Project Script

Thomas Zwiller

Introduction

William Shakespeare, otherwise known as the Bard of Avon or simply the 'Bard,' is one of the most influential authors of all time. His 38 plays and more than 130 sonnets have been translated into countless languages and continue to be studied and performed. However, some academics who study Shakespeare's works have concluded that plays historically accredited to Shakespeare were either co-authored by another unnamed author or were written by another author altogether. Other academics think Shakespeare may have not written any of his plays, partly because Shakespeare was the son of a glove maker and, thus, a member of the working class with a basic grammar school education. While somewhat classist, this argument is backed by the lack of a Shakespeare paper trail; 'The Bard' has only a handful of signatures attributed to him, and they are of a rather low quality. The goal of this project is to determine the authorship of three plays with contested authorship:

Henry VI Part 1

Arden of Feversham

Edward III

These three plays have been attributed to multiple authors who could have worked with Shakespeare or independently. The authorship of the contested plays will be compared to known Shakespeare plays and the plays of Christopher Marlowe, a contemporary of Shakespeare in the Elizabethan era and a potential co-author.

Step 1.

My initial idea was to comb through Irish Illustrated (the company I work for) and get 20-30 articles from each author to perform a multi-class prediction. The main challenge I anticipated was differentiating between a handful of people on staff who write a majority of the "quick hitter" stories. Weekly columns and profiles allow for the most personality to show, but a quick hitter uses the same structure and format and, by design, is relatively short.

However, I didn't even get that far. Irish Illustrated (and likely all 247 Websites) rejected my BeautifulSoup requests for every page I tried to pull, even the complimentary pieces. I tried Blue Gold Illustrated, which allowed me to pull the free articles but not the premium articles, which would make authorship a more complex challenge. I checked a few other Notre Dame-focused outlets, but they outright rejected the request or only allowed me to access free content.

I finally settled on using two authors from CBS (which had zero issues with me scraping their content). If I could pull content from those two and distinguish between the two, I'd feel good about expanding and including new writers who covered different sports. Here's my first attempt:

```
#importing required libraries
from bs4 import BeautifulSoup
import re
import numpy as np
import requests
import pandas as pd
from langdetect import detect, DetectorFactory

#initalizing an empty list
author_database = []

#This was the loop I used to get the first 25 pages of Chip Pattersons writer
page, which allowed me to collect a large amount of articles
for i in range(25):
  #getting the link
  link_2_req = f'https://www.cbssports.com/writers/chip-patterson/{i}/'

  #checking the request
  request = requests.get(link_2_req)

  #getting the HTML
  Link_Soup = BeautifulSoup(request.content, 'html.parser')

  #pulling out the links
  author_links = Link_Soup.select('a[data-ajax="false"]')

  #and then manufacturing the links that I'll need to go and grab articls
  for link in author_links:
    #grabbing the links
    href = link.get('href')
    #the first part of all cbs sports links
    link_start = 'https://www.cbssports.com/'
    #concatenating
    href = link_start + href
    #appending to the author database
    author_database.append(href)

#a new empty list that will house all the articles I've pulled
article = []

for link in author_database:
  #checking the request
  request = requests.get(link)
  #getting the html content
```

```
  soup = BeautifulSoup(request.content, 'html.parser')
  #finding all the sentences with a p tag
  all_sentences = soup.find_all('p')
  #empty list for the sentences in the article that is re-intialized each time
the loop runs
  sentences = []
  for sentence in all_sentences:
   #the following if statements checked to make sure the text didn't contain any
of the standard boilerplate content that was contained in the html as a sentence
    #if it did, it skipped it
    if sentence.get_text(strip=True) == 'If not listed, please contact your TV
provider.':
      continue
    elif 'this site' in sentence.get_text(strip=True):
      continue
    elif '©' in sentence.get_text(strip=True):
      continue
        elif 'registered trademark of CBS Broadcasting Inc.' in
sentence.get_text(strip=True):
      continue
    elif 'Images by Getty Images and Imagn' in sentence.get_text(strip=True):
      continue
   #any text that made it through the if statements got appended to the sentences
list
    else:
      sentences.append(sentence.text)
  #once the article was completely loaded in I passed it to a dictionary which
included the authors name
  temp_article = {'Author' : 'Chip Patterson',
                  'Article' : ' '.join(sentences)}
  article.append(temp_article)

#and made the dictionary into a dataframe
Patterson_DF = pd.DataFrame(article)

Patterson_DF.head()
```

	Author	Article
0	Chip Patterson	Spring practice is in the books across college...
1	Chip Patterson	The Power Five is back in college football. N...
2	Chip Patterson	The deadline for undergraduate players to ente...
3	Chip Patterson	With the 2025 NFL Draft in the books, the foot...
4	Chip Patterson	Fran Brown put together one of the season's bi...

I got 360 Chip Patterson stories, which gave me a solid starting point. Because the links to the author pages on CBS only differ in the author's name, I could pretty much re-use the code I used to get the Patterson stories, only changing the end of the link to Tom Fornelli.

```
#empty list to house page links
author_database = []

#like with Chip, I used a for loop to get the first 25 author pages from Fornelli's
page
for i in range(25):
  #the link
  link_2_req = f'https://www.cbssports.com/writers/tom-fornelli/{i}/'
  #the request
  request = requests.get(link_2_req)
  #collecting the html
  Link_Soup = BeautifulSoup(request.content, 'html.parser')
  #getting the links to the stories
  author_links = Link_Soup.select('a[data-ajax="false"]')
  #making the links to each story using the default first part of the CBS link
  for link in author_links:
    href = link.get('href')
    link_start = 'https://www.cbssports.com/'
    href = link_start + href
    author_database.append(href)

#making the article list
article = []

#I used this for loop to visit each page and scrape it
for link in author_database:
  #for each story in the database I passed the link and got the request
  request = requests.get(link)
  #then the html
  soup = BeautifulSoup(request.content, 'html.parser')
  #added all the sentences to all_sentences
  all_sentences = soup.find_all('p')
  #and made a new empty list to temporarily house the sentences
  sentences = []
  for sentence in all_sentences:
    #once again trying to limit what gets through to the author database
    if sentence.get_text(strip=True) == 'If not listed, please contact your TV
provider.':
      continue
    if 'this site' in sentence.get_text(strip=True):
      continue
    if '©' in sentence.get_text(strip=True):
      continue
          if   'registered   trademark   of   CBS   Broadcasting   Inc.'   in
```

```
sentence.get_text(strip=True):
      continue
    if 'Images by Getty Images and Imagn' in sentence.get_text(strip=True):
      continue
    #if the sentence passed through everything, it was added
    else:
      sentences.append(sentence.text)
  #and then the sentences list was put into a dictionary
  temp_article = {'Author' : 'Tom Fornelli',
                  'Article' : ' '.join(sentences)}
  #the dict was sent to the article list
  article.append(temp_article)

#and that list was passed to a DF
Fornelli_DF = pd.DataFrame(article)

Fornelli_DF.head()
```

	Author	Article
0	Tom Fornelli	I've long argued that May is the single worst ...
1	Tom Fornelli	Like a hidden camera you don't know is there, ...
2	Tom Fornelli	Have you ever sat down and thought about? I me...
3	Tom Fornelli	Quarterback is the most important position on ...
4	Tom Fornelli	Everything about the way Nico Iamaleava's time...

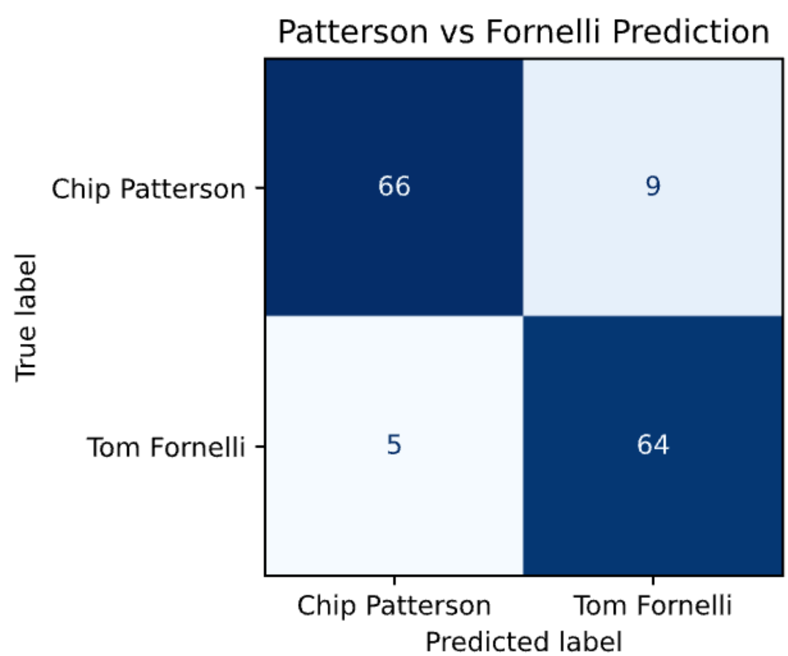
I now had 360 stories from both authors. Now, I could combine the two into one main author data frame and begin predicting which author wrote which story.

Step 2

```
#getting the functions I needed, mainly from the sklearn library
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import matplotlib.pyplot as plt
import numpy as np

#I made the two author dataframes into one main author dataframe
Author_Data = pd.concat([Patterson_DF, Fornelli_DF], ignore_index=True)
```

```
#and then I used the train_test_split function to split the data into X_train,
X_test, y_train and y_test
X_train,  X_test,  y_train,  y_test  =  train_test_split(Author_Data['Article'],
Author_Data['Author'], test_size=.2)

#and then I initalized a word vectorizer with the goal of removing any stop words
wordVectorizer = CountVectorizer(lowercase=True,
                                 ngram_range=(1,2),
                                 stop_words="english", min_df=2)

#and then fit the vectorizer on the author data from the main dataframe
wordVectorizer.fit(Author_Data['Article'])

#then getting the train text features
trainTextFeatures = wordVectorizer.transform(X_train).toarray()

#and the test text features
testTextFeatures = wordVectorizer.transform(X_test).toarray()

#I then kept the top 75% of the features
MAX_FEATURE_PERCENTAGE = 75
KEEP_FEATURES = int((len(trainTextFeatures[0])*MAX_FEATURE_PERCENTAGE)/100)

#and used a feature selector to fit the training data
featureSelector = SelectKBest(chi2, k=KEEP_FEATURES)
featureSelector.fit(trainTextFeatures, y_train)

#transforming the train data
trainTextFeatures = featureSelector.transform(trainTextFeatures)
#and then the text data
testTextFeatures = featureSelector.transform(testTextFeatures)

#using an XGBoost model to predict
bst = XGBClassifier(n_estimators= 100,
                    max_depth= 5,
                    learning_rate= 0.1,
                    objective= 'multi:softmax',
                    num_class= 2)

#making the label encoder to pass the y values to the boost model
label_encoder = LabelEncoder()

#encoding the labels
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

#fitting the model
bst.fit(trainTextFeatures, y_train_encoded)
```

```
#making predictions
preds = bst.predict(testTextFeatures)

#starting the confusion matrix
cm = confusion_matrix(y_test_encoded, preds)

#printing the confusion matrix
cm = confusion_matrix(y_test_encoded, preds)
disp                    =                 ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=label_encoder.classes_)
disp.plot(cmap=plt.cm.Blues, colorbar=False)
plt.title("Patterson vs Fornelli Prediction")
plt.show()
```



With my model made, I tuned it by playing with the number of estimators and the learning rate, and I settled on the current version. Based on the tokenized text data, the model could detect which writer was which.

I thought about including a few more authors and wrapping up the project, but it didn't feel like there was a legitimate application of the overall process. I had been able to prove which authors had written the text using an XGBoost model, yes, but which author had written the piece was publicly available.

I thought about possible uses of authorship identification, and the book *Yellow Face* came to mind. A basic synopsis is an author who witnesses a fellow writer die, steals the deceased author's unpublished manuscript, turns it into her own work, and becomes famous because of the book's success. The book wrestles with interesting ideas and themes, mainly focusing on how racial identity impacts authorship. The book also asks whether someone who is a cultural outsider should tell the stories of other cultures, and if they do, is it stealing or empowering?

However, the idea of authorship when a label is only suspected and not wholly known appealed to me. I just needed a real-world application.

And so I turned to one of the most well-known writers in history: Shakespeare.

Using this link

https://www.folger.edu/explore/shakespeares-works/download/

I downloaded most of Shakespeare's complete works as text files, which I needed to parse, clean, and store. Luckily, I was able to use the Python Belt Challenge as a starter code of sorts.

Step 3

```
#getting glob, pandas and regex
import glob as glob, pandas as pd, re, os

#Import path here:
files = '/Users/TomTheIntern/Desktop/Mendoza/Mod 3/Unstructured/Project Script/
shakespeares-works_TXT_FolgerShakespeare'

#setting the path
path = os.path.join(files, '*.txt')

#getting the Shakespeare stories
Shakespeare_Stories = glob.glob(path)

#making some placeholder lists
titles = []
total_text =[]

#the things that were constants in most stories that I didn't want to include
drop_list = ['\n', '',
             'ACT 1', 'ACT 2', 'ACT 3', 'ACT 4', 'ACT 5',
             'Scene 1', 'Scene 2', 'Scene 3',
             '=====', '=======']

#and now to loop through every story
for story in Shakespeare_Stories:
  #this allowed me to get the title
  title = re.search(r"(?<=FolgerShakespeare/)(.*?)(?=_TXT)", story)
   #I set this flag to false, which made the following nested loop ignore the
first lines of the play until a certain condition was met
  flag = False
  #empty list
```

```
  story_text = []
  with open(story, "r") as file:
    #this read in the text of the file
    file_text = file.readlines()
    for line in file_text:
      #here was the condition that had to be met
       #it originally was a lot cleaner, but as I added in stories from a few different sources, I had to adapt
     if line.strip() == 'ACT 1' or line.strip() == 'THE ARGUMENT' or line.strip() == 'Venus and Adonis' or line.strip() == 'The Phoenix and Turtle' or line.strip() == 'From fairest creatures we desire increase,' or line.strip() == 'ACT I':
        #if the condition was met, the flag was set to true and thus began to capture content
        flag = True
          #unless it was in the drop list, which was other text that either wasn't helpful or could potentially be in other plays, thus making it harder to ascertain the truth authorship
      if line.strip() in drop_list:
        continue
        #this prevented anything that was a stage direction from being included
      if re.search(r"(?<=\[)(.*?)(?=\])", line):
        continue
        #if the flag was set to true, I removed any all uppercase characters, only keeping dialogue
      if flag:
        line = re.sub(r'[A-Z]{2,}\'?[A-Z]{1,3}', '', line.strip())
        #stripping the string
        cleaned_string = line.strip()
        if cleaned_string:
          #Only append if the string is not empty
          story_text.append(cleaned_string)
  #eventually, this would result in a dictionary with the Author, text and title
  total_text.append({'Author': 'Shakespeare',
                  'Title': title.group(),
                  'Text': story_text})

#once the for loop went through every story I made it into a dataframe
Shakespeare_DF = pd.DataFrame(total_text)

#and wrote it out to a dataframe
Shakespeare_DF.to_csv('Shakespeare.csv', index=False)

Shakespeare_DF.head()
```

	Author	Title	Text
0	Shakespeare	much-ado-about-nothing	[[Enter Leonato, Governor of Messina, Hero his...

	Author	Title	Text
1	Shakespeare	richard-iii	[Now is the winter of our discontent, Made glo...
2	Shakespeare	the-winters-tale	[If you shall chance, Camillo, to visit Bohemi...
3	Shakespeare	richard-ii	[[Enter King Richard, John of Gaunt, with othe...
4	Shakespeare	henry-vi-part-3	[[Alarum. Enter Richard Plantagenet, Duke of Y...

I now had my primary class but needed an author to compare to. There were a handful to choose from, so I selected Christopher Marlowe since he had a handful of plays that I could download as txt files from the following link:

https://kitmarlowe.org/files-for-text-analysis/7452/

Step 4

```
#file path
files = '/Users/TomTheIntern/Desktop/Mendoza/Mod 3/Unstructured/Project Script/
Christopher Marlowe'

#setting the path
path = os.path.join(files, '*.txt')

#getting the txt files
Marlowe_Stories = glob.glob(path)

#setting i = 1
i = 1
#and then the total text as a list
total_text = []

for story in Marlowe_Stories:
  #for some reasons my files were encoded weirdly and I had to figure out how
to read them in as normal string literals
    #https://www.geeksforgeeks.org/effect-of-b-character-in-front-of-a-string-
literal-in-python/
  with open(story, "r", encoding="cp1252") as file:
    marlowe_text = []
     #I was less interested in the names here so I just pulled them in as a
numerical value
    play_title = f'Marlowe Play {i}'
    #read in the text
    file_text = file.readlines()
    for line in file_text:
```

```
        #cleaned the text
        cleaned_line = line.strip()
        #appended it
        marlowe_text.append(cleaned_line)
        #and passed the dict to the list
      total_text.append({'Author': 'Marlowe',
                          'Title': play_title,
                          'Text': marlowe_text})
      i += 1

#and then made a dataframe
Marlowe_DF = pd.DataFrame(total_text)

Marlowe_DF.head()
```

	Author	Title	Text
0	Marlowe	Marlowe Play 1	[, , Enter Chorus., NOt marching now in fields...
1	Marlowe	Marlowe Play 2	[, Tamburlaine, the great., [portrait of Tambu...
2	Marlowe	Marlowe Play 3	[, Tamburlaine the Great., , Who, from a Scyth...
3	Marlowe	Marlowe Play 4	[, , The troublesome, reign and lamentable dea...
4	Marlowe	Marlowe Play 5	[, THE MASSACRE AT PARIS., , With the Death of...

With my Marlowe plays safely in a data frame, I now had the information I needed to address the authorship problem.

Step 5

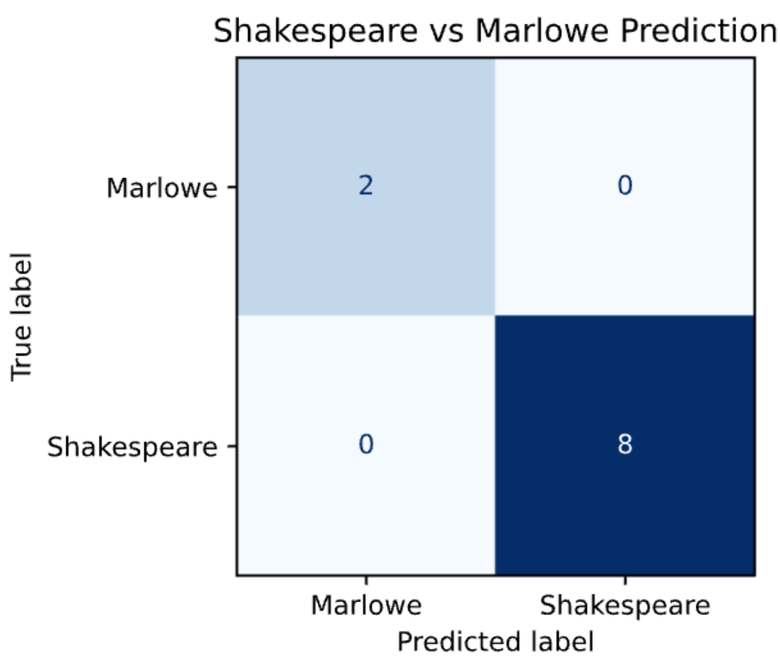
```
#grabbing the libraries I need
import random
from sklearn.svm import SVC
from sklearn.metrics import RocCurveDisplay

#I made a copy of the Shakespeare dataframe because I needed to remove the plays
in question from the dataframe, but wanted to be able to reference the dataframe
again later without re-running the Shakespeare code
Shakespeare_DF_copy = Shakespeare_DF.copy()

#set aside the plays that were 'unknowns'
Unknown_Plays = Shakespeare_DF.loc[
    (Shakespeare_DF['Title'] == 'henry-vi-part-1') |
     (Shakespeare_DF['Title'] == 'arden-of-feversham')|
     (Shakespeare_DF['Title'] == 'edward-iii')]

#removed the unknowns from the copy dataframe
```

```
Shakespeare_DF_copy                                                    =
Shakespeare_DF[~Shakespeare_DF['Title'].isin(Unknown_Plays['Title'])]

#added the known Shakespeare plays with the Marlowe plays
Play_Data = pd.concat([Shakespeare_DF_copy, Marlowe_DF], ignore_index=True)

#partitioned the data
X_train,   X_test,   y_train,   y_test   =   train_test_split(Play_Data['Text'],
Play_Data['Author'], test_size=.2, random_state = 45)

#initalized a wordVectorized
wordVectorizer = CountVectorizer(lowercase=True,
                                 ngram_range=(1,2),
                                 stop_words="english", min_df=1)

#and then I had to make each document into strings instead of tokens, otherwise
I got an error
X_train = [' '.join(doc) if isinstance(doc, list) else doc for doc in X_train]
X_test = [' '.join(doc) if isinstance(doc, list) else doc for doc in X_test]

#and thenfit the wordVectorizer
wordVectorizer.fit(X_train)

#got the text features for the test and train data
trainTextFeatures = wordVectorizer.transform(X_train).toarray()
testTextFeatures = wordVectorizer.transform(X_test).toarray()

#kept the top 75% of the feautres
maxFeaturePercentage = 75
keepFeatures = int((len(trainTextFeatures[0])*maxFeaturePercentage)/100)

#selected the key features after fitting a feature selector
featureSelector = SelectKBest(chi2, k=keepFeatures)
featureSelector.fit(trainTextFeatures, y_train)

trainTextFeatures = featureSelector.transform(trainTextFeatures)
testTextFeatures = featureSelector.transform(testTextFeatures)

#Originally I made an XGBoost model because it had done such a good job of
capturing the differences in the sports authorship data, however it really
struggled and predicted all of the test data as Shakespeare, which wasn't ideal.
Even after using Smote it struggled.

class_model = XGBClassifier(
    n_estimators=100,
    max_depth=4,
    learning_rate=0.001,
    objective='binary:logistic',
```

```
    scale_pos_weight=5,
    min_child_weight=2,
    reg_alpha=1,
    reg_lambda=1,
    eval_metric='logloss',
    use_label_encoder=False
)

#I decided to use an SVM model because I found a research paper that came to the
conclusion that SVMs outperformed most other models when performing authorship
tasks on limited text https://www.sciencedirect.com/science/article/pii/S01674
04820302194?via=ihub

svm_model = SVC(
  class_weight='balanced',
  C = 1.4, #I performed a random grid search to tune the regularization
  max_iter = 1000,
  random_state = 42,
  probability = True)

#encoding the labels
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

#fitting the model
svm_model.fit(trainTextFeatures, y_train_encoded)

#I had the SVM return probabilties
probs = svm_model.predict_proba(testTextFeatures)
classes = []

#and then if the probability was greater than 50 for the first value in the
probability pair was greater than 50 (meaning Marlowe) I assigned it a 0,
otherwise it was a 1
for prob in probs:
  if prob[0] > .50:
    author = 0
  else:
    author = 1
  classes.append(author)

#printing the classes and labels so I can verify/get the results
print(classes)
print(probs)

#and then printing a confusion matrix
cm              =            confusion_matrix(y_test_encoded,            classes,
```

```
labels=label_encoder.transform(label_encoder.classes_))
disp              =              ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=label_encoder.classes_)
disp.plot(cmap=plt.cm.Blues, colorbar=False)
plt.title("Shakespeare vs Marlowe Prediction")
plt.show()
```

```
[0, 1, 1, 1, 1, 1, 1, 0, 1, 1]
[[0.7235462  0.2764538 ]
 [0.04849031 0.95150969]
 [0.00563198 0.99436802]
 [0.01099801 0.98900199]
 [0.01898577 0.98101423]
 [0.0335014  0.9664986 ]
 [0.00388864 0.99611136]
 [0.99221884 0.00778116]
 [0.01217116 0.98782884]
 [0.03119795 0.96880205]]
```



With a working model that returned a high % for the Shakespeare plays, I felt confident about the SVM's ability to correctly determine authorship despite the limited sample size. My next step was to retrain the model on the training and test and then apply it to the unknown plays.

Step 6

```python
#making an all_train frame from the Shakespeare copy and the Marlow df
all_train = pd.concat([Shakespeare_DF_copy, Marlowe_DF], ignore_index=True)

#can't pass it as a list like I had
all_train_text = [' '.join(doc) if isinstance(doc, list) else doc for doc in
all_train['Text']]

final_test = [' '.join(doc) if isinstance(doc, list) else doc for doc in
Unknown_Plays['Text']]

#fitting a word vectorizer on the training data
wordVectorizer.fit(all_train_text)

#and then getting the text features of the train and the unknown
trainTextFeatures = wordVectorizer.transform(all_train_text).toarray()
unknownTextFeatures = wordVectorizer.transform(final_test).toarray()

#creating the shakes model
Shake_Model = SVC(
  class_weight='balanced',
  C = 1.4,
  max_iter = 1000,
  random_state = 42,
  probability = True)

#encoding the training labels
y_train_encoded = label_encoder.fit_transform(all_train['Author'])

#training the model
Shake_Model.fit(trainTextFeatures, y_train_encoded)

#making predictions for the unknown sample
unknown_preds = Shake_Model.predict_proba(unknownTextFeatures)

#and returning the probabilites
print("Predicted labels:")
print("Marlowe      vs     Shakespeare")
print(unknown_preds)
```

```
Predicted labels:
Marlowe      vs     Shakespeare
[[0.16777452 0.83222548]
 [0.204175   0.795825  ]
 [0.21849954 0.78150046]]
```

The model determined that Shakespeare had a pretty solid chance of writing the plays in question, but it is worth noting that percentages were much lower than they had been with the confirmed

Shakespeare plays, which makes sense. Had the model been overly confident, I would have worried that there was a class imbalance issue, but the 75-85% range can solidly support the idea that Shakespeare did author the plays without being too confident.

However, I also wanted to look at some additional factors to determine why the model returned the percent that it did.

Step 7

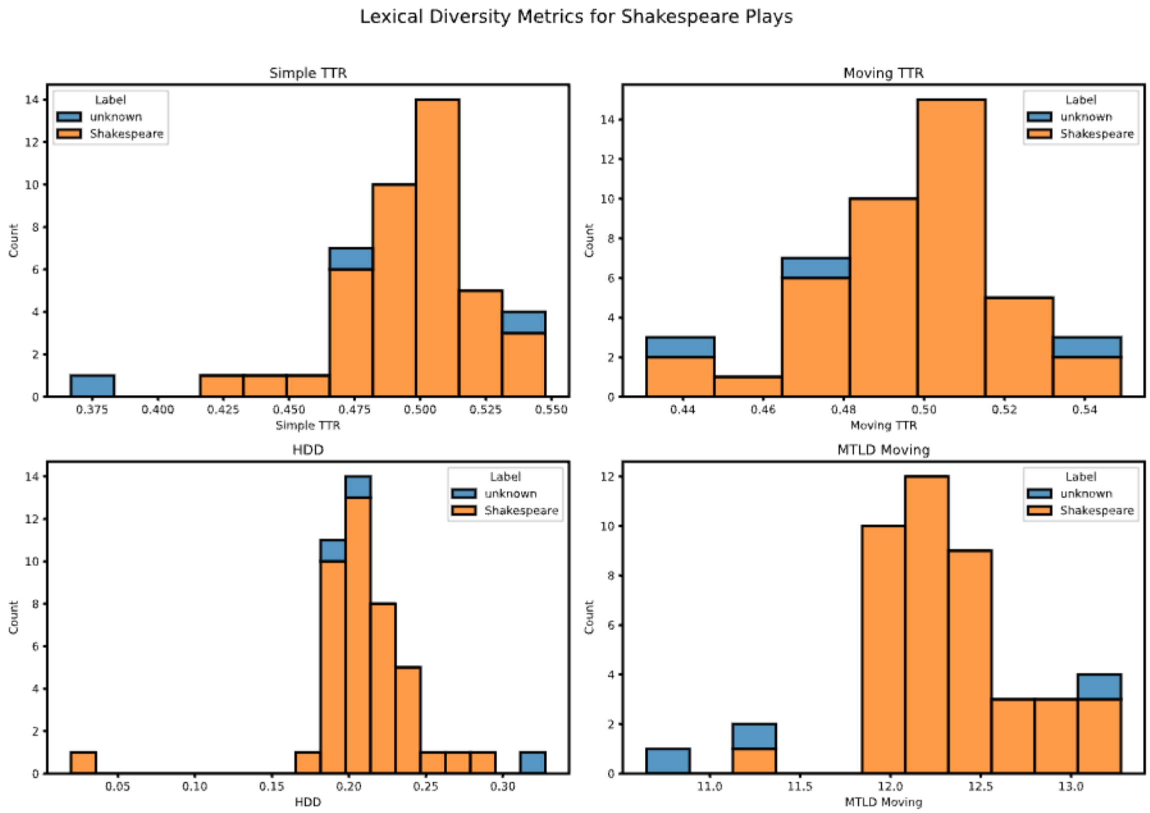
```
#grabbing the last set of libraries I will need
from lexical_diversity import lex_div as ld
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#setting up an empty list
results = []
#i so the name of the play changes
i = 0
#dropping nulls
min_word_count = 1

#iterating over the Shakespeare copy
for play in Shakespeare_DF_copy['Text']:
  #empty lists
  simple_ttr = []
  mov_ttr = []
  hdd = []
  mtld_mov = []
  #nested for loop for each line in the play
  for line in play:
    if len(line.split()) < min_word_count:
        continue
    #appending each lexcial diversity metric to its own list
    simple_ttr.append(ld.ttr(line))
    mov_ttr.append(ld.mattr(line))
    hdd.append(ld.hdd(line))
    mtld_mov.append(ld.mtld_ma_bid(line))
    #making a dict for each play by taking the mean
  results.append({
        'Title': Shakespeare_DF['Title'].values[i],
        'Simple TTR': np.mean(simple_ttr),
        'Moving TTR': np.mean(mov_ttr),
        'HDD': np.mean(hdd),
        'MTLD Moving': np.mean(mtld_mov),
    })
  i += 1

#making a dataframe
```

```python
Text_Analysis = pd.DataFrame(results)
Text_Analysis.to_csv('Text_Analysis.csv', index=False)

#and then repeating the process for the unknown plays
unknown_results = []
i = 0

#for plays in unknown
for play in Unknown_Plays['Text']:
  simple_ttr = []
  mov_ttr = []
  hdd = []
  mtld_mov = []

  #for each line in each play
  for line in play:
    #getting the lexical diversity
    simple_ttr.append(ld.ttr(line))
    mov_ttr.append(ld.mattr(line))
    hdd.append(ld.hdd(line))
    mtld_mov.append(ld.mtld_ma_bid(line))
  #making it a dict from the average vals
  unknown_results.append({
        'Title': Unknown_Plays['Title'].values[i],
        'Simple TTR': np.mean(simple_ttr),
        'Moving TTR': np.mean(mov_ttr),
        'HDD': np.mean(hdd),
        'MTLD Moving': np.mean(mtld_mov),
    })
  i += 1

#making them into a dataframe
Unknown_Analysis = pd.DataFrame(unknown_results)
Unknown_Analysis.to_csv('Unknown_Analysis.csv', index=False)

#making labels for plotting
Unknown_Analysis['Label'] = 'unknown'
Text_Analysis['Label'] = 'Shakespeare'

#merging
All_Analysis = pd.concat([Unknown_Analysis, Text_Analysis])

#Got the Z-Scores from co-pilot
metrics = ['Simple TTR', 'Moving TTR', 'HDD', 'MTLD Moving']
for metric in metrics:
  mean = All_Analysis[All_Analysis['Label'] == 'Shakespeare'][metric].mean()
  std = All_Analysis[All_Analysis['Label'] == 'Shakespeare'][metric].std()
  All_Analysis[f'{metric} Z-Score'] = (All_Analysis[metric] - mean) / std
```

```
All_Analysis[All_Analysis['Label'] == 'unknown'].head(3)
```

	Title	Simple TTR	Moving TTR	HDD	MTLD Moving	Label	Simple TTR Z-Score	Moving TTR Z-Score	HDD Z-Score	MTLD Moving Z-Score
0	henry-vi-part-1	0.471589	0.471503	0.205755	13.229571	unknown	−1.053815	−1.068689	−0.055169	2.260837
1	arden-of-feversham	0.366878	0.435763	0.327606	10.649794	unknown	−5.446400	−2.565476	3.083721	−4.330837
2	edward-iii	0.538690	0.538619	0.191917	11.260721	unknown	1.761039	1.742057	−0.411632	−2.769838

```
#Shoutout to co-pilot for making this plot for me
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
fig.suptitle('Lexical Diversity Metrics for Shakespeare Plays', fontsize=16)

metrics = ['Simple TTR', 'Moving TTR', 'HDD', 'MTLD Moving']
for i, metric in enumerate(metrics):
  ax = axes[i // 2, i % 2]
    sns.histplot(data=All_Analysis,  x=metric,  hue='Label',  multiple='stack',
edgecolor='black', ax=ax)
  ax.set_title(metric)
  ax.grid(False)

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



Results

The SVM Model returned the following percentages for the three plays:

Henry VI Part 1: 83.22%

Arden of Feversham: 79.58%

Edward III: 78.15%

The Lexical Diversity analysis did offer some interesting insights into the three plays. The distribution of the the Simple TTR, Moving TTR and HDD seemed to be evenly distributed for the most part while the MTLD Moving average did display some skew.

Both Henry VI Part 1 and Edward III fell within the distribution of the known play. However, Arden of Feversham was a key outlier, most notably Simple TTR and the MTLD Moving Average.

Discussion

While the SVM Model was confident that Shakespeare was more likely to have written the three plays than Marlowe, this does not rule out potential co-authors entierly. There is the potential that other authors were attempting to imitate Shakespeare and succeeded in doing so well enough to fool the model. Because the plays overlapped in genre there could be common tropes that lead to a similar style as well. A more thorough analysis breaking down the plays act by act might be a way to narrow down which parts of the play were written by which author; if there were a major departure in style on an act by act basis, it might suggest that it was written by different authors. These acts could then be compared to known works.

The Lexical Diversity analysis did support the idea that Henry VI Part 1 and Edward the III were indeed written by Shakespeare as their Simple TTR, Moving TTR and HDD fall well within the distribution of known works. The only play that seemed to break normal trends was Arden of Feversham, which was multiple standard deviations away from Simple TTR, HDD and MTLD Moving Average. This suggests an increased chance that Arden of Feversham was likely co-authored, as there is an increased variety in the text.