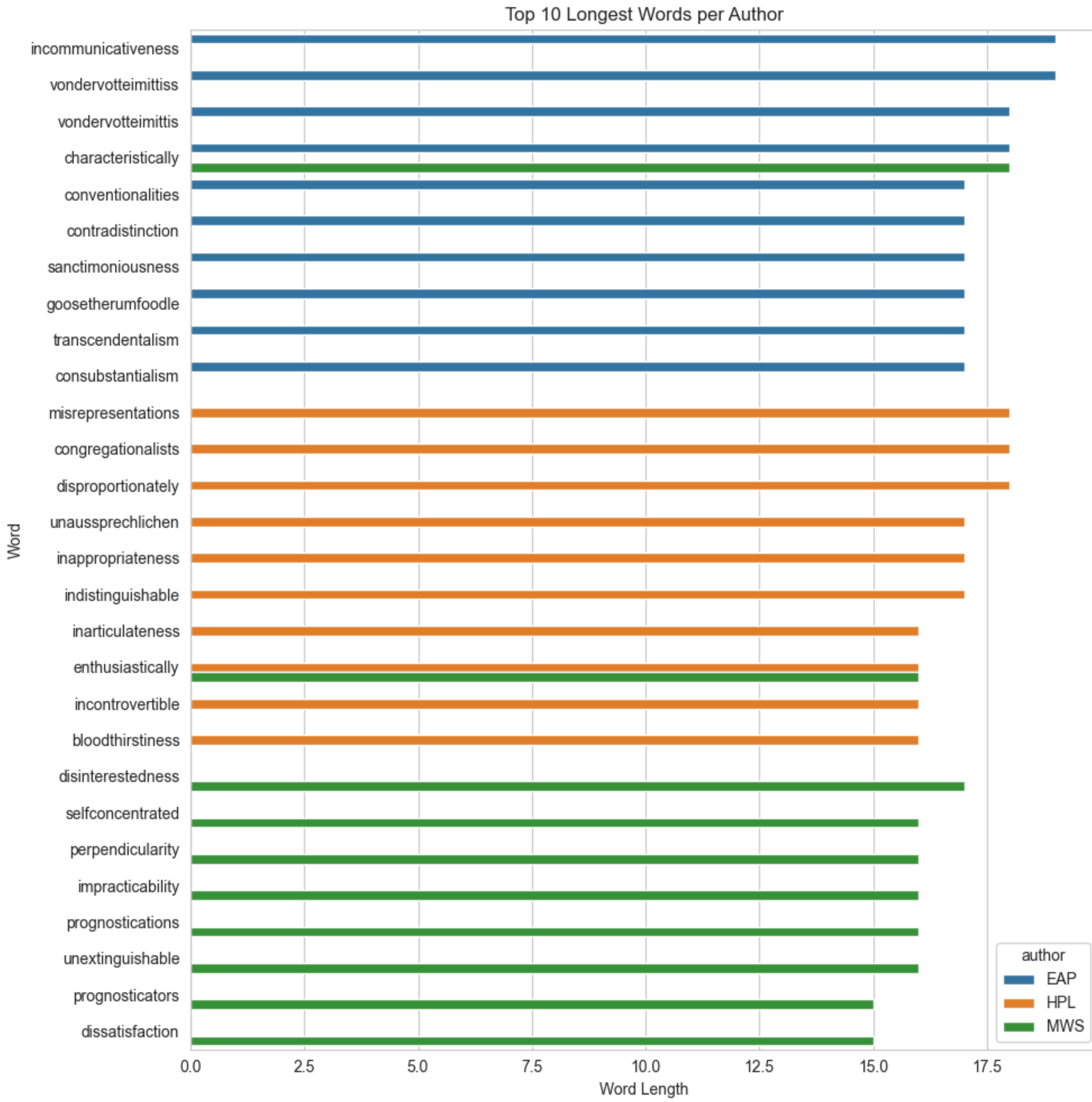
```python
# Stage 1 Analysis and Visualizations
from pyspark.sql.window import Window
from pyspark.sql import functions as F
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style("whitegrid")

# ---------- CHART 1: Most Frequent Word Lengths ----------
# Get word frequency
df_word_freq = df_train_filtered.groupBy("word").agg(count("*").alias("frequency"))

# Get top 30 most frequent words
df_top_words = df_word_freq.orderBy(col("frequency").desc()).limit(30)

# Convert to pandas for sns
pdf_top_words = df_top_words.toPandas()

# Plot Chart 1
plt.figure(figsize=(12, 6))
sns.barplot(data=pdf_top_words, x="word", y="frequency", color="skyblue")
plt.xticks(rotation=45, ha="right")
```

```
plt.title("Top 30 Most Frequent Non-Stopwords")
plt.xlabel("Words")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()

# ---------- CHART 2: Most Frequent Word Lengths ----------
# Get word lengths
df_word_lengths = df_train_filtered.withColumn("length", length(col("word")))

# Group by author and length, then count occurrences
df_grouped = df_word_lengths.groupBy("author", "length").agg(count("*").alias("coun

# Convert to pandas for sns
pdf_word_lengths = df_grouped.toPandas()

# Plot chart 2
plt.figure(figsize=(10, 6))
sns.barplot(data=pdf_word_lengths, x="length", y="count", hue="author")
plt.title("Most Frequent Word Lengths by Author (Excluding Stop Words)")
plt.xlabel("Word Length")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# ---------- CHART 3: Top 10 Longest Words per Author ----------
# Group by author and word, get length of word
df_longest = df_word_lengths.groupBy("author", "word") \
    .agg(count("*").alias("count"),
    F.max(length(col('word'))).alias('length')
)

# Rank words by length within each author
windowSpec = Window.partitionBy("author").orderBy(col("length").desc())

# Get top 10 words per author
df_top_longest = df_longest.withColumn("rank", row_number().over(windowSpec)).filte

# Convert to pandas for sns
pdf_longest = df_top_longest.select("author", "word", "length").toPandas()

# Plot chart 3
plt.figure(figsize=(10, 10))
sns.barplot(data=pdf_longest, x="length", y="word", hue="author")
plt.title("Top 10 Longest Words per Author")
plt.xlabel("Word Length")
plt.ylabel("Word")
plt.tight_layout()
plt.show()

# ----- CHART 4: count unique words by author ----------
# Get unique words per author
df_unique_words = df_train_filtered.select("author", "word").distinct()

# Count the unique words per author
df_word_diversity = df_unique_words.groupBy("author").count().withColumnRenamed("co
```

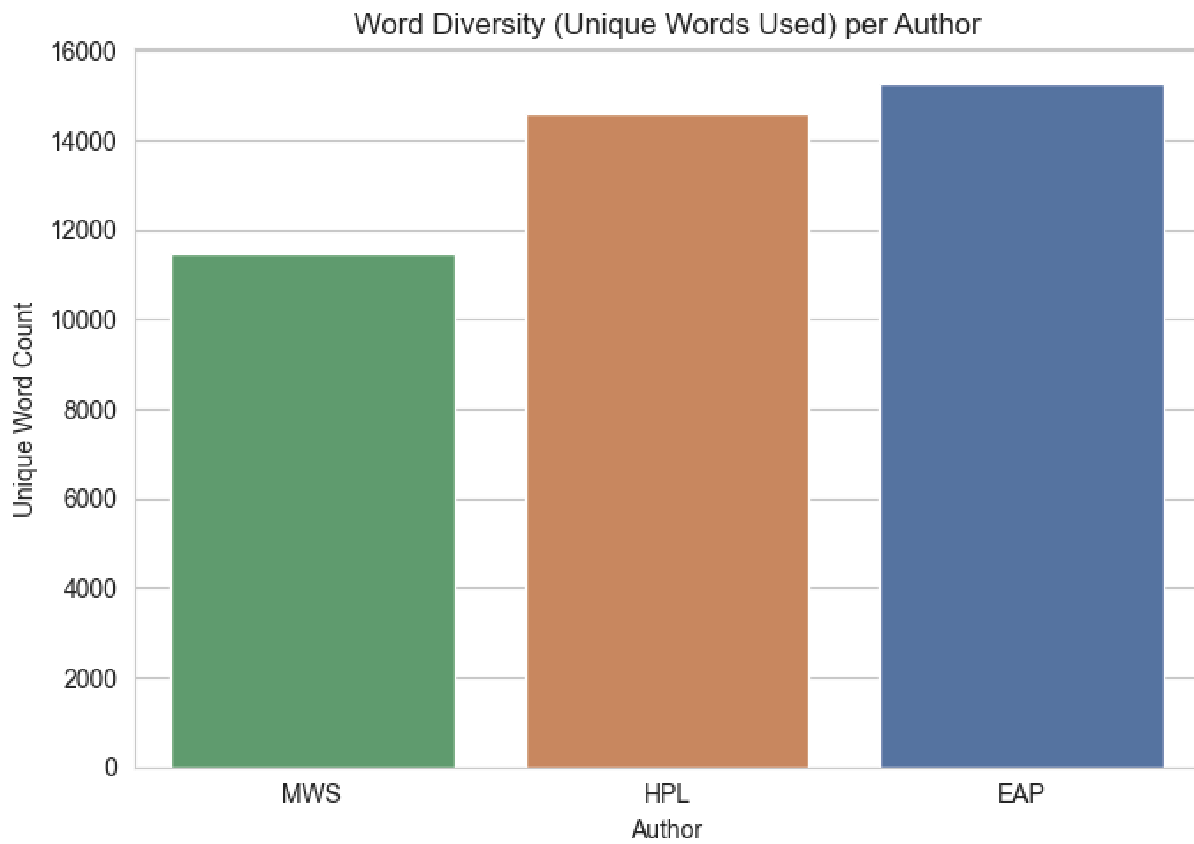
```
# Convert to pandas for sns
pdf_diversity = df_word_diversity.toPandas()

# Define custom color palette
palette = {"EAP": "#4C72B0", "HPL": "#DD8452", "MWS": "#55A868"}

# Plot chart 4
plt.figure(figsize=(7, 5))
sns.barplot(data=pdf_diversity, x="author", y="unique_word_count", hue="author", pa
plt.title("Word Diversity (Unique Words Used) per Author")
plt.xlabel("Author")
plt.ylabel("Unique Word Count")
plt.tight_layout()
plt.show()
```







Top 10 Longest Words per Author
incommunicativeness
vondervotteimittiss
vondervotteimittis
characteristically
conventionalities
contradistinction
sanctimoniousness
goosetherumfoodle
transcendentalism
consubstantialism
misrepresentations
congregationalists
disproportionately
unaussprechlichen
inappropriateness
indistinguishable
inarticulateness
enthusiastically
incontrovertible
bloodthirstiness
disinterestedness
selfconcentrated
perpendicularity
impracticability
prognostications
unextinguishable
prognosticators
dissatisfaction
Word
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Word Length
author
EAP
HPL
MWS



Stage 2 - Feature Extraction

1. Perform TFIDF to quantify word importance
   https://en.wikipedia.org/wiki/Tf%E2%80%93idf
2. Normalize is scaling or standardizing the numerical features to a standard range or distribution
   - In text mining, normalization vectorizes features with methods like TFIDF, a numerical measurement, to ensure a consistent scale
   - It handles variations in the magnitude of feature values impacting machine-learning algorithm performance. Normalize the features to ensure a similar scale and prevent features with larger values from dominating the analysis or modeling process

In [5]:

```
# Stage 2 - TFIDF and Normalization
from pyspark.sql.functions import col, collect_list, struct, first
from pyspark.ml.feature import HashingTF, IDF
from pyspark.ml.feature import Normalizer
from pyspark.sql.types import StringType, ArrayType

# Aggregate words into list per id, author (currently they are a single field per r
df_grouped = df_train_filtered.groupBy("id", "author").agg(
    collect_list("word").alias("words"),
    first('clean_text').alias('clean_text')
)
```

```
# Compute HashingTF
hashingTF = HashingTF(inputCol='words', outputCol='tf', numFeatures=4096) # I selec
tf_data_train = hashingTF.transform(df_grouped)

# Compute IDF
idf = IDF(inputCol='tf', outputCol='tfidf', minDocFreq=3)
idf_model_train = idf.fit(tf_data_train)
tfidf_data_train = idf_model_train.transform(tf_data_train)

# Normalize the data
normalizer = Normalizer(inputCol='tfidf', outputCol='tfidf_norm', p=2.0)
tfidf_data_train = normalizer.transform(tfidf_data_train)

# Drop unneeded columns and show a few rows
tfidf_data_train = tfidf_data_train.drop('tf', 'words')
tfidf_data_train.select('tfidf').show(truncate=False)
```

```
+---------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------+
|tfidf
|
+---------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------+
|(4096,[1443,2605,3302,3695,3950],[4.59384365608207,3.5399892680553915,6.0979210528
58282,4.962129760948418,5.538305264922859])
|
|(4096,[102,167,1204,1370,1520,2934,3433,3812],[5.086320141179802,4.313766183015446,
6.144441068493174,6.746616470847393,3.5208082092035475,3.342524731158873,4.933350796
398375,7.802669145096707])
|
|(4096,[205,433,455,630,1024,1097,1185,1355,1369,1607,1650,1667,1844,1942,1982,2510,
2528,2608,2807,3168,3327,3383,3417,3629,3710,3909,4027],[4.764116874359788,4.0071799
55924512,6.355750162160382,4.053165069166336,5.1029871936650135,4.007179955924512,6.
624014148755061,6.193231232626064,4.431072233210842,5.451293887933229,5.56462257324
0232,3.963216832503396,5.578045593572373,5.154722868064202,5.297143208105971,7.04889
7342720326,5.181630320984127,7.048897342720326,5.970087681348397,6.218549040646896,
5.874777501544072,4.940468264167238,6.144441068493174,5.619430809735227,5.3823010164
46277,7.243053357161284,5.7077234168809055])
|
|(4096,[2194,2357],[5.338815904506538,5.551377346490212])
|
|(4096,[131,147,352,380,389,419,574,726,883,942,1120,1520,1602,1608,1728,1903,2011,2
273,2279,2376,2589,2761,2768,2801,2835,2868,2973,3106,3115,3148,3202,3283,3286,3875,
3921],[4.4571606692951 4,4.683613655510717,5.8567589960413935,5.804573242870823,4.819
515653749576,3.4222062323990077,5.73897596038501,4.389049243435994,4.23666378913330
5,5.086320141179802,6.624014148755061,3.5208082092035475,4.578805778717467,5.5126628
34309521,6.416374783976816,4.298614377994843,3.4222062323990077,4.247321083607293,5.
404773872298336,6.746616470847393,5.578045593572373,4.764116874359788,6.416374783976
816,5.190762804547399,7.048897342720326,4.700327136484457,5.463270078979945,6.271192
774132318,3.8347385077302647,4.466010284572122,5.821667676230123,6.031963085066484,
3.870843512372381,5.172580485464208,6.144441068493174])|
|(4096,[507,535,560,742,860,1308,1366,1587,1644,1932,1960,2124,2761,2777,2856,3159,3
338,3654,3665,3676,3751,4032],[5.59165124562815 1,4.7581467073732835,5.78776612455444
2,4.384942461483341,6.624014148755061,3.557751724395232,3.9185313431580964,4.2473210
83607293,5.053796949474242,6.991738928880378,6.298591748320432,5.7549763017314515,4.
764116874359788,5.787766124554442,4.073968196796098,6.514814856790068 6,5.06182912117
1506,7.39720403698854 2,4.947636753645851,4.11378969098277,3.6795751695886203,5.99029
```

0388665916])
|
|(4096,[551,748,935,1209,1525,2319,2566,2961,3154,3177,3372,3532,3894,4077],[5.11142
6062310878,5.257137873492272,6.298591748320432,7.317161329315006,5.662602981600436,
3.839477853094161,5.393474317044403,7.802669145096707,5.061829121171506,6.7466164708
47393,4.683613655510717,5.950285054052217,4.711626691738391,6.746616470847393])
|
|(4096,[358,390,1727,2363,2478,3150],[3.6516292391980607,6.326762625287129,5.2377197
8763517,5.8567589960413935,3.996006655326387,5.551377346490212])
|
|(4096,[1010,1272,1353],[7.1095219645367616,5.086320141179802,4.116919583991698])
|
|(4096,[628,898,1239,1353,1366,1496,1882,2350,3263,3323,3540,3725,3736],[4.291123706
265686,5.591651245628151,4.317590279453849,4.116919583991698,3.9185313431580964,6.24
4524527050157,7.1095219645367616,5.427763390523035,5.163611815481448,5.6626029816004
36,5.970087681348397,5.257137873492272,4.329151101854925])
|
|(4096,[600,937,2308,2840,3051,3864],[4.598906958038554,3.7998917764000963,4.0184795
111784455,4.233136448615337,5.3282337951760015,4.520818521067118])
|
|(4096,[106,303,485,705,1308,1492,1820,1920,2532,2613,2640,2771,2813,2964,3071,3151,
3451,3497,3539,3579,3636],[4.474938915316424,5.930866968195115,3.585001366842607,5.9
30866968195115,3.557751724395232,6.326762625287129,5.525401860086951,4.5542345179869
62,4.926283629175282,5.2669901699352835,5.181630320984127,6.168538620072235,6.053469
290287448,6.704056856428597,4.912297387200542,4.418278881750933,4.161798910169131,3.
9686076811382724,4.7581467073732835,5.8567589960413935,4.933350796398375])
|
|(4096,[155,496,937,976,1044,1231,1371,1675,1760,2032,2612,2733,2880,3131,3859],[6.1
68538620072235,4.912297387200542,3.7998917764000963,4.053165069166336,4.435373315110
232,4.7581467073732835,5.2869908366419525,6.326762625287129,3.8538321665458444,5.102
9871936650135,6.168538620072235,3.242234852950007,5.500084052102661,4.57880577871746
7,4.554234517986962])
|
|(4096,[231,344,787,796,822,1346,1835,2359,2508,2620,2690,2949,3411,3458,3645,4013],
[6.5148148567900686,6.168538620072235,4.563990692932326,5.23771978763517,5.371251180
2596925,7.684886109440323,6.480913305114387,7.579525593782497,6.83758824905312,4.926
283629175282,5.564622573240232,5.633615444727184,5.3712511802596925,3.42064251042282
53,6.075448197006223,6.075448197006223])
|
|(4096,[1100,1168,1290,1389,1750,1787,1858,2580,2934,3329,3744,3749,3776,3798,3890,3
975],[4.640363671716901,4.838685569857296,4.764116874359788,3.493549281230913,6.0109
096758686515,12.106938580574896,5.317762495308706,4.661754861698218,3.34252473115887
3,4.891678099997806,5.094618943994496,5.061829121171506,4.5302525533004765,5.8931266
40212269,5.297143208105971,5.257137873492272])
|
|(4096,[389,1080,1520,1551,2041,2624,2837,2840,2883,2957,3056,3139,3601],[4.81951565
3749576,3.788540916731407,3.5208082092035475,5.218671592664475,7.802669145096707,5.1
19936751978787,4.444031377853348,4.233136448615337,3.941939434056111,6.1932312326626
064,5.692455944750117,5.3282337951760015,5.821667676230123])
|
|(4096,[27,1629,1985,3092],[5.037923600317951,6.3856031253100625,4.053165069166336,
5.037923600317951])
|
|(4096,[31,121,123,370,1149,1343,1369,1575,2217,2532,2660,2687,2773,2934,3001,3079],
[3.5070858669484464,5.338815904506538,5.538305264922859,3.9185313431580964,6.0534692
90287448,2.566892297023568,4.431072233210842,4.80070632179208,4.098285504446805,4.92

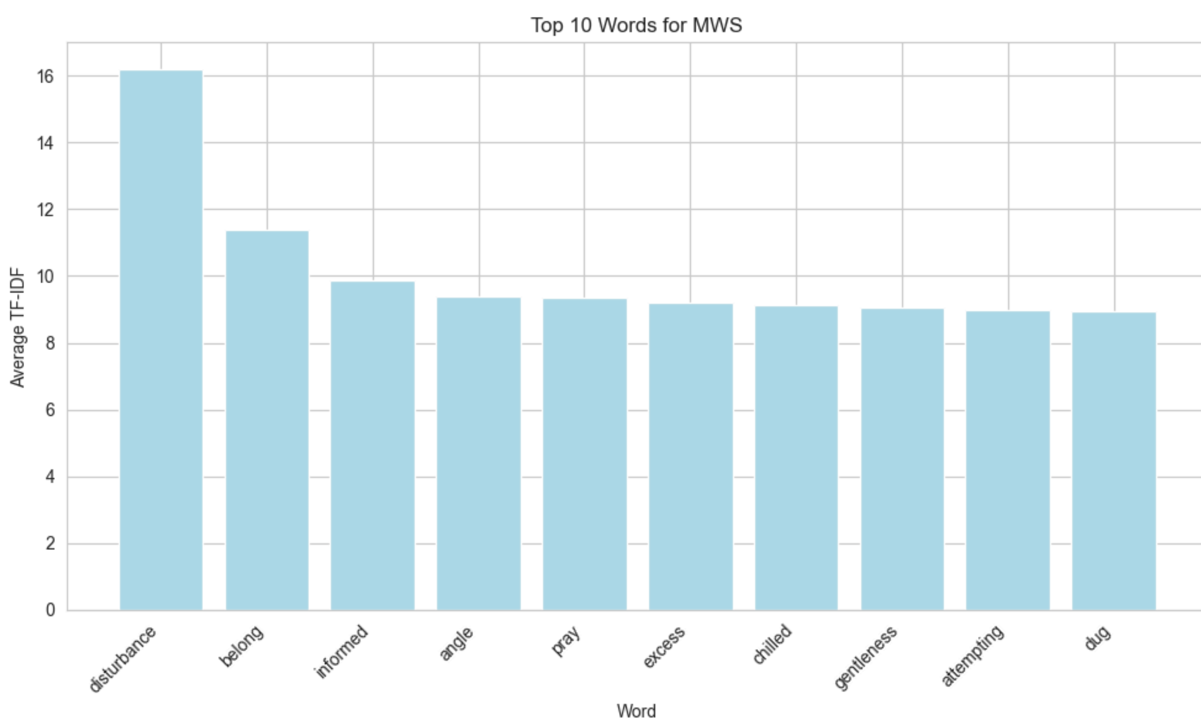
```
6283629175282,3.5019881498767775,2.8285249595828663,3.3126292663622467,3.34252473115
8873,5.512662834309521,3.7530604767159974])
|
|(4096,[612,804,892,1051,1104,1493,2160,2422,2830,3215,3350,3783,3964],[4.2156839986
64111,6.053469290287448,5.525401860086951,6.0109096758686515,5.564622573240232,5.328
2337951760015,4.089097078392399,5.2869908366419525,4.740447130273883,6.8375882490531
2,6.991738928880378,4.333034601881323,4.832254679527005])
|
|(4096,[755,1188,1328,1381,2247,2291,2883,3256,3292,3567],[5.137178558413292,5.30739
9708273159,2.7377035064554036,4.5302525533004765,5.970087681348397,5.66260298160043
6,3.941939434056111,4.954857001619338,5.564622573240232,4.838685569857296])
|
+-----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----------------------------------------------------+
only showing top 20 rows
```

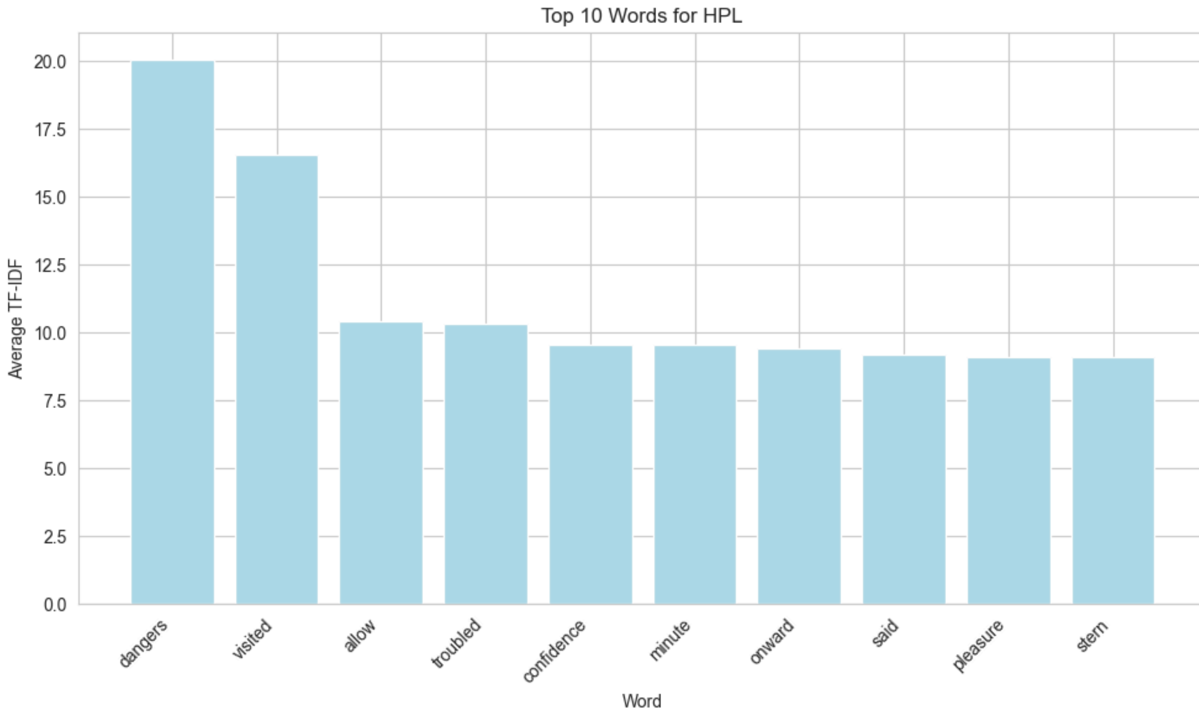
The data has this structure: `[Vector length], [indicies], [tf-idf values]`

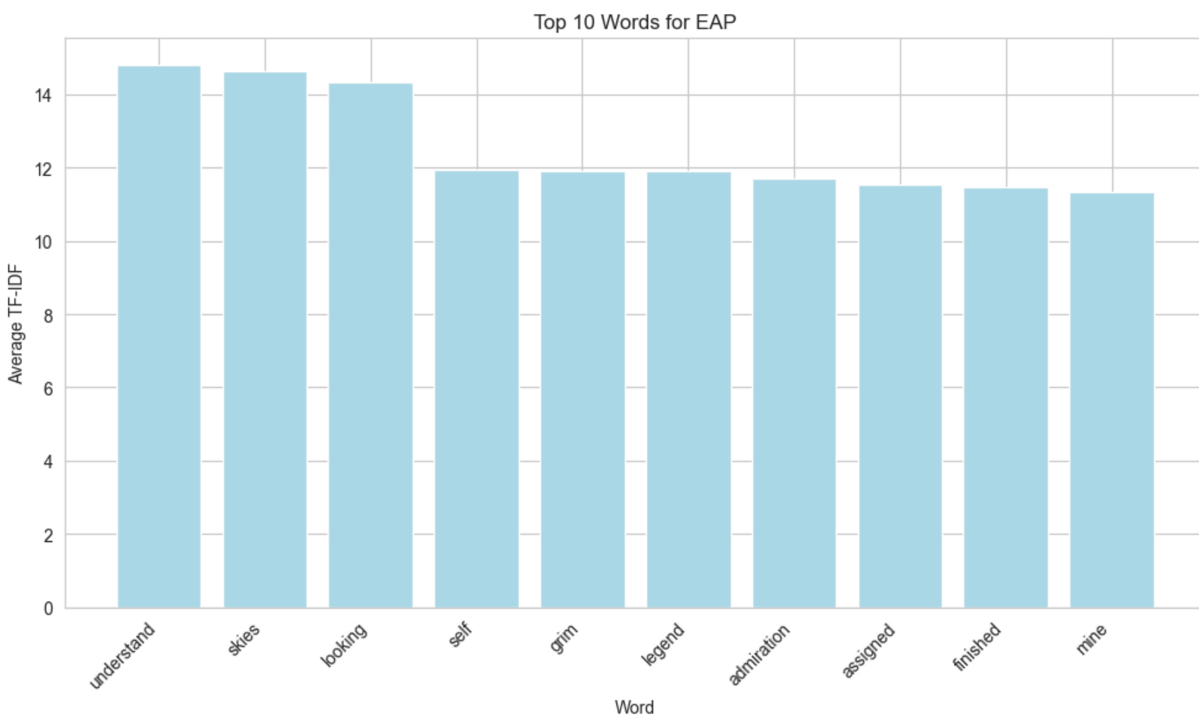
In [6]:

```python
# Stage 2 Visualizations (ex: Most Important Word By Author)
from pyspark.ml.feature import CountVectorizer, IDF
from pyspark.sql.functions import col
import pandas as pd
import matplotlib.pyplot as plt
from collections import defaultdict

# Redo TFIDF: Need to use CountVectorizer here to retain the words for future analy
cv = CountVectorizer(inputCol='words', outputCol='tf_cv', minDF=3.0, vocabSize=4096
cv_model = cv.fit(tf_data_train)
tf_cv_data = cv_model.transform(tf_data_train)
idf_cv = IDF(minDocFreq=3, inputCol='tf_cv', outputCol='tfidf')
idf_cv_model = idf.fit(tf_cv_data)
tfidf_cv_data = idf_cv_model.transform(tf_cv_data).drop('tf_cv')
tfidf_cv_data.cache()

# Here I have to switch to using Pandas, as Spark would have timeout issues when tr
tfidf_pandas = tfidf_cv_data.select('author', 'tfidf').toPandas()

# Group the tfidf vectors per author
top_words_per_author = defaultdict(list)
for _, row in tfidf_pandas.iterrows():
    author = row['author']
    vector = row['tfidf']
    for index, value in zip(vector.indices, vector.values):
        top_words_per_author[author].append((index, value))

# Now compute the average tfidf value per word per author
avg_tfidf_per_author = {}
vocab = cv_model.vocabulary
```

```
for author, terms in top_words_per_author.items():
    # For each term get the sum and counts
    index_sums = defaultdict(lambda: {'sum': 0.0, 'count': 0})
    for index, value in terms:
        index_sums[index]['sum'] += value
        index_sums[index]['count'] += 1

    # Now compute the averages
    avg_tfidf = [(vocab[index], data['sum'] / data['count']) for index, data in ind

    # Sort averages by descending average value (element 1 in avg_tfidf above) and
    avg_tfidf = sorted(avg_tfidf, key=lambda x: x[1], reverse=True)[:10]

    # Assign them to the correct author
    avg_tfidf_per_author[author] = pd.DataFrame(avg_tfidf, columns=['word', 'avg(va

# Plot the best words per author
for author, df in avg_tfidf_per_author.items():
    plt.figure(figsize=(10, 6))
    plt.bar(df['word'], df['avg(value)'], color='lightblue')
    plt.xlabel('Word')
    plt.ylabel('Average TF-IDF')
    plt.title(f'Top 10 Words for {author}')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```







Stage 3 - Machine Learning

1. Perform train/test split
2. Perform algorithmic analysis to assess and predict test labels
   - Use as many algorithms as you need to get a good answer.
   - Supervised: logistic regression, random forest, support vector machines, etc.
   - Unsupervised: K-means, dimensionality reduction, PCA, etc.

In [7]:
```python
# Stage 3 Solution (Due by Monday 7/21)
# Each team member will do 2 algorithms of their choosing

# Train test split for below
train_data, test_data = tfidf_data_train.randomSplit([0.7, 0.3], seed=42)
train_data.cache()
test_data.cache()
print(f"Training set size: {train_data.count()} rows")
print(f"Test set size: {test_data.count()} rows")
```

```
Training set size: 13608 rows
Test set size: 5968 rows
```

In [8]:
```python
# Aidan: Logistic Regression, Agglomerative Heirarchical Clustering

# ---------- Logistic Regression ----------
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import StringIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Convert author labels to numeric index
indexer = StringIndexer(inputCol='author', outputCol='label')
indexer_model = indexer.fit(train_data)
indexed_train = indexer_model.transform(train_data)
indexed_test = indexer_model.transform(test_data)
indexed_train.cache()
indexed_train.count() # Force the train data to cache

# Define model
lr = LogisticRegression(featuresCol='tfidf', labelCol='label', maxIter=1000)

# Try to tune hyper params
paramGrid = ParamGridBuilder() \
    .addGrid(lr.regParam, [1/c for c in [0.1, 1, 10]]) \
    .build()
evaluator = MulticlassClassificationEvaluator(labelCol='label', predictionCol='pred
cv = CrossValidator(estimator=lr,
                    estimatorParamMaps=paramGrid,
                    evaluator=evaluator,
                    numFolds=3,
                    parallelism=4)
cv_model = cv.fit(indexed_train)
lr_model = cv_model.bestModel

# Predict test data and convert numerical labels back to authors
lr_predictions = lr_model.transform(indexed_test)

label_mapping = indexer_model.labels  # Get mapping of indices to author names (e.g
lr_predictions = lr_predictions.withColumn("predicted_author",
    col("prediction").cast("integer").cast("string"))  # Convert prediction to stri
lr_predictions = lr_predictions.replace(
    to_replace={str(i): label_mapping[i] for i in range(len(label_mapping))},
    subset=["predicted_author"]
)
```

```
# Print top 5 predictions
print("---------------- Logistic Regression Predictions ----------------")
lr_predictions.select("clean_text", "author", "label", "prediction").show(5, trunca

# ---------- Agglomerative Heirarchical Clustering (Using Bisecting KMeans) -------
from pyspark.ml.clustering import BisectingKMeans

# Train the Bisecting KMeans model
bkm = BisectingKMeans(featuresCol='tfidf', k=3, seed=42)
bkm_model = bkm.fit(train_data)

# Cluster predictions on train and test data
bkm_train_predictions = bkm_model.transform(train_data)
bkm_test_predictions = bkm_model.transform(test_data)

# Show sample cluster assignments for training data
print("\n\n---------------- Train Data Cluster Assignments ----------------")
bkm_train_predictions.select("id", "clean_text", "author", "prediction").show(5, tr

# Show sample cluster assignments for test data
print("\n\n---------------- Test Data Cluster Assignments ----------------")
bkm_test_predictions.select("id", "clean_text", "prediction").show(5, truncate=Fals
```

```
---------------- Logistic Regression Predictions ----------------
+--------------------------------------------------------------------------------------------------------------------------------------+------+-----+----------+
|clean_text
|author|label|prediction|
+--------------------------------------------------------------------------------------------------------------------------------------+------+-----+----------+
|soon they became excessively numerous like impious catacombs of nameless menace and their pungent odour of decay grew quite unbearable|HPL   |2.0  |0.0       |
|he looks to number one
|EAP   |0.0  |0.0       |
|the other face may wear off some
|HPL   |2.0  |0.0       |
|write il pover huomo che non sen era accorto andava combattendo e era morto thats italian you perceive from ariosto                    |EAP   |0.0  |0.0       |
|it seemed in any event to be contagious for hannah bowen one of the two servants died of it in the following june                     |HPL   |2.0  |1.0       |
+--------------------------------------------------------------------------------------------------------------------------------------+------+-----+----------+
only showing top 5 rows


---------------- Train Data Cluster Assignments ----------------
+-------+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+------+----------+
|id     |clean_text
|author|prediction|
+-------+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+------+----------+
|id00070|yet even then i have checked thick coming fears with one thought i would not fear death for the emotions that linked us must be immortal
|MWS   |2         |
|id00093|seems that human folks has got a kind o relation to sech water beasts that everything alive come aout o the water onct an only needs a little change to go back agin                             |HPL   |2         |
|id00361|this spirit existed as a breath a wish a far off thought until communicated to adrian who imbibed it with ardour and instantly engaged himself in plans for its execution                         |MWS   |1         |
|id01530|still i would not hurry on i would pause for ever on the recollections of these happy weeks i would repeat every word and how many do i remember record every enchantment of the faery habitation|MWS   |2         |
|id01661|it first rolled down the side of the steeple then lodge for a few seconds in the gutter and then made its way with a plunge into the middle of the street
|EAP   |1         |
+-------+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+------+----------+
only showing top 5 rows


---------------- Test Data Cluster Assignments ----------------
+-------+------------------------------------------------------------------------------------------------------------------------------+----------+
|id     |clean_text
```

```
|prediction|
+-------+--------------------------------------------------------------------------
-----------------------------------------------------------+----------+
|id00095|soon they became excessively numerous like impious catacombs of nameless me
nace and their pungent odour of decay grew quite unbearable|1         |
|id01803|he looks to number one
|0         |
|id02318|the other face may wear off some
|0         |
|id02936|write il pover huomo che non sen era accorto andava combattendo e era morto
thats italian you perceive from ariosto                   |0         |
|id04131|it seemed in any event to be contagious for hannah bowen one of the two ser
vants died of it in the following june                     |1         |
+-------+--------------------------------------------------------------------------
-----------------------------------------------------------+----------+
only showing top 5 rows
```

In [9]:

```
# Daniel: K-means
from pyspark.ml import Pipeline
from pyspark.ml.feature import PCA
from pyspark.ml.clustering import KMeans

# pca = PCA(k=10, inputCol="tfidf_norm", outputCol="tfidf_norm_pca")

# train_data_pca = pca.fit(train_data).transform(train_data)

# I'd previously attempted to use a PCA, but now with the
# hashingtf feature count being higher, I cannot run.

kmeans = KMeans(k=3,featuresCol='tfidf_norm')

k_means_model = kmeans.fit(train_data)

k_means_result = k_means_model.transform(train_data)
```

In [10]:

```
# Daniel: Neural network
from pyspark.ml.classification import MultilayerPerceptronClassifier

layers = [hashingTF.getNumFeatures(), 50, 12, 6, 6, 3]
#layers = [hashingTF.getNumFeatures(), 25, 6, 3, 3, 3]

# create the trainer and set its parameters
nn_trainer = MultilayerPerceptronClassifier(maxIter=150, layers=layers,stepSize= 0.

nn_pipeline = Pipeline(stages=[
    indexer
    ,nn_trainer
])

# Fit and transform using same pipeline
nn_model = nn_pipeline.fit(train_data)
nn_results = nn_model.transform(train_data)
```

In [11]:

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```python
evaluator = MulticlassClassificationEvaluator(
    labelCol='label',
    predictionCol='prediction',
    metricName='accuracy'
)

accuracy = evaluator.evaluate(nn_results)
print(f'Training Accuracy: {accuracy:.4f}')
```

```
Training Accuracy: 0.8474
```

In [12]:

```python
# Claudine:  MulticlassClassification and LDA
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.feature import StringIndexer
from pyspark.ml.clustering import LDA
from pyspark.ml import Pipeline

# Prepare Data
data = tfidf_data_train.select("tfidf_norm", "author")

# Convert author names to label numbers
label_indexer = StringIndexer(inputCol="author", outputCol="label")
indexed_data = label_indexer.fit(data).transform(data)

# Train-test split
train_data, test_data = indexed_data.randomSplit([0.7, 0.3], seed=42)

# Supervised: Multilayer Perceptron
input_size = train_data.select("tfidf_norm").first()[0].size
layers = [input_size, 128, 64, 3]

mlp = MultilayerPerceptronClassifier(labelCol="label", featuresCol="tfidf_norm", ma
mlp_model = mlp.fit(train_data)
mlp_predictions = mlp_model.transform(test_data)

# Unsupervised: LDA
lda = LDA(k=3, seed=42, featuresCol="tfidf_norm")
lda_model = lda.fit(tfidf_data_train)
topics = lda_model.describeTopics(10)
topics.show(truncate=False)
```

```
+-----+------------------------------------------------------------+----------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-----------------------------------------+
|topic|termIndices                                                 |termWeights
|
+-----+------------------------------------------------------------+----------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-----------------------------------------+
|0    |[131, 3380, 2687, 383, 3081, 3676, 1389, 109, 991, 737]    |[0.00225306685838
2698, 0.0021549642717594, 0.0021539412945318906, 0.002016198647588732, 0.00183554223
7109634, 0.0018245919046211857, 0.0017976855321940226, 0.0017875834392742347, 0.0016
214113432608884, 0.0016102953337570054]  |
|1    |[1343, 1328, 2687, 383, 1040, 31, 419, 2773, 485, 2011]    |[0.00318598322413
0552, 0.0031553504790761333, 0.0024645380923397336, 0.0022792068775724806, 0.0021879
806098741602, 0.002081574591825907, 0.0020686686294866218, 0.001996043324088303, 0.0
01954779161938257, 0.001916491790465898] |
|2    |[1307, 1343, 383, 2660, 2687, 3257, 3930, 1424, 2856, 1587]|[0.00279246383685
64403, 0.002621661590321394, 0.0025252443984009093, 0.0025124291093043824, 0.0019972
97335766142, 0.001982440967930216, 0.0018045163039332257, 0.001682679463459467, 0.00
16734414476656321, 0.0016137434121452402]|
+-----+------------------------------------------------------------+----------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-----------------------------------------+
```

In [13]:

```python
# Radhika: Random Forest, NaiveBayes
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import StringIndexer
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Train Random Forest Classifier Model
rf = RandomForestClassifier(featuresCol='tfidf', labelCol='label', numTrees=4)
rf_model = rf.fit(indexed_train)
rf_predictions = rf_model.transform(indexed_test) # Predict test data and convert n
print("rf_predictions", rf_predictions.head())

# Train Naive Bayes Classifier Model
nb = NaiveBayes(featuresCol='tfidf', labelCol='label', modelType='multinomial') # I
nb_model = nb.fit(indexed_train) # Train the model
nb_predictions = nb_model.transform(indexed_test) # Make predictions on the test da
print("nb_predictions", nb_predictions.head())

# Print top 5 predictions
#print("---------------- PCA Predictions ----------------")
#train_pca.select("pca_features").show(5, truncate=False)
#pca_predictions.select("clean_text", "author", "label", "prediction").show(5, trun
```

```
rf_predictions Row(id='id00095', author='HPL', clean_text='soon they became excessively numerous like impious catacombs of nameless menace and their pungent odour of decay grew quite unbearable', tfidf=SparseVector(4096, {193: 5.0379, 294: 5.3388, 520: 4.7346, 892: 5.5254, 976: 4.0532, 1157: 4.5115, 1722: 6.624, 1733: 5.8568, 1899: 6.5499, 2865: 5.1547, 2972: 6.032, 3110: 6.3558, 3458: 3.4206, 3560: 4.1618, 3793: 4.2261}), tfidf_norm=SparseVector(4096, {193: 0.2473, 294: 0.2621, 520: 0.2324, 892: 0.2713, 976: 0.199, 1157: 0.2215, 1722: 0.3252, 1733: 0.2875, 1899: 0.3216, 2865: 0.2531, 2972: 0.2961, 3110: 0.312, 3458: 0.1679, 3560: 0.2043, 3793: 0.2075}), label=2.0, rawPrediction=DenseVector([1.6891, 1.2156, 1.0953]), probability=DenseVector([0.4223, 0.3039, 0.2738]), prediction=0.0)
nb_predictions Row(id='id00095', author='HPL', clean_text='soon they became excessively numerous like impious catacombs of nameless menace and their pungent odour of decay grew quite unbearable', tfidf=SparseVector(4096, {193: 5.0379, 294: 5.3388, 520: 4.7346, 892: 5.5254, 976: 4.0532, 1157: 4.5115, 1722: 6.624, 1733: 5.8568, 1899: 6.5499, 2865: 5.1547, 2972: 6.032, 3110: 6.3558, 3458: 3.4206, 3560: 4.1618, 3793: 4.2261}), tfidf_norm=SparseVector(4096, {193: 0.2473, 294: 0.2621, 520: 0.2324, 892: 0.2713, 976: 0.199, 1157: 0.2215, 1722: 0.3252, 1733: 0.2875, 1899: 0.3216, 2865: 0.2531, 2972: 0.2961, 3110: 0.312, 3458: 0.1679, 3560: 0.2043, 3793: 0.2075}), label=2.0, rawPrediction=DenseVector([-613.0272, -651.7795, -608.9266]), probability=DenseVector([0.0163, 0.0, 0.9837]), prediction=2.0)
```

Stage 4 - Evaluation and Visualization

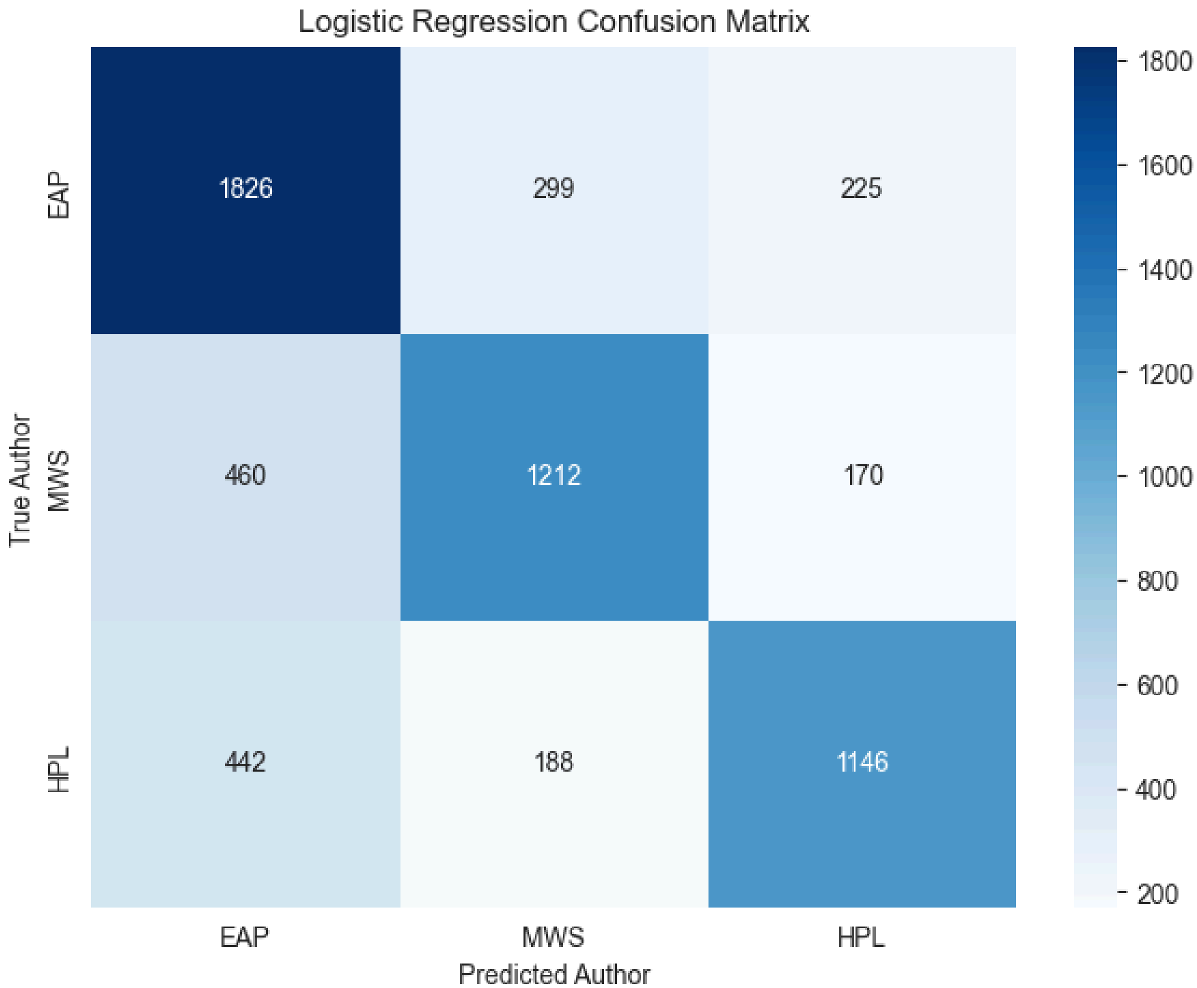
1. Choose a metric strategy to assess algorithmic performance like accuracy, precision, recall, or F1 score
2. Visualize confusion matrix, correlations, and similar
3. Identify important features contributing to classification
4. Write a 2-3 sentence minimum of findings, learnings, and what you would do next

In [14]:
```
# Stage 4 Solution (Due by Monday 7/21)
# Each team member will evaluate their models
```

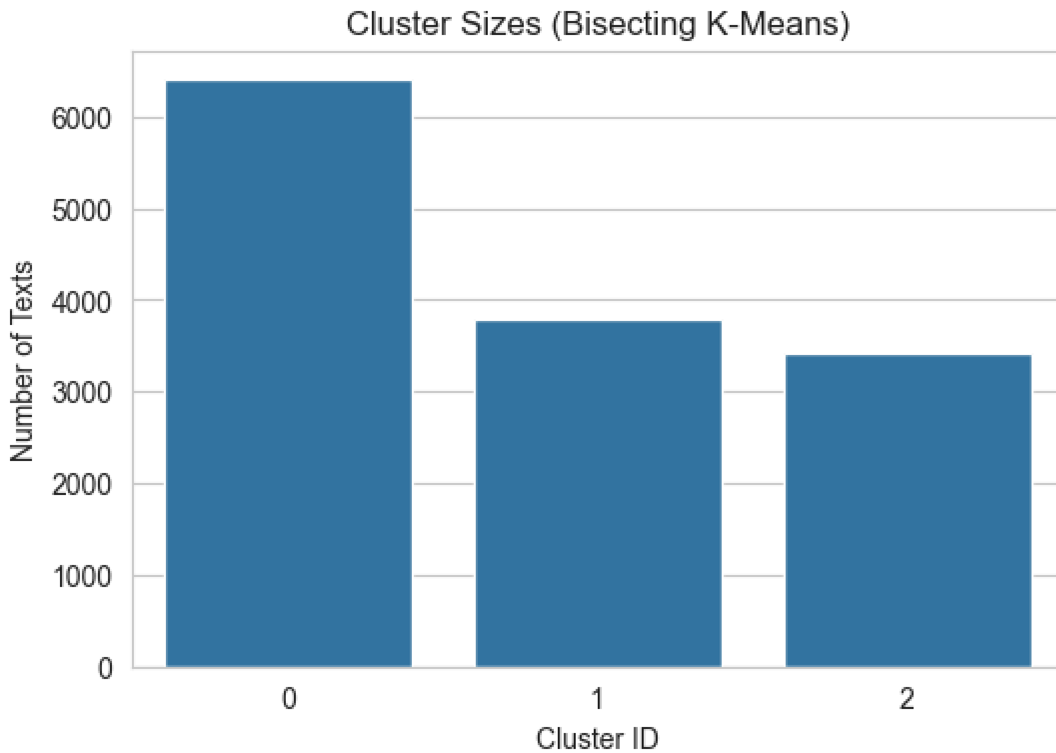
In [15]:
```
# Aidan
# ---------- Logistic Regression ----------
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

evaluator = MulticlassClassificationEvaluator(labelCol='label', predictionCol='pred
f1_score = evaluator.evaluate(lr_predictions)
print(f"Logistic Regression F1 Score: {f1_score}")

# Get confusion matrix
df_lr_predictions = lr_predictions.select('label', 'prediction').toPandas()
conf_mat = confusion_matrix(df_lr_predictions['label'], df_lr_predictions['predicti

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', xticklabels=label_mapping,
plt.title("Logistic Regression Confusion Matrix")
```

```
plt.xlabel("Predicted Author")
plt.ylabel("True Author")
plt.show()

# ---------- Agglomerative Heirarchical Clustering (Using Bisecting KMeans) -------
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.sql.functions import count

# Calculate silhouette score
evaluator = ClusteringEvaluator(featuresCol='tfidf', predictionCol='prediction')
silhouette = evaluator.evaluate(bkm_train_predictions)
print(f"Bisecting K-Means Silhouette Score: {silhouette}")

# Map clusters to authors
cluster_author_counts = bkm_train_predictions.groupBy("prediction", "author").agg(c
cluster_author_counts.show()

# Plot cluster sizes
cluster_counts_pd = bkm_train_predictions.groupBy("prediction").count().toPandas()
plt.figure(figsize=(6, 4))
sns.barplot(data=cluster_counts_pd, x="prediction", y="count")
plt.title("Cluster Sizes (Bisecting K-Means)")
plt.xlabel("Cluster ID")
plt.ylabel("Number of Texts")
plt.show()
```

```
Logistic Regression F1 Score: 0.7001024067235121
```




```
Bisecting K-Means Silhouette Score: -0.019673061092032554
+----------+------+-----+
|prediction|author|count|
+----------+------+-----+
|         0|   MWS| 1972|
|         1|   MWS| 1247|
|         2|   EAP| 1278|
|         2|   HPL| 1151|
|         0|   EAP| 3116|
|         0|   HPL| 1311|
|         2|   MWS|  982|
|         1|   HPL| 1397|
|         1|   EAP| 1154|
+----------+------+-----+
```



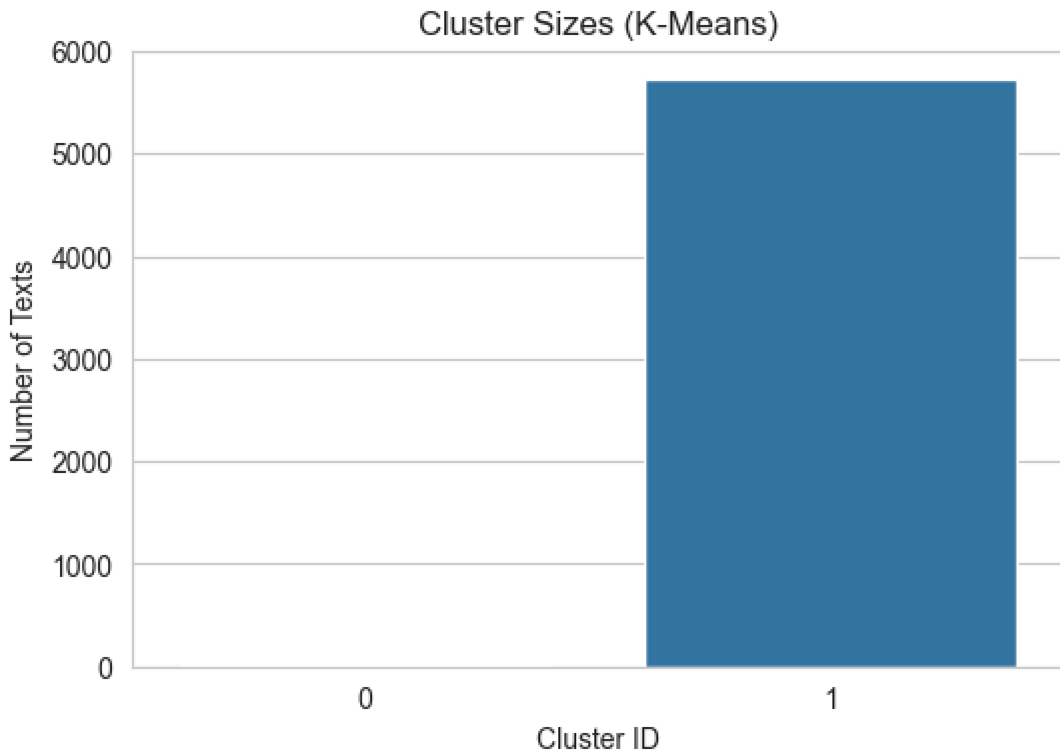
Aidan - Logistic Regression and Agglomerative Heirarchical Clustering (Bisecting K-Means)

1. Logistic regression did a good job of predicting the test data with minimal hyper param tuning (only 3 params and 3 folds, so 9 models), but it still only achieved ~70% for its accuracy score
   - Next I would dive into tuning this more in depth and perform a large cross validation with many different params and iteration counts
2. Bisecting K-Means did not perform well. As indicated by it's silhouette score of ~-0.01, the clusters are not well defined and probably overlap significantly.
   - I would not go any further with this algorithm for 2 reasons. It's unsupervised and would require a lot of preprocessing to get a tangible result. Since the data is labeled, a supervised algorithm would be a better fit here as seen by Logistic Regressions accuracy

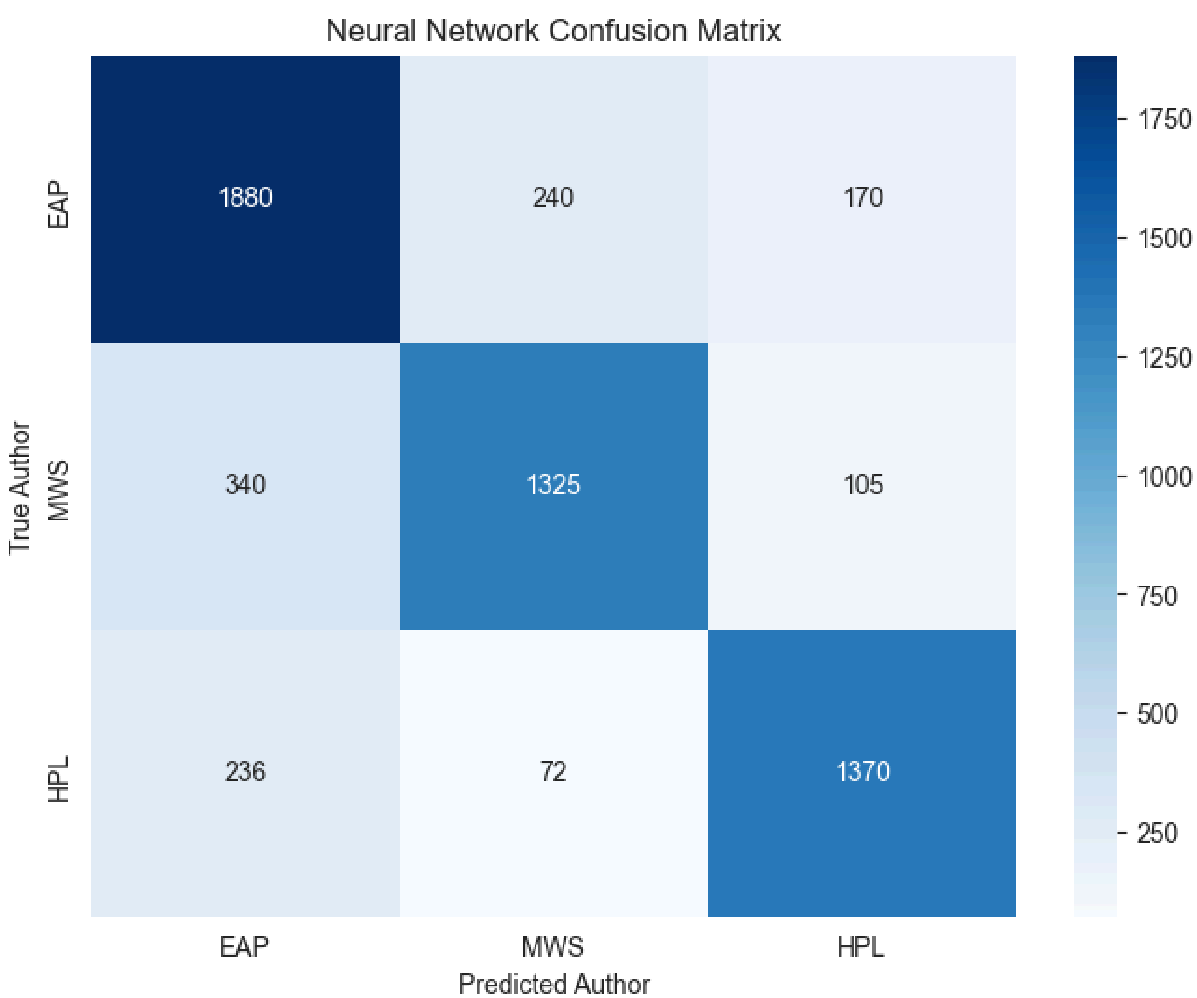
In [17]:
```
# Daniel
# -----------------    K-means      ---------------------------
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.sql.functions import count

km_test_predictions = k_means_model.transform(test_data)

# Calculate silhouette score
evaluator = ClusteringEvaluator(featuresCol='tfidf_norm', predictionCol='prediction
silhouette = evaluator.evaluate(km_test_predictions)
print(f"K-Means Silhouette Score: {silhouette}")

# Map clusters to authors
```

```
cluster_author_counts = km_test_predictions.groupBy("prediction", "author").agg(cou
cluster_author_counts.show()

# Plot cluster sizes
cluster_counts_pd = km_test_predictions.groupBy("prediction").count().toPandas()
plt.figure(figsize=(6, 4))
sns.barplot(data=cluster_counts_pd, x="prediction", y="count")
plt.title("Cluster Sizes (K-Means)")
plt.xlabel("Cluster ID")
plt.ylabel("Number of Texts")
plt.show()


# ----------------- Neural Network ---------------------------

test_data_nn = test_data.drop('label')
nn_test_predictions = nn_model.transform(test_data_nn)

nn_evaluator = MulticlassClassificationEvaluator(labelCol='label', predictionCol='p
acc_score = nn_evaluator.evaluate(nn_test_predictions)
print(f"Neural Network Accuracy Score: {acc_score}")

# Get confusion matrix
nn_test_predictions = nn_test_predictions.select('label', 'prediction').toPandas()
nn_conf_mat = confusion_matrix(nn_test_predictions['label'], nn_test_predictions['p

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(nn_conf_mat, annot=True, fmt='d', cmap='Blues', xticklabels=label_mappi
plt.title("Neural Network Confusion Matrix")
plt.xlabel("Predicted Author")
plt.ylabel("True Author")
plt.show()
```

```
K-Means Silhouette Score: 0.0006583631811794777
+----------+------+-----+
|prediction|author|count|
+----------+------+-----+
|         0|   MWS|   12|
|         1|   MWS| 1758|
|         0|   EAP|    6|
|         0|   HPL|    2|
|         1|   HPL| 1676|
|         1|   EAP| 2284|
+----------+------+-----+
```



Neural Network Accuracy Score: 0.7972417463473579



Daniel Lillard Analysis - K means, MultilayerPerceptronClassifier

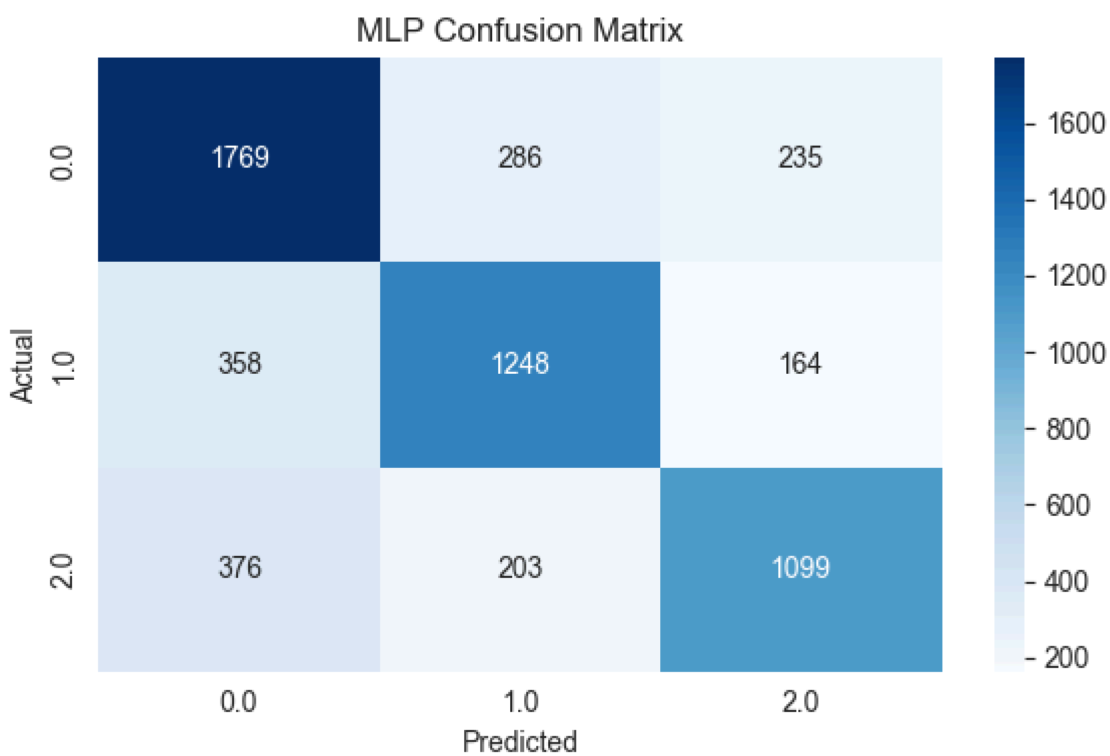
1. K means: This algorithm ended up doing very poorly, it grouped all the text into one single cluster, there would have to be some feature engineering done, I had tried to do a PCA, however we had to reduce the TF number of features. We would have to find a way to explode the differences between authors for this to be viable.

2. MultilayerPerceptronClassifier: It seems that this is a neural network. I used Keras in a previous class and found it easier to work with then this, I suppose that Spark really does make things harder! I was able to get around a 70% accuracy with minimal effort, NN's are good function approximators and I know can *theoretically* solve for any function. Some feature engineering and hyper-parameter tuning should make this model viable, unlike k-means.

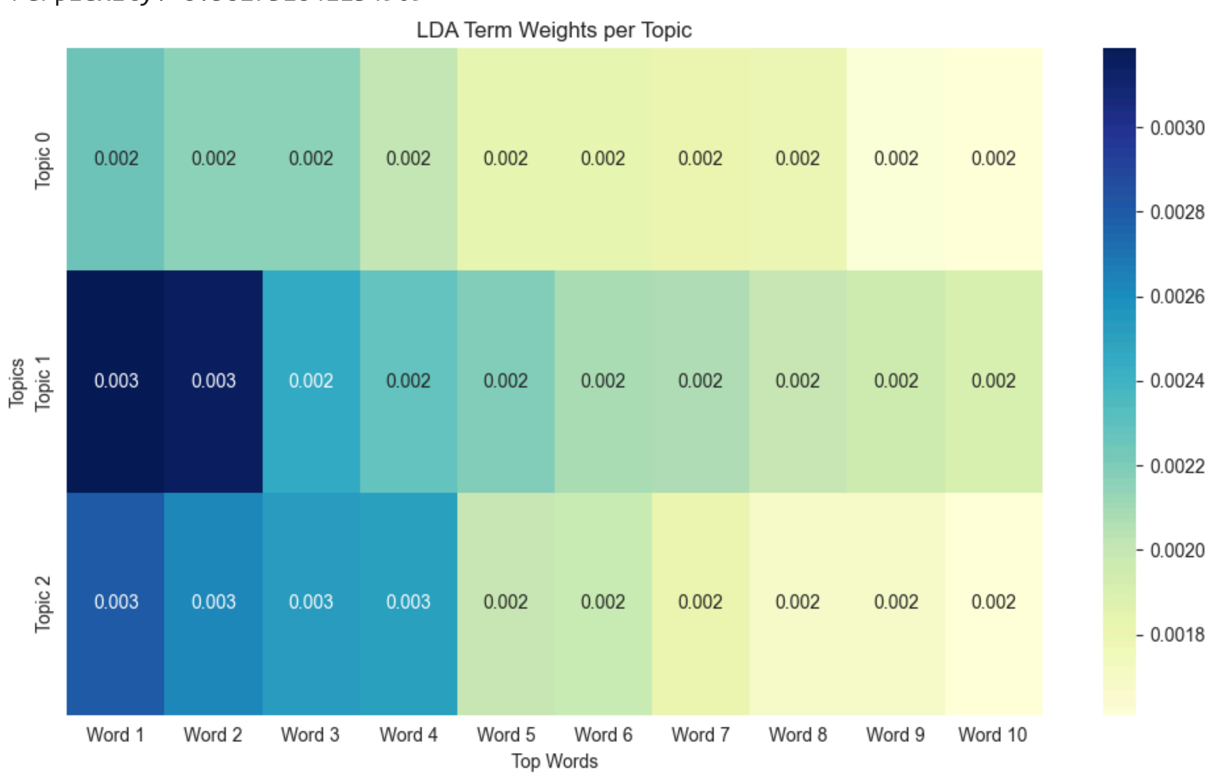
In [18]:
```
# Claudine: Stage 4 Evaluation
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# MLP Classifier
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="pred

accuracy = evaluator.evaluate(mlp_predictions, {evaluator.metricName: "accuracy"})
precision = evaluator.evaluate(mlp_predictions, {evaluator.metricName: "weightedPre
recall = evaluator.evaluate(mlp_predictions, {evaluator.metricName: "weightedRecall
f1 = evaluator.evaluate(mlp_predictions, {evaluator.metricName: "f1"})

print("Multilayer Perceptron Performance:")
print(f"Accuracy:  {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1 Score:  {f1:.4f}")

# Confusion Matrix
conf_matrix_df = mlp_predictions.select("label", "prediction").toPandas()
conf_matrix = pd.crosstab(conf_matrix_df["label"], conf_matrix_df["prediction"], ro

# Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.title("MLP Confusion Matrix")
plt.tight_layout()
plt.show()

# LDA Topic Interpretation
# Show top 10 term indices and term weights for each topic
print("Top 10 Words per Topic from LDA Model:")
topics.show(truncate=False)

print("LDA Evaluation Metrics:")
print("Log Likelihood:", lda_model.logLikelihood(tfidf_data_train))
print("Perplexity:", lda_model.logPerplexity(tfidf_data_train))

# Convert topics to Pandas DataFrame for heatmap
```

```
topic_words_df = topics.toPandas()

# heatmap of term weights
term_matrix = pd.DataFrame(topic_words_df['termWeights'].to_list(),
                           index=[f"Topic {i}" for i in topic_words_df['topic'].tol
term_matrix.columns = [f"Word {i+1}" for i in range(term_matrix.shape[1])]

plt.figure(figsize=(10, 6))
sns.heatmap(term_matrix, annot=True, cmap="YlGnBu", fmt=".3f")
plt.title("LDA Term Weights per Topic")
plt.ylabel("Topics")
plt.xlabel("Top Words")
plt.tight_layout()
plt.show()
```

```
Multilayer Perceptron Performance:
Accuracy:  0.7173
Precision: 0.7182
Recall:    0.7173
F1 Score:  0.7165
```



```
Top 10 Words per Topic from LDA Model:
+-----+-----------------------------------------------------------+----------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----------------------------------------+
|topic|termIndices                                                |termWeights
|
+-----+-----------------------------------------------------------+----------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----------------------------------------+
|0    |[131, 3380, 2687, 383, 3081, 3676, 1389, 109, 991, 737]    |[0.00225306685838
2698, 0.0021549642717594, 0.0021539412945318906, 0.002016198647588732, 0.00183554223
7109634, 0.0018245919046211857, 0.0017976855321940226, 0.0017875834392742347, 0.0016
214113432608884, 0.0016102953337570054]  |
|1    |[1343, 1328, 2687, 383, 1040, 31, 419, 2773, 485, 2011]    |[0.00318598322413
0552, 0.0031553504790761333, 0.0024645380923397336, 0.0022792068775724806, 0.0021879
806098741602, 0.002081574591825907, 0.0020686686294866218, 0.001996043324088303, 0.0
01954779161938257, 0.001916491790465898] |
|2    |[1307, 1343, 383, 2660, 2687, 3257, 3930, 1424, 2856, 1587]|[0.00279246383685
64403, 0.002621661590321394, 0.0025252443984009093, 0.0025124291093043824, 0.0019972
97335766142, 0.001982440967930216, 0.0018045163039332257, 0.001682679463459467, 0.00
16734414476656321, 0.0016137434121452402]|
+-----+-----------------------------------------------------------+----------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----------------------------------------+

LDA Evaluation Metrics:
Log Likelihood: -557805.3470146725
Perplexity: 8.582731841134969
```



Claudine - Neural Network and LDA

I used accuracy, precision, recall, and F1 score to evaluate the Multilayer Perceptron, which hit over 71% accuracy. The confusion matrix showed good performance in classifying authors. For LDA, I used log likelihood and perplexity to assess topic quality and reviewed the top words per topic. TF-IDF weights helped identify the most influential words in classification and topic grouping. Next, I'd try tuning MLP hyperparameters and adding features like sentence length or punctuation to see if they improve performance

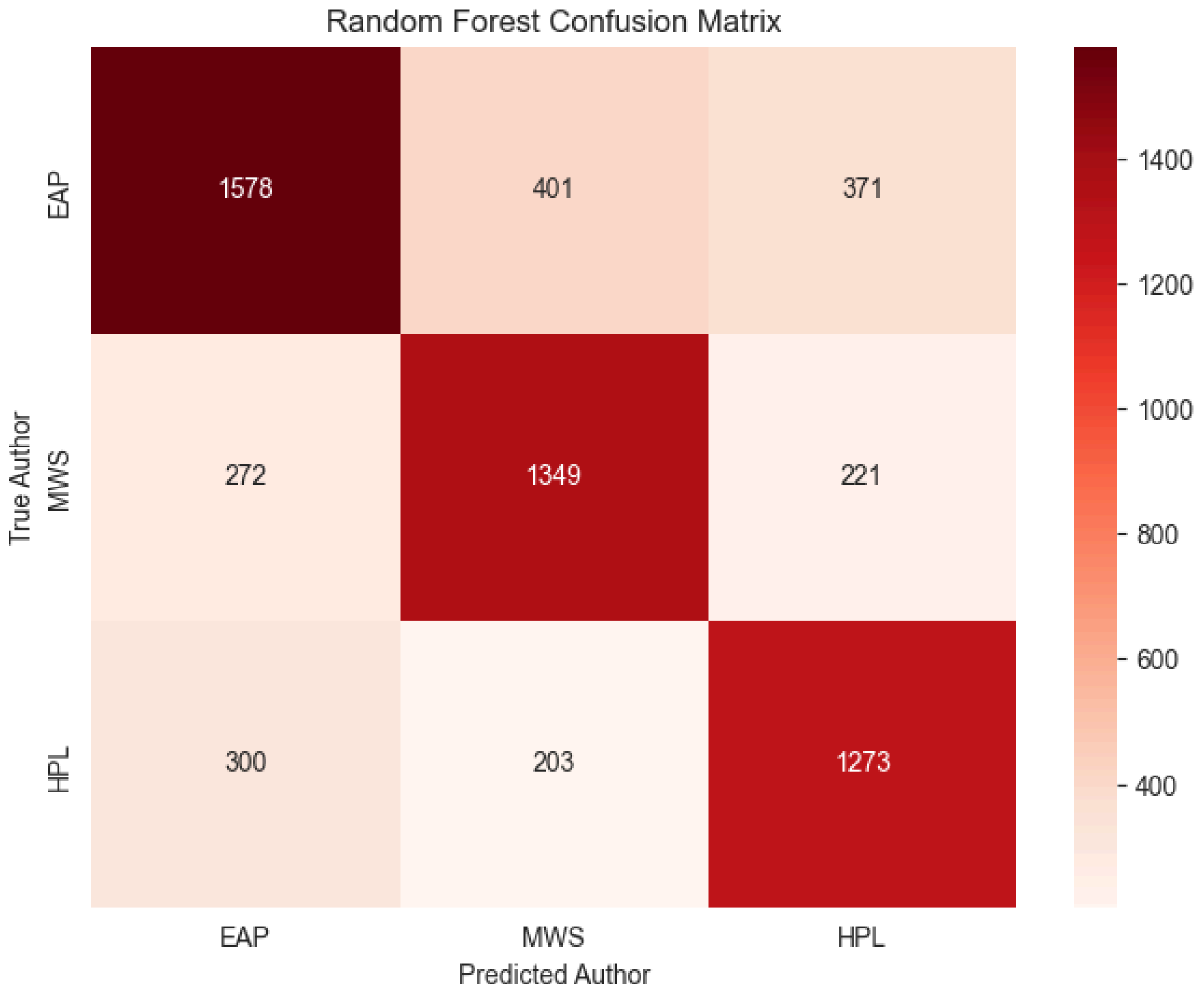
In [19]:

```
# Radhika
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from sklearn.metrics import confusion_matrix

evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="pred

# ------------------------- Random Forest ----------------------
accuracy = evaluator.evaluate(rf_predictions, {evaluator.metricName: "accuracy"})
precision = evaluator.evaluate(rf_predictions, {evaluator.metricName: "weightedPrec
recall = evaluator.evaluate(rf_predictions, {evaluator.metricName: "weightedRecall"
f1 = evaluator.evaluate(rf_predictions, {evaluator.metricName: "f1"})

print("Random Forest Algorithm Performance Metrics:")
print(f"Accuracy:  {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1 Score:  {f1:.4f}")

# Get confusion matrix
df_rf_predictions = rf_predictions.select('label', 'prediction').toPandas()
conf_matx = confusion_matrix(df_rf_predictions['label'], df_rf_predictions['predict

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matx, annot=True, fmt='d', cmap='Blues', xticklabels=label_mapping
plt.title("Random Forest Confusion Matrix")
plt.xlabel("Predicted Author")
plt.ylabel("True Author")
plt.show()

# The confusion matrix for the random forest is very much skewed towards author Edg
# This indicates that this algorithm is not well-suited to this data, and I think t

# ------------------------- Naive Bayes ----------------------
accuracy = evaluator.evaluate(nb_predictions, {evaluator.metricName: "accuracy"})
precision = evaluator.evaluate(nb_predictions, {evaluator.metricName: "weightedPrec
recall = evaluator.evaluate(nb_predictions, {evaluator.metricName: "weightedRecall"
f1 = evaluator.evaluate(nb_predictions, {evaluator.metricName: "f1"})

print("Naive Bayes Algorithm Performance Metrics:")
print(f"Accuracy:  {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1 Score:  {f1:.4f}")

# Get confusion matrix
```

```
df_nb_predictions = nb_predictions.select('label', 'prediction').toPandas()
conf_matx = confusion_matrix(df_nb_predictions['label'], df_nb_predictions['predict

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matx, annot=True, fmt='d', cmap='Reds', xticklabels=label_mapping,
plt.title("Random Forest Confusion Matrix")
plt.xlabel("Predicted Author")
plt.ylabel("True Author")
plt.show()

# The confusion matrix for naive bayes, on the other hand, is very well balanced. A
# quite high, indicating that this algorithm is very well-suited to this data. I th
```

```
Random Forest Algorithm Performance Metrics:
Accuracy:  0.4353
Precision: 0.5496
Recall:    0.4353
F1 Score:  0.3270
```


```
Naive Bayes Algorithm Performance Metrics:
Accuracy:  0.7038
Precision: 0.7053
Recall:    0.7038
F1 Score:  0.7037
```



Radhika - Random Forest and Naive-Bayes

1. Random Forest: The confusion matrix for the random forest is very much skewed towards author Edgar Allen Poe. Also, the accuracy for this algorithm is <50%. This indicates that this algorithm is not well-suited to this data, and I think this is because it is a classification problem.

2. Naive-Bayes: The confusion matrix for naive bayes, on the other hand, is very well balanced. Also, the accuracy for this algorithm is naturally quite high, indicating that this algorithm is very well-suited to this data. I think this is also because it is a classification problem.