a4

Graded

Group

Mihret Kemal
Tagore Kosireddy
Michael Ngala
...and 1 more
View or edit group

Total Points

227.5 / 244 pts

Autograder Score

134.0 / 144.0

Failed Tests

q2d (2/8)
q2e (1/5)

Passed Tests

Public Tests
q0 (2/2)
q1a (12/12)
q1b (10/10)
q1c (6/6)
q1f (8/8)
q1g (8/8)
q1i (20/20)
q1j (8/8)
q2a (9/9)
q2b (4/4)
q2c (13/13)
q3a (4/4)
q3b (9/9)
q3c (9/9)
q3d (9/9)

Question 2

Question 1 **52** / 58 pts

2.1 **Q1d** **5.5** / 6 pts

✔ **− 0.5 pts** unnecessary markup, color (too bright or too dim), or addition to the plot

2.2 **Q1e** **6.5** / 8 pts

✔ **− 0.5 pts** missing/incorrect titles

✔ **− 0.5 pts** unnecessary markup, color (too bright/ too dim), or addition to the plot

✔ **− 0.5 pts** Plot is too large, doesn't fit when printing

2.3 **Q1f** **8** / 8 pts

✔ **− 0 pts** Correct

2.4 **Q1g** **8** / 8 pts

✔ **− 0 pts** Correct

2.5 **Q1h** **3** / 6 pts

✔ **− 2 pts** error in response to problem

problem is overfitting.

✔ **− 1 pt** error in how to solve it

Possible solutions: limit features, get more samples, simplify model

2.6 **Q1i** **12** / 12 pts

✔ **− 0 pts** Correct

2.7 **Q1j** **9** / 10 pts

✔ **− 0.5 pts** missing/incorrect y-axis labels

✔ **− 0.5 pts** Feature labels incorrect / plot shape incorrect

Question 3

Question 2 **7.5** / 8 pts

3.1 **Q2f** **7.5** / 8 pts

✔ **− 0.5 pts** extra color, information, markings not necessary for plot

✔ **− 0 pts** different x and y-scales

Question 4

Question 3 **24** / 24 pts

4.1 Q3b **8** / 8 pts

✔ **− 0 pts** Correct

4.2 Q3c **8** / 8 pts

✔ **− 0 pts** Correct

4.3 Q3d **8** / 8 pts

✔ **− 0 pts** Correct

Question 5

General **10** / 10 pts

✔ **− 0 pts** Correct

Autograder Results

Autograder Output

```
  _________        __        __
 /  _____  \    __|  |__  __|  |__
|  /     \  |  |__    __||__    __|   ______    _  _____
| |       | |     |  |      |  |    |  __  |  | |/ ____|
| |       | |     |  |      |  |    | |__| |  |   /
| |       | |     |  |      |  |    |  ____|  |  |
|  \_____/  |     |  |_     |  |_   | |____   |  |
 \_________/       \___|     \___|   \______| |__|
                                            v5.2.3
```

------------------------------- GRADING SUMMARY -------------------------------

Score for q1c (4.000) differs from logged score (6.000)
Score for q1f (6.000) differs from logged score (8.000)
Score for q1g (6.000) differs from logged score (8.000)
Score for q1j (4.000) differs from logged score (8.000)
Score for q2b (1.000) differs from logged score (4.000)
Score for q2d (2.000) differs from logged score (8.000)
Score for q3a (1.000) differs from logged score (4.000)

Total Score: 134.000 / 144.000 (93.056%)

	name	score	max_score
0	Public Tests	NaN	NaN
1	q0	2.0	2.0
2	q1a	12.0	12.0
3	q1b	10.0	10.0
4	q1c	6.0	6.0
5	q1f	8.0	8.0
6	q1g	8.0	8.0
7	q1i	20.0	20.0
8	q1j	8.0	8.0
9	q2a	9.0	9.0
10	q2b	4.0	4.0
11	q2c	13.0	13.0
12	q2d	2.0	8.0
13	q2e	1.0	5.0
14	q3a	4.0	4.0
15	q3b	9.0	9.0
16	q3c	9.0	9.0
17	q3d	9.0	9.0

Public Tests

q0 results: All test cases passed!

q1a results: All test cases passed!

q1b results: All test cases passed!

q1c results: All test cases passed!

q1f results: All test cases passed!

q1g results: All test cases passed!

q1i results: All test cases passed!

q1j results: All test cases passed!

q2a results: All test cases passed!

q2b results: All test cases passed!

q2c results: All test cases passed!

q2d results: All test cases passed!

q2e results: All test cases passed!

q3a results: All test cases passed!

q3b results: All test cases passed!

q3c results: All test cases passed!

q3d results: All test cases passed!

q0 (2/2)

q0 results: All test cases passed!

q1a (12/12)

q1a results: All test cases passed!

q1b (10/10)

q1b results: All test cases passed!

q1c (6/6)

q1c results: All test cases passed!

q1f (8/8)

q1f results: All test cases passed!

q1g (8/8)

q1g results: All test cases passed!

q1i (20/20)

q1i results: All test cases passed!

q1j (8/8)

q1j results: All test cases passed!

q2a (9/9)

q2a results: All test cases passed!

q2b (4/4)

q2b results: All test cases passed!

q2c (13/13)

q2c results: All test cases passed!

q2d (2/8)

```
q2d results:
    q2d - 1 result:
        Test case passed

    q2d - 2 result:
        Test case failed
        Trying:
            all(clusterStats.Num == [204, 119, 46, 51])
        Expecting:
            True
        **********************************************************************
        Line 1, in q2d 1
        Failed example:
            all(clusterStats.Num == [204, 119, 46, 51])
        Expected:
            True
        Got:
            False

    q2d - 3 result:
        Test case failed
        Trying:
            all(np.isclose(clusterStats.TRB, [0.138268, 0.217647, 0.402464, 0.496863]))
        Expecting:
            True
        **********************************************************************
        Line 1, in q2d 2
        Failed example:
            all(np.isclose(clusterStats.TRB, [0.138268, 0.217647, 0.402464, 0.496863]))
        Expected:
            True
        Got:
            False

    q2d - 4 result:
        Test case failed
        Trying:
            all(np.isclose(clusterStats.STL, [0.200758, 0.411765, 0.52668, 0.357398]))
        Expecting:
            True
        **********************************************************************
        Line 1, in q2d 3
        Failed example:
            all(np.isclose(clusterStats.STL, [0.200758, 0.411765, 0.52668, 0.357398]))
        Expected:
            True
        Got:
            False
```

q2e (1/5)

q2e results:
    q2e - 1 result:
        Test case passed

    q2e - 2 result:
        Test case failed
        Trying:
            all(np.isclose(clusterStatsOrig.ORB, [0.652451, 0.697479, 1.252174, 2.376471]))
        Expecting:
            True
        ****
        Line 1, in q2e 1
        Failed example:
            all(np.isclose(clusterStatsOrig.ORB, [0.652451, 0.697479, 1.252174, 2.376471]))
        Expected:
            True
        Got:
            False

    q2e - 3 result:
        Test case failed
        Trying:
            all(np.isclose(clusterStatsOrig.STL, [0.441667, 0.905882, 1.158696, 0.786275]))
        Expecting:
            True
        ****
        Line 1, in q2e 2
        Failed example:
            all(np.isclose(clusterStatsOrig.STL, [0.441667, 0.905882, 1.158696, 0.786275]))
        Expected:
            True
        Got:
            False

q3a (4/4)

q3a results: All test cases passed!

q3b (9/9)

q3b results: All test cases passed!

q3c (9/9)

q3c results: All test cases passed!

q3d (9/9)

q3d results: All test cases passed!

Submitted Files

▾ a4 (1) (1).ipynb Download

a4 - Python

This assignment will cover topics of text mining and clustering

Make sure that you keep this notebook named as "a4.ipynb"

Any other packages or tools, outside those listed in the assignments or Canvas, should be cleared by Dr. Brown before use in your submission.

Q0 - Setup

The following code looks to see whether your notebook is run on Gradescope (GS), Colab (COLAB), or the linux Python environment you were asked to setup.

In [147]:

```
import re
import os
import platform
import sys

# flag if notebook is running on Gradescope
if re.search(r'am', platform.uname().release):
    GS = True
else:
    GS = False

# flag if notebook is running on Colaboratory
try:
  import google.colab
  COLAB = True
except:
  COLAB = False

# flag if running on Linux lab machines.
cname = platform.uname().node
if re.search(r'(guardian|colossus|c28|coc-15954-m)', cname):
    LLM = True
else:
    LLM = False

print("System: GS - %s, COLAB - %s, LLM - %s" % (GS, COLAB, LLM))
```

System: GS - False, COLAB - False, LLM - True

Notebook Setup

It is good practice to list all imports needed at the top of the notebook. You can import modules in later cells as needed, but listing them at the top clearly shows all which are needed to be available / installed.

If you are doing development on Colab, the otter-grader package is not available, so you will need to install it with pip (uncomment the cell directly below).

In [148]:

```
# Only uncomment if you developing on Colab
# if COLAB == True:
#     print("Installing otter:")
#     !pip install otter-grader==4.2.0
```

In [149]:

```
# Import standard DS packages
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import math
import scipy
import statistics
import textwrap
%matplotlib inline


from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn import tree       # decision tree classifier
from sklearn import neighbors   # knn classifier
from sklearn import naive_bayes # naive bayes classifier
from sklearn import svm        # svm classifier
from sklearn import ensemble    # ensemble classifiers
from sklearn import metrics     # performance evaluation metrics
from sklearn import model_selection
from sklearn import preprocessing
from sklearn.decomposition import PCA
from sklearn.datasets import load_files
```

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTransformer

from sklearn import preprocessing
from sklearn.cluster import AgglomerativeClustering
from scipy.spatial.distance import pdist
from scipy.spatial.distance import squareform
from sklearn import cluster
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster import hierarchy

# Package for Autograder
import otter
grader = otter.Notebook()
```

In [150]:

```
grader.check("q0")
```

Out [150]:

q0 results: All test cases passed!

Q1 - Text Classification

You will look to predict whether scenes in Shakespeare's plays come from the comedies or histories. Shakespeare's comedies include plays such as: The Taming of the Shrew, The Merchant of Venice, Much Ado About Nothing, and more. The histories include: Richard II, Richard III, Henry IV part 1, Henry IV part 2, Henry V, Henry VI (part 1-3).

The plays were downloaded from the [Shakespeare Corpus](). Note, the original plays were downloaded from [Project Gutenberg]().

Note, the plays have already had significant preprocessing. The plays have been scrubbed by: removing digits, making the file all lowercase, and removing punctuation, excluding hyphens and word-internal apostrophes. Also, the character names and stage directions have been removed manually. An example of the text would be like this:

Before scrubbing:

```
ADAM. Yonder comes my master, your brother.
ORLANDO. Go apart, Adam, and thou shalt hear how he will shake me
up. [ADAM retires]
OLIVER. Now, sir! what make you here?
```

After scrubbing:

```
yonder comes my master your brother
go apart adam and thou shalt hear how he will shake me
up
now sir what make you here
```

The text files are split into negative - comedies and positive - histories.

Q1(a) - Load the Data

Load the plays into a list `textdata` and a np.ndarray `yvalues`. I highly suggest using `scikit-learn`'s `load_files` function, with the `random_state` set to 42.

In [151]:

```
# Load the plays data

plays = load_files("data/shakespeare",random_state=42)

textdata= plays.data
yvalues=plays.target

print("Samples per class: {}".format(np.bincount(yvalues)))

plays.filenames[0:10]
```

```
Samples per class: [119 208]
```

Out [151]:

```
array(['data/shakespeare/histories/henryVIpartiiActIIIScenei_noCNnoSD.txt',
       'data/shakespeare/comedies/twelfthNightActIScenei_noCNnoSD.txt',
       'data/shakespeare/histories/henryVIpartiiActIVSceneviii_noCNnoSD.txt',
       'data/shakespeare/comedies/asYouLikeItActIIScenev_noCNnoSD.txt',
       'data/shakespeare/comedies/tempestActIIISceneii_noCNnoSD.txt',
       'data/shakespeare/histories/henryVIpartiActIVSceneii_noCNnoSD.txt',
       'data/shakespeare/histories/henryVIpartiiiActIVSceneviii_noCNnoSD.txt',
       'data/shakespeare/histories/henryVIIIActVScenei_noCNnoSD.txt',
       'data/shakespeare/histories/henryVIpartiActIIIScenei_noCNnoSD.txt',
       'data/shakespeare/comedies/twelfthNightActIIIScenei_noCNnoSD.txt'],
      dtype='<U72')
```

In [152]:

```
grader.check("q1a")
```

Out [152]:

```
q1a results: All test cases passed!
```

Q1(b) - Prepare the Data

Split the data into `text_trainval`, `text_test` and `y_trainval`, `y_test` variables. Use 20% of the data in the test set with a `random_state` of 42 and make sure to stratify the split (the data is imbalanced).

In [153]:

```
# Split the data
text_trainval, text_test, y_trainval, y_test = train_test_split(textdata, yvalues,
                                test_size=0.2, random_state=42, stratify=yvalues)
```

In [154]:

```
grader.check("q1b")
```

Out [154]:

q1b results: All test cases passed!

Q1(c) - Explore the Data

Create a document-term count matrix for the "trainval" data using the default tokenizer, removing the standard English stopwords and store this in `dtm_trainval`.

Store the names of the terms in the dtm matrix in the variable `vocab`.

In [155]:

```
# Create document-term count matrix for the "trainval" text data
vectorizer = CountVectorizer(stop_words='english')

# fit and transform the trainval data to create the document-term count matrix
dtm_trainval = vectorizer.fit_transform(text_trainval)

# store the names of the terms in the dtm matrix in the variable vocab
vocab = vectorizer.get_feature_names_out()
```

In [156]:

```
grader.check("q1c")
```

Out [156]:

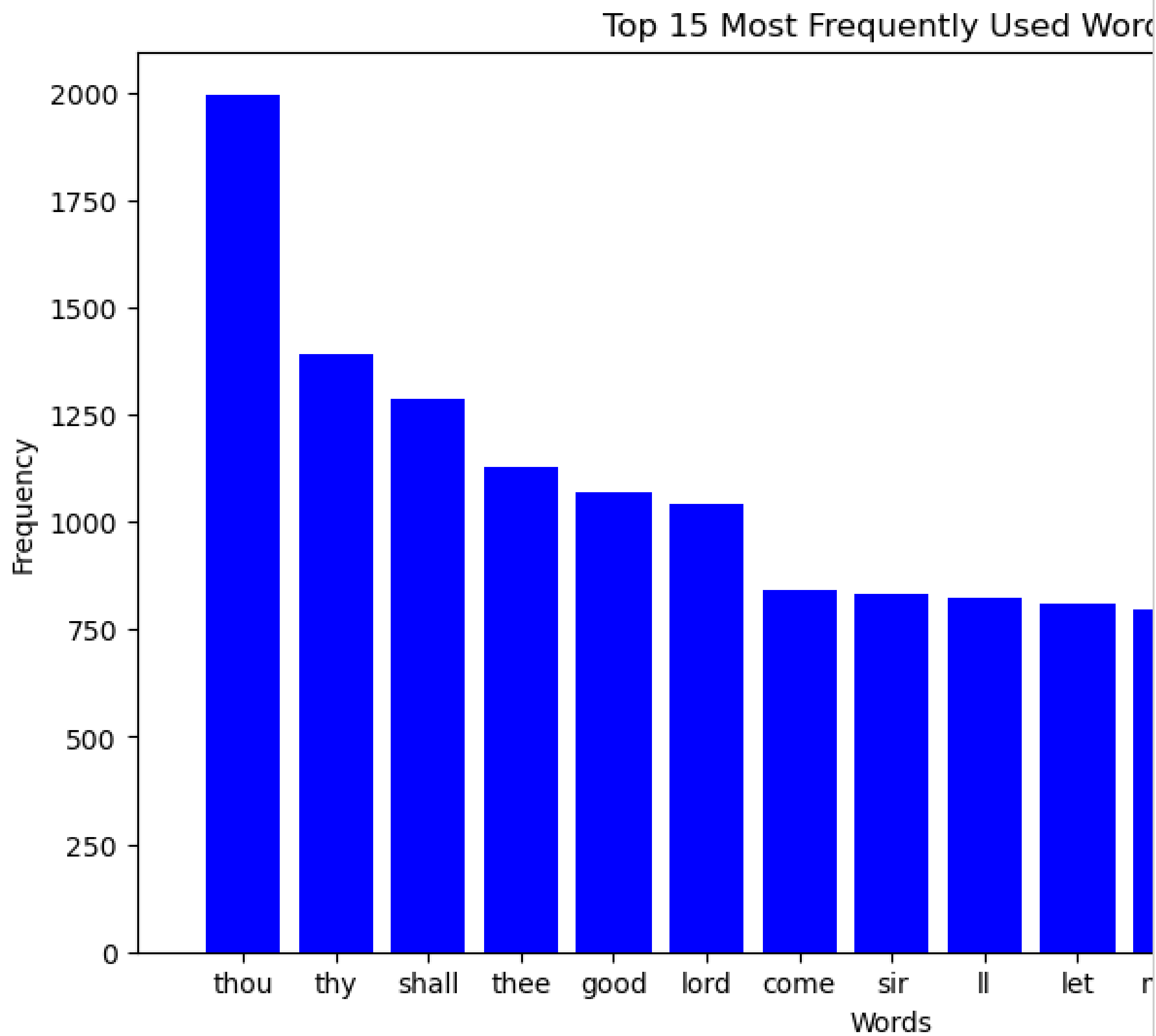
q1c results: All test cases passed!

Q1(d) - Explore the Data

Create a plot showing the top 15 most frequently used words in the trainval text data.

In [157]:

```
# Create a plot of the top 15 most frequently used words

# sum the counts of each term across all documents
term_frequencies = np.asarray(dtm_trainval.sum(axis=0))[0]

# sort the terms based on their frequency and extract the top 15 most frequent
terms
top_indices = term_frequencies.argsort()[::-1][:15]
top_terms = vocab[top_indices]
top_frequencies = term_frequencies[top_indices]

# convert top_terms to a list of strings
top_terms = [str(term) for term in top_terms]

# plot the top 15 most frequent terms using a loop
plt.figure(figsize=(10, 6))
for i in range(15):
    plt.bar(top_terms[i], top_frequencies[i], color='blue')

plt.xlabel('Words')
plt.ylabel('Frequency')
plt.title('Top 15 Most Frequently Used Words')
plt.show()
```



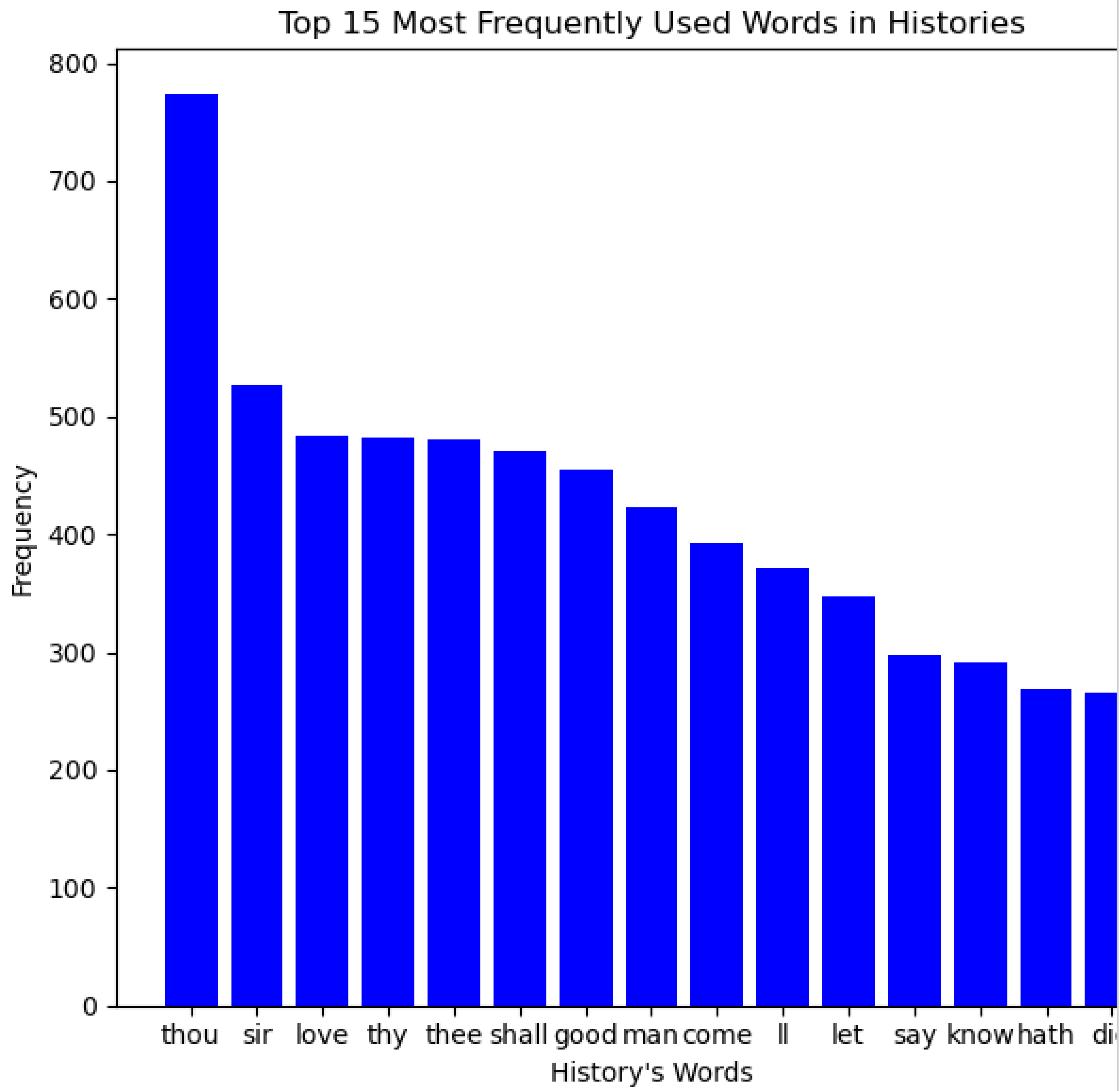
Q1(e) - Explore the Data

For the trainval text data, plot the top 15 most frequently used words in the histories and the comedies. Put these two bar plots side-by-side to compare the results.

In [158]:

```
# Create a plot of the top 15 most frequently used words in the
#  Comedies and Histories.

# catagorize trainval data as (histories or comedies)
histories_indices = np.where(y_trainval == 0)[0]
comedies_indices = np.where(y_trainval == 1)[0]

# catagorize text data based on indices
histories_text = [text_trainval[i] for i in histories_indices]
comedies_text = [text_trainval[i] for i in comedies_indices]

# Create document-term count matrices for histories and comedies
```

```
histories_dtm = vectorizer.transform(histories_text)
comedies_dtm = vectorizer.transform(comedies_text)

# Get vocabulary
vocab = vectorizer.get_feature_names_out()

# Get term frequencies for histories and comedies
histories_term_frequencies = np.asarray(histories_dtm.sum(axis=0))[0]
comedies_term_frequencies = np.asarray(comedies_dtm.sum(axis=0))[0]

# Sort the terms based on their frequency for histories and comedies
histories_top_indices = histories_term_frequencies.argsort()[::-1][:15]
comedies_top_indices = comedies_term_frequencies.argsort()[::-1][:15]

# Get top terms and their frequencies for histories and comedies
histories_top_terms = vocab[histories_top_indices]
comedies_top_terms = vocab[comedies_top_indices]
histories_top_frequencies = histories_term_frequencies[histories_top_indices]
comedies_top_frequencies = comedies_term_frequencies[comedies_top_indices]

# Convert top terms to a list of strings
histories_top_terms = [str(term) for term in histories_top_terms]
comedies_top_terms = [str(term) for term in comedies_top_terms]

# Plot the top 15 most frequent terms for histories and comedies side-by-side
plt.figure(figsize=(13, 6))

# Plot for histories
plt.subplot(1, 2, 1)
for i in range(15):
    plt.bar(histories_top_terms[i], histories_top_frequencies[i], color='blue')
plt.xlabel("History's Words")
plt.ylabel('Frequency')
plt.title('Top 15 Most Frequently Used Words in Histories')

# Plot for comedies
plt.subplot(1, 2, 2)
for i in range(15):
    plt.bar(comedies_top_terms[i], comedies_top_frequencies[i], color='red')
plt.xlabel("Comedy's Words")
plt.ylabel('Frequency')
plt.title('Top 15 Most Frequently Used Words in Comedies')

plt.tight_layout()
plt.show()
```



Q1(f) - Bernoulli Naive Bayes

Let's know explore using Bernoulli Naive Bayes as a classifier, `bern_nb`, to predict the type of play.

We will use the split of the data into trainval / test found above to train the model and then evaluate it's performance.

Create the training data, `X_trainval` to be binary with features using the default tokenizer, stop words removed, appear in at least 5 documents and is limited to the top 5000 features.

Calculate the training accuracy `train_acc_bern` and testing accuracy `test_acc_bern` for the model.

In [159]:

```
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score
# Run Bernoulli Naive Bayes model

# Initialize CountVectorizer with binary=True, stop words removed, min_df=5,
max_features=5000
vectorizer = CountVectorizer(binary=True,stop_words='english', min_df=5,
max_features=5000)

# Fit and transform the training data
X_trainval = vectorizer.fit_transform(text_trainval)
X_test = vectorizer.transform(text_test)

bern_nb = BernoulliNB()

# Train the classifier
bern_nb.fit(X_trainval, y_trainval)

# Make predictions on training and testing data
y_trainval_pred = bern_nb.predict(X_trainval)
y_test_pred = bern_nb.predict(X_test)

# Calculate training and testing accuracy
train_acc_bern = accuracy_score(y_trainval, y_trainval_pred)
test_acc_bern = accuracy_score(y_test, y_test_pred)

print(f"Training Accuracy: {train_acc_bern:}")
print(f"Testing Accuracy: {test_acc_bern:}")
```

```
Training Accuracy: 0.9501915708812261
Testing Accuracy: 0.8636363636363636
```

In [160]:

```
grader.check("q1f")
```

Out [160]:

```
q1f results: All test cases passed!
```

Q1(g) - Multinomial Naive Bayes

Let's know explore using multinomial Naive Bayes as a classifier, `mult_nb`, to predict the type of play.

We will use the split of the data into trainval / test found above to train the model and then evaluate it's performance.

Create the training data, `X_trainval` with features using the default tokenizer, stop words removed, appear in at least 5 documents and is limited to the top 5000 features.

Calculate the training accuracy `train_acc_mult` and testing accuracy `test_acc_mult` for the model.

In [161]:

```
from sklearn.naive_bayes import MultinomialNB
# Run Multinomial Naive Bayes model
# Initialize CountVectorizer with default tokenizer stop words removed, min_df=5,
max_features=5000
vectorizer = CountVectorizer(binary=False,stop_words='english', min_df=5,
max_features=5000)

# Fit and transform the training data
X_trainval = vectorizer.fit_transform(text_trainval)
X_test = vectorizer.transform(text_test)

mult_nb = MultinomialNB()

# Train the classifier
mult_nb.fit(X_trainval, y_trainval)

# Make predictions on training and testing data
y_trainval_pred = mult_nb.predict(X_trainval)
y_test_pred = mult_nb.predict(X_test)

# Calculate training and testing accuracy
train_acc_mult = accuracy_score(y_trainval, y_trainval_pred)
test_acc_mult = accuracy_score(y_test, y_test_pred)

print(f"Training Accuracy: {train_acc_mult:}")
print(f"Testing Accuracy: {test_acc_mult:}")
```

```
Training Accuracy: 1.0
Testing Accuracy: 0.9696969696969697
```

In [162]:

```
grader.check("q1g")
```

Out [162]:

q1g results: All test cases passed!

Q1(h) - Naive Bayes Models

Looking at the results of the two models above. Answer the following questions.

Which of the two models is preferred? Why? (10 words or less)

Ans: Multinomial model is prefered for it's high testing accuracy and frequency consideration.

What is a problem for both models? How might you solve it? (12 words or less)

Ans: Problem in classifiying infrequent but important words(Vocabularies), solved using TF-IDF weighting.

Q1(i) - Other Models

Let's now look to explore using other models.

You will set up a pipeline, `pipe`, that will use a Random Forest model with 100 trees and a `random_state` = 42.

In the pipeline (`param_grid`), you will consider using both a document term count matrix as well as a TF-IDF matrix. In either case, limit the matrix to words that appear in at least 5 documents and remove English stop words. Consider features of unigrams, unigrams + bigrams, and bigrams. Examine a maximum feature limit of either 2500 or 5000.

Optimize your choice of hyperparameters using GridSearchCV, `grid`, with stratified 5-fold cross-validation (random_state = 42), select the parameters using AUC. (See how to set up the scorer below)

Note, do not run the jobs in parallel, you may exceed the memory resources of the autograder on Gradescope.

In [163]:

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import make_scorer, roc_auc_score
```

```
# Set up the pipeline with CountVectorizer and RandomForestClassifier
pipe = Pipeline([
    ('vec', CountVectorizer()),  # CountVectorizer
    ('rf', RandomForestClassifier(n_estimators = 100, random_state = 42))
])

# Define the parameter grid for GridSearchCV
param_grid = {
    'vec': [CountVectorizer(stop_words = 'english', min_df = 5),
TfidfVectorizer(stop_words = 'english', min_df = 5)],
    'vec__ngram_range': [(1, 1), (1, 2), (2, 2)],
    'vec__max_features': [2500, 5000],
}

# Create StratifiedKFold for cross validation
cvStrat = StratifiedKFold(n_splits = 5, random_state = 42, shuffle = True)

# Create scorer for auc
score_fn = make_scorer(roc_auc_score, needs_threshold=False)

# Create GridSearchCV object with cross validation and auc scoring
grid = GridSearchCV(pipe, param_grid, cv = cvStrat, scoring = score_fn)

# Fit the GridSearchCV to the data
grid.fit(text_trainval, y_trainval)

# Print the best cross validation score and best parameters
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
print("Best parameters:\n", grid.best_params_)
```

```
Best cross-validation score: 0.89
Best parameters:
 {'vec': TfidfVectorizer(max_features=2500, min_df=5, stop_words='english'), 'vec__max_fea
```

In [164]:

```
grader.check("q1i")
```

Out [164]:

q1i results: All test cases passed!

Q1(j) - Explore the Results

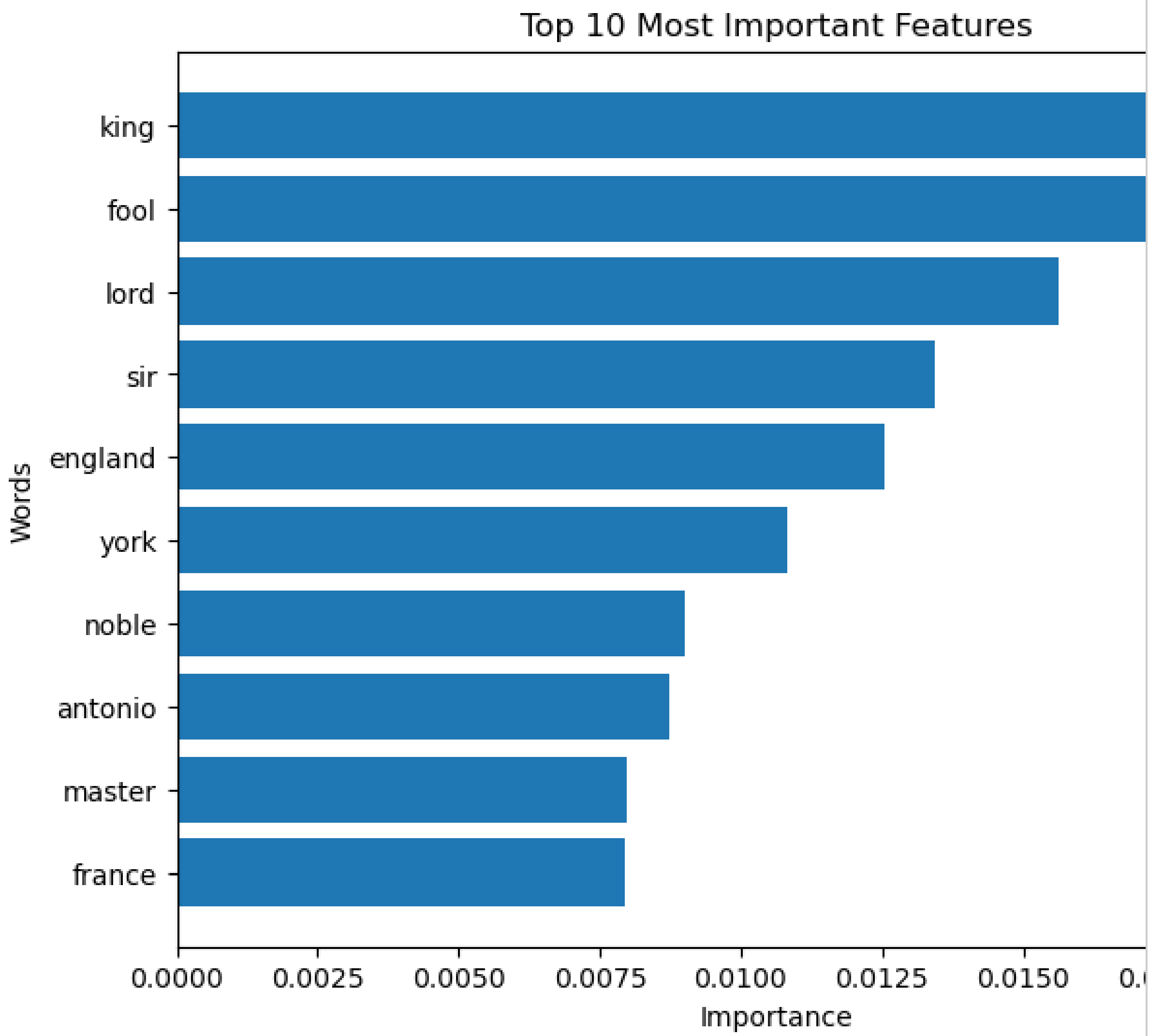
Calculate the AUC on the test text, `auc_test`.

Gather the importances of the features in the best model in `importance`. Feature Importance Example

Create a bar plot with the top 10 features sorted by importance.

In [165]:

```
# Calculate performance on test data auc_test
best_model = grid.best_estimator_
y_pred = best_model.predict(text_test)
auc_test = roc_auc_score(y_test, y_pred)

# Extract feature importances and sort
importance = best_model.named_steps['rf'].feature_importances_
sorted_ind = np.argsort(importance)[::-1]
top_imp = importance[sorted_ind]
top_features = [best_model.named_steps['vec'].get_feature_names_out()[i] for i in
sorted_ind]

# Create plot of top 10 features sorted by importance
plt.figure(figsize=(8, 6))
plt.barh(top_features[1:11], top_imp[1:11])
plt.gca().invert_yaxis()
plt.xlabel('Importance')
plt.ylabel('Words')
plt.title('Top 10 Most Important Features')
plt.show()

print("AUC on test data:", auc_test)
```



AUC on test data: 0.875

In [166]:

```
grader.check("q1j")
```

Out [166]: q1j results: All test cases passed!

Q2

Consider methods to cluster NBA players based on their statistics.

Q2(a)

Load in data for NBA players from the 2018-2019 season.

Filter the players to only consider those who have played in more than 20 games.

Ignore the first 7 columns as well as ignore columns of statistics with percentages (FG%, 3P%, 2P%, eFG%, FT%).

In [167]:

```
# Load in data and filter out requested rows and columns.
nba = pd.read_csv('data/nba18-19.csv')
nba = nba[nba["G"] > 20]
# nba = nba.drop(nba.columns[:7], axis=1)
columns_to_drop = ['FG%', '3P%', '2P%', 'eFG%', 'FT%']
nba = nba.drop(columns = columns_to_drop)

nba.head()
```

Out [167]:

```
   Rk                 Player Pos  Age   Tm   G  GS    MP   FG   FGA  \
0   1      Álex Abrines\abrinal01  SG   25  OKC  31   2  19.0  1.8   5.1
2   3      Jaylen Adams\adamsja01  PG   22  ATL  34   1  12.6  1.1   3.2
3   4      Steven Adams\adamsst01   C   25  OKC  80  80  33.4  6.0  10.1
4   5       Bam Adebayo\adebaba01   C   21  MIA  82  28  23.3  3.4   5.9
7   8  LaMarcus Aldridge\aldrila01   C   33  SAS  81  81  33.2  8.4  16.3

   ...  FTA  ORB  DRB  TRB  AST  STL  BLK  TOV   PF   PTS
0  ...  0.4  0.2  1.4  1.5  0.6  0.5  0.2  0.5  1.7   5.3
2  ...  0.3  0.3  1.4  1.8  1.9  0.4  0.1  0.8  1.3   3.2
3  ...  3.7  4.9  4.6  9.5  1.6  1.5  1.0  1.7  2.6  13.9
4  ...  2.8  2.0  5.3  7.3  2.2  0.9  0.8  1.5  2.5   8.9
7  ...  5.1  3.1  6.1  9.2  2.4  0.5  1.3  1.8  2.2  21.3

[5 rows x 25 columns]
```

In [168]:

```
nba.shape
```

Out [168]:

```
(420, 25)
```

In [169]:

```
grader.check("q2a")
```

Out [169]:

```
q2a results: All test cases passed!
```

Q2(b)

The features have different ranges, therefore we should scale the data before considering the clustering analysis. Scale the data using min-max normalization with range of [0, 1].

In [170]:

```
# Scale the data
scaler = MinMaxScaler()
nbaScaled = scaler.fit_transform(nba.iloc[:,7:]) #ignore the first 7 columns
```

In [171]:

```
grader.check("q2b")
```

Out [171]:

q2b results: All test cases passed!

Q2(c)

Run Kmeans clustering on the data with k=2, . . . , 12. Set the `random_state` in the Kmeans method to 42 and `n_init` to 10. For each value of k, keep track of the within-cluster variation (this quantity is referred to as different terms such as "inertia" and total "within-cluster sum-of-squares"), the Calinski-Harabasz score, and the Davies-Bouldin index on the resulting clusters.

In [172]:

```
# Run Kmeans

from sklearn.cluster import KMeans
from sklearn.metrics import calinski_harabasz_score
from sklearn.metrics import davies_bouldin_score
sse = []
dbscore = []
chscore = []

for i in range(2,11):
    kmeans = KMeans(n_clusters = i, init='k-means++', n_init=10,random_state=42)
    kmeans.fit(nbaScaled)
    sse.append(kmeans.inertia_)
    chscore.append(calinski_harabasz_score(nbaScaled, kmeans.labels_))
    dbscore.append(davies_bouldin_score(nbaScaled, kmeans.labels_))
```

In [175]:

```
grader.check("q2c")
```

Out [175]:

q2c results: All test cases passed!

Q2(d)

Assuming the best number of clusters is 4 (depending on which measure we use different number of clusters is preferred with this data).

Run Kmeans again with this value for k (use `n_init` = 10 and `random_state` = 10).

Create a DataFrame `clusterStats` with the mean statistics (centers) of each group.

The DataFrame should have rows for each cluster group 0, 1, 2, 3 and columns for the mean statistics.

Add a column `Num` reporting the number of samples in each group.

In [176]:

```
# Create a Data Frame for the mean statistics of each group
# runing k means with this value for (use n_init = 10 and random_state = 10).
kmeans = KMeans(n_clusters=4, init='k-means++', n_init=10, random_state=10)
kmeans.fit(nbaScaled)
#finding the cluster centers for each group
cluster_centers = kmeans.cluster_centers_
#create a dataframe for each group with the columns for mean statistics
clusterStats = pd.DataFrame(cluster_centers, columns=nba.columns[7:]) #ignoring the first 7 cols
clusterStats['Num'] = pd.Series(kmeans.labels_).value_counts().sort_index()

clusterStats[['Num', 'MP', 'FG', '3P', 'FT']]
```

Out [176]:

	Num	MP	FG	3P	FT
0	201	0.339842	0.171313	0.117842	0.077704
1	46	0.893997	0.735029	0.397272	0.447333
2	125	0.701462	0.375774	0.296471	0.167505
3	48	0.682905	0.466785	0.093546	0.231959

In [177]:

```
grader.check("q2d")
```

Out [177]:

q2d results: All test cases passed!

Q2(e)

Report the same statistics as in (e), but using the original data scaling (reverse the scaling back to the original data range).

Store results in `clusterStatsOrig`; this DataFrame should not have the "Num" column.

In [178]:

```
# Create a Data Frame for the mean statistics of each group (using the
#   original data scaling)
#reversing the scaling back to the original data
original_data= scaler.inverse_transform(kmeans.cluster_centers_)
#create a dataframe for each group with the columns for mean statistics
clusterStatsOrig = pd.DataFrame(original_data,
columns=nba.columns[7:])#ignoring the first 7 cols

clusterStatsOrig[['MP', 'FG', '3P', 'FT']]
```

Out [178]:

	MP	FG	3P	FT
0	15.048756	2.015920	0.600995	0.753731
1	33.391304	7.991304	2.026087	4.339130
2	27.018400	4.183200	1.512000	1.624800
3	26.404167	5.147917	0.477083	2.250000

In [179]:

```
grader.check("q2e")
```

Out [179]:

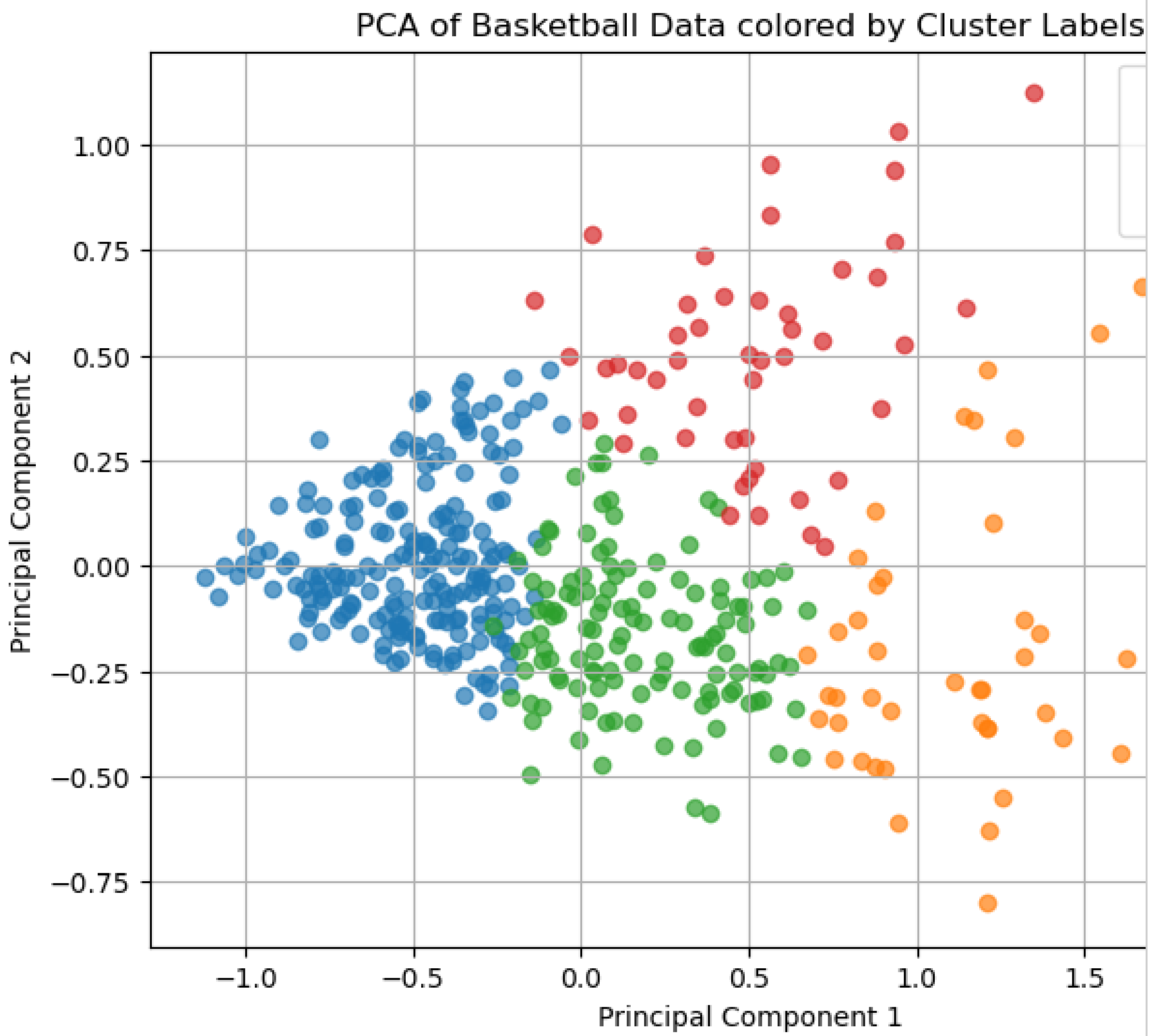
q2e results: All test cases passed!

Q2(f)

Apply PCA to the basketball data. Plot the first two principal components, colored by the best group labels found above.

In [180]:

```
# Run PCA on the nba data and plot the first two principal components
#  colored by the group labels.
pca = PCA(n_components=2)
pca_result = pca.fit_transform(nbaScaled)
plt.figure(figsize=(8, 6))
for cluster in range(4):
    plt.scatter(pca_result[kmeans.labels_ == cluster, 0],
                pca_result[kmeans.labels_ == cluster, 1],
                label=f'Cluster {cluster}', alpha=0.7)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Basketball Data colored by Cluster Labels')
plt.legend()
plt.grid(True)
plt.show()
```



Q3 : Clustering - Spotify Music

For this problem you will look at popular streaming music. Specifically, Spotify's top 100 streaming songs. For each song information about the song is described with different properties: `duration`, `energy`, `key`, etc.

Q3(a) - Load and Prepare the Data

Load in the `music.csv` data.

The clustering algorithms will only consider variables of `duration` to the end of the DataFrame.

Standardize the variables to be used in clustering.

In [181]:

```
# Load in music data
```

```
music = pd.read_csv('data/music.csv')
#consider variable of duration to the end of the dataframe
considered_col = music.columns[5:]
# Standardize the considered columns
scaler = StandardScaler()
music[considered_col] = scaler.fit_transform(music[considered_col])

music.head()
```

Out [181]:

```
               Song                         Artist  Streams (Billions)  \
0   Blinding Lights                     The Weeknd               3.449
1      Shape of You                     Ed Sheeran               3.398
2      Dance Monkey                    Tones And I               2.770
3  Someone You Loved                 Lewis Capaldi               2.680
4          Rockstar  Post Malone featuring 21 Savage               2.620

  Release Date                      id  duration    energy       key  \
0    29-Nov-19  0VjIjW4GlUZAMYd2vXMi3b -0.379751  0.656231 -1.202571
1    06-Jan-17  7qiZfU4dY1lWllzX7mPBI3  0.329251  0.166413 -1.202571
2    10-May-19  2XU0oxnq2qxCpomAAuJY8K -0.180733 -0.235490  0.182880
3    08-Nov-18  7qEHsqek33rTcFNT9PFqLf -0.740472 -1.384680 -1.202571
4    15-Sep-17  0e7ipj03S05BNilyu5bRzt  0.005846 -0.662511 -0.094211

   loudness      mode  speechiness  acousticness  instrumentalness  liveness  \
0  0.121245  0.733799    -0.414447     -0.956146         -0.159096 -0.621522
1  1.497770 -1.362770    -0.183746      1.179255         -0.161053 -0.593711
2 -0.111928 -1.362770    -0.045778      1.588251         -0.158919 -0.136456
3  0.248840  0.733799    -0.729964      1.805645         -0.161053 -0.496370
4  0.020170 -1.362770    -0.285526     -0.504629         -0.159615 -0.283694

    valence     tempo  danceability
0 -0.750727  1.692429     -0.942577
1  1.916528 -0.859868      1.218755
2  0.049002 -0.790131      1.211806
3 -0.250338 -0.386542     -1.032922
4 -1.666619  1.311292     -0.449154
```

In [182]:

```
grader.check("q3a")
```

Out [182]:

q3a results: All test cases passed!

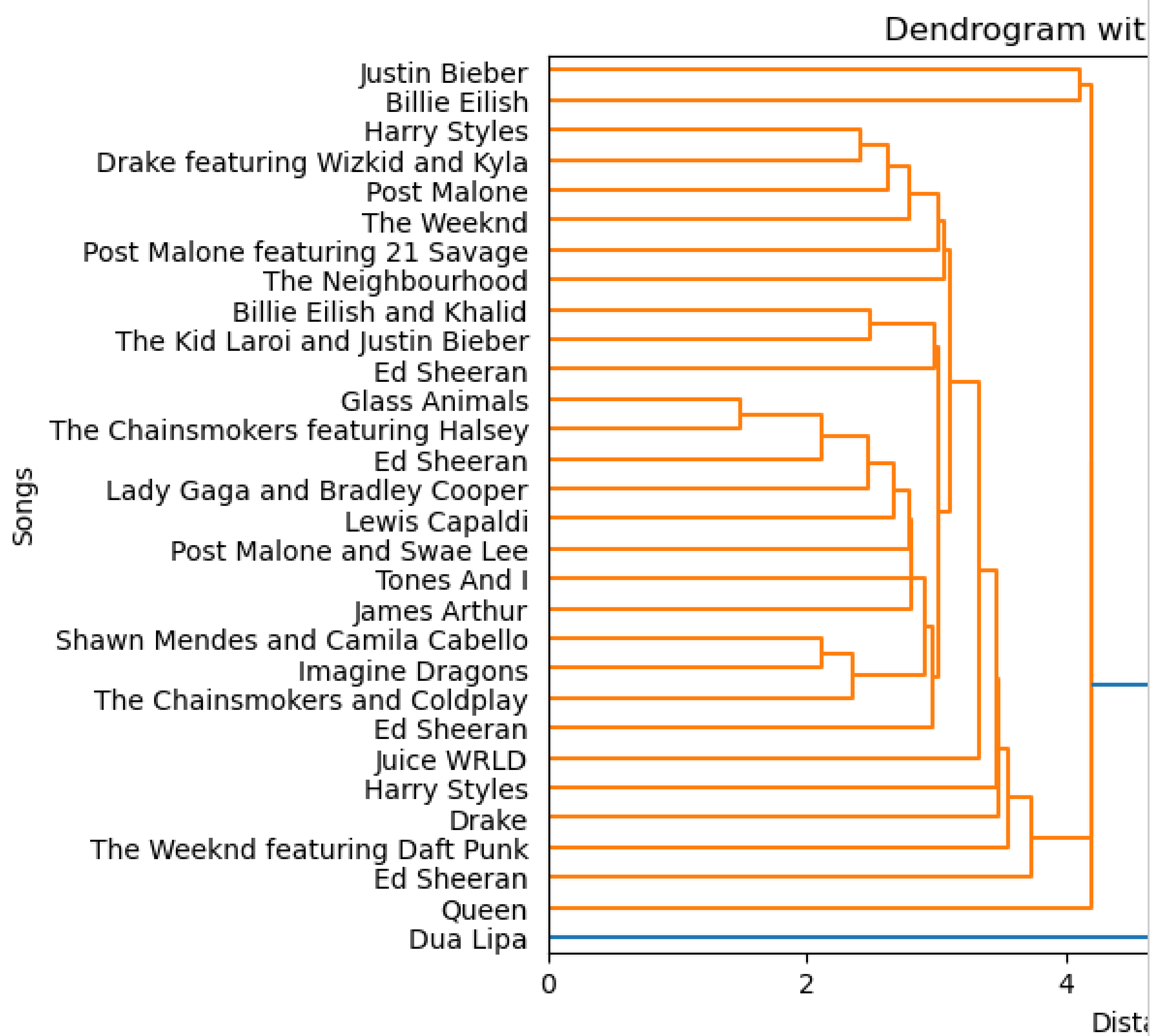
Q3(b) - Hierarchical Clustering

Perform Hierarchical clustering with **single** linkage on just the top 30 songs.

Report results in a dendrogram, `dg_single` and label the samples by the Artist.

In [183]:

```
# Perform Hierarchical clustering with single linkage on top 30 songs
# Report results in a dendrogram, dg_single
top_30_songs = music.head(30)
clustering_vars = top_30_songs.iloc[:,5:].values
#find the ecludian distance
distance_matrix = pdist(clustering_vars, metric = 'euclidean')
#create a linkage matrix using single linkage
link_matrix = hierarchy.linkage(distance_matrix, 'single')
#plot the dendrogram
plt.figure(figsize=(8,6))
dg_single = hierarchy.dendrogram(link_matrix,
          labels = top_30_songs['Artist'].tolist(), orientation='right')
plt.title('Dendrogram with Single Linkage')
plt.ylabel('Songs')
plt.xlabel('Distance')
plt.show()
```



In [184]:

```
grader.check("q3b")
```

Out [184]:

q3b results: All test cases passed!

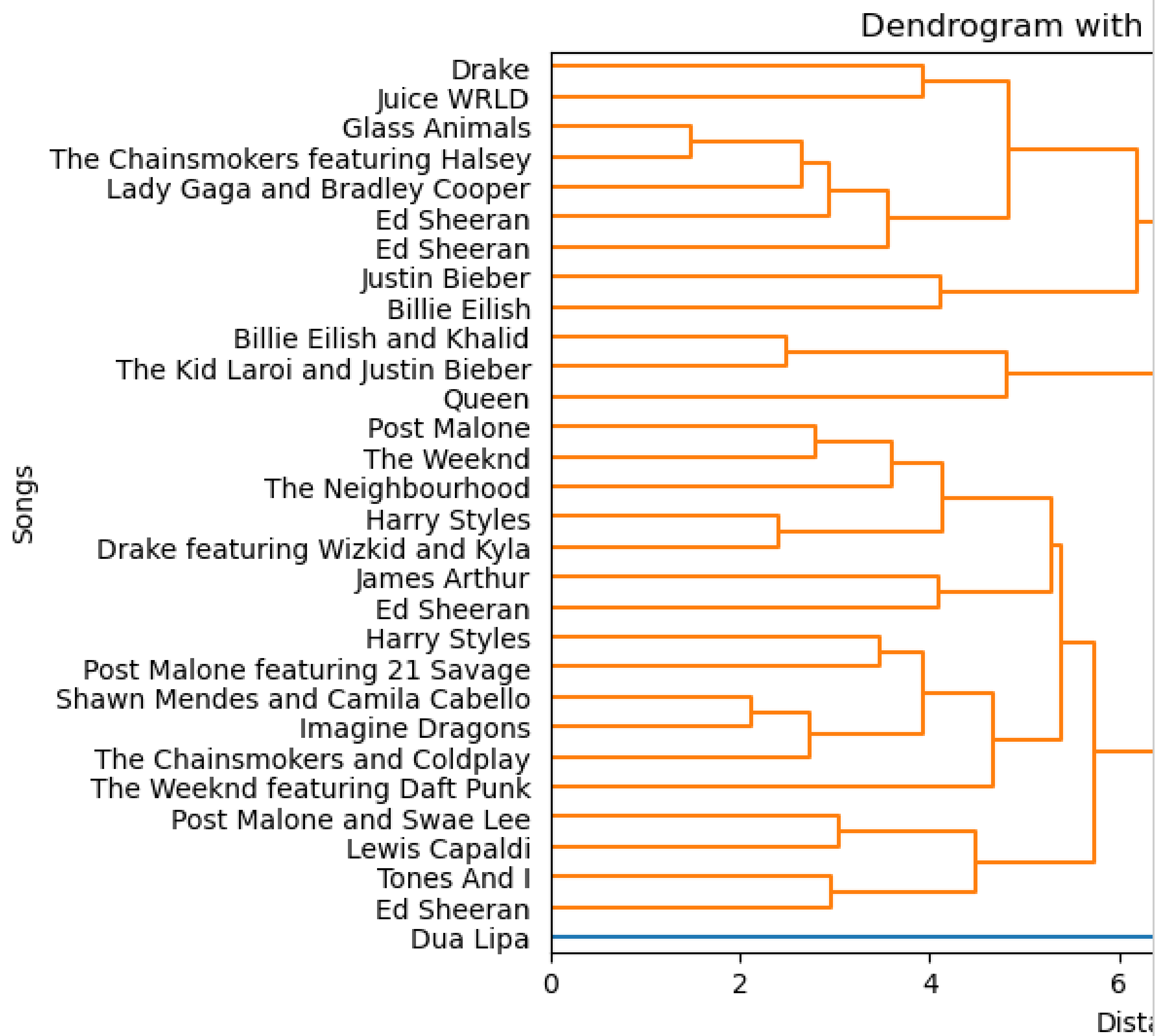
Q3(c) - Hierarchical Clustering, part 2

Perform Hierarchical clustering with **complete** linkage on just the top 30 songs.

Report results in a dendrogram, `dn_complete` and label the samples by the Artist.

In [185]:

```
# Perform Hierarchical clustering with complete linkage on top 30 songs
# Report results in a dendrogram, dg_complete
#create a linkage matrix using complete linkage
link_matrix = hierarchy.linkage(distance_matrix, 'complete')
#plot the dendrogram
plt.figure(figsize=(8,6))
dg_complete = hierarchy.dendrogram(link_matrix, labels =
top_30_songs['Artist'].tolist(), orientation='right')
plt.title('Dendrogram with Complete Linkage')
plt.ylabel('Songs')
plt.xlabel('Distance')
plt.show()
```



In [141]:

```
grader.check("q3c")
```

Out [141]:

q3c results: All test cases passed!

Q3(d) - Hierarchical Clustering, part 3

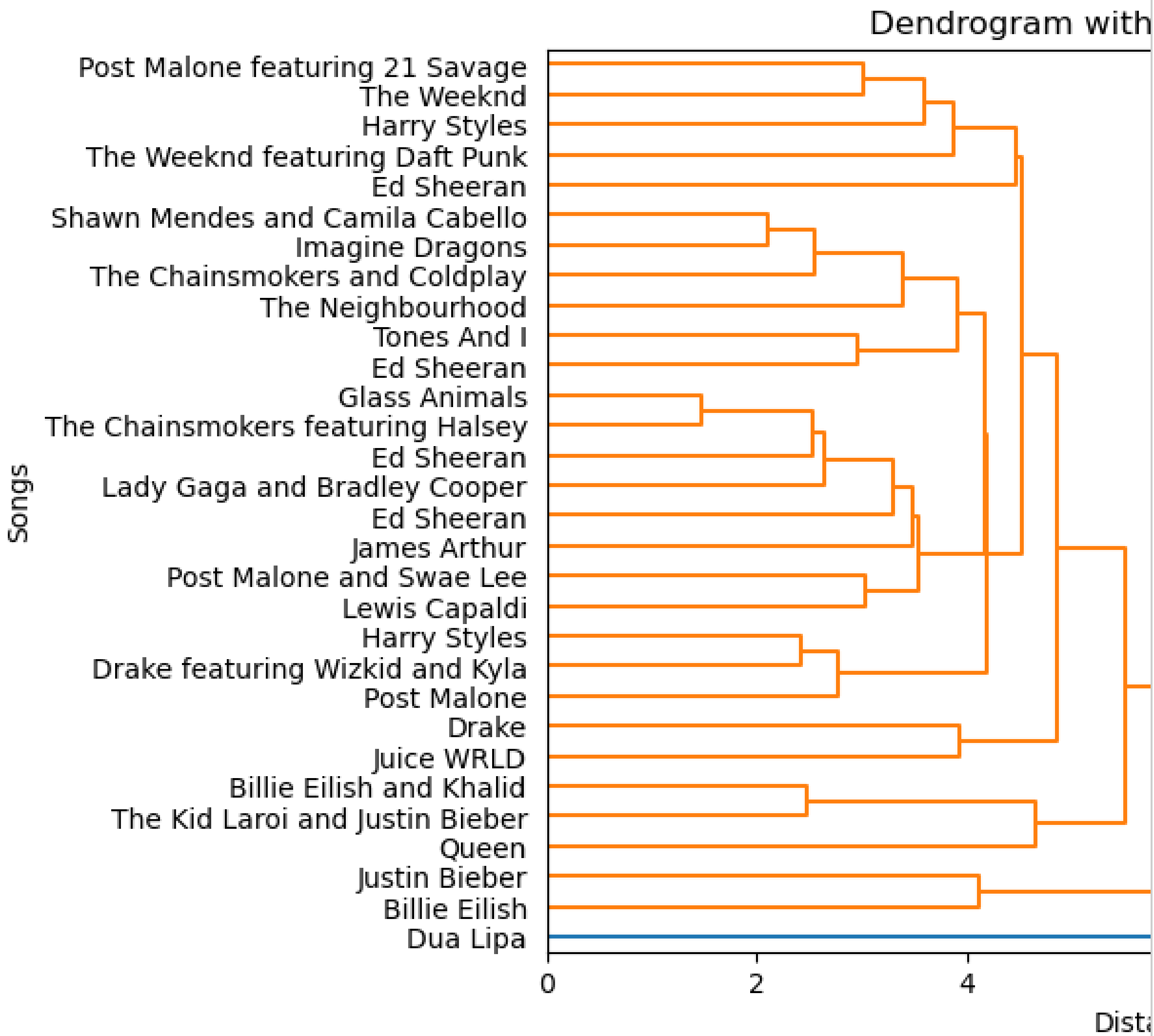
Perform Hierarchical clustering with **aveage** linkage on just the top 30 songs.

Report results in a dendrogram, `dn_average` and label the samples by the Artist.

In [142]:

```
# Perform Hierarchical clustering with average linkage on top 30 songs
# Report results in a dendrogram, dg_average
#create a linkage matrix using average linkage
link_matrix = hierarchy.linkage(distance_matrix, 'average')
#plot the dendrogram
plt.figure(figsize=(8,6))
```

```
dg_average = hierarchy.dendrogram(link_matrix, labels =
top_30_songs['Artist'].tolist(), orientation='right')
plt.title('Dendrogram with average Linkage')
plt.ylabel('Songs')
plt.xlabel('Distance')
plt.show()
```



In [143]:

```
grader.check("q3d")
```

Out [143]:

q3d results: All test cases passed!

Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

NOTE the submission must be run on the campus linux machines. See the instruction in the Canvas assignment.

In [145]:

```
# Save your notebook first, then run this cell to export your submission.
grader.export(pdf=False)
```

<IPython.core.display.HTML object>

▾ .OTTER_LOG Download

1	Large file hidden. You can download it using the button above.

▾ __zip_filename__ Download

1	a4 (1) (1)_2024_03_29T23_31_58_633182.zip