```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
%matplotlib inline
```

```
# Read in the csv file using pandas
df = pd.read_csv('federalist.csv')
```

```
# Convert the author column to categorical data.
df['author'] = df.author.astype('category')
```

```
# Display the first few rows.
df.head()
```

	author	text
0	HAMILTON	FEDERALIST. No. 1 General Introduction For the...
1	JAY	FEDERALIST No. 2 Concerning Dangers from Forei...
2	JAY	FEDERALIST No. 3 The Same Subject Continued (C...
3	JAY	FEDERALIST No. 4 The Same Subject Continued (C...
4	JAY	FEDERALIST No. 5 The Same Subject Continued (C...

```
# Display the counts by author.
df['author'].value_counts()
```

```
HAMILTON               49
MADISON                15
HAMILTON OR MADISON    11
JAY                     5
HAMILTON AND MADISON    3
Name: author, dtype: int64
```

```
# Divide into train and test, with 80% in train. Use random state 1234.
from sklearn.model_selection import train_test_split
X = df.drop('author', axis=1)
y = df['author']

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=123
```

```
# Display the shape of train and test.
print("Train shape:", X_train['text'].shape)
```

✓ 0s completed at 10:04 PM

```
Train shape: (66,)
Test shape: (17,)
```

```
from nltk import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
stopwords = stopwords.words('english')

X_train = X_train['text']
X_test = X_test['text']

# Process the text by removing stop words and performing tf-idf vectorization
# fit to the training data only, and applied to train and test.
vectorizer = TfidfVectorizer(stop_words=stopwords)
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)

X_train_vectorized = X_train_vectorized.toarray()
X_test_vectorized = X_test_vectorized.toarray()


# Output the training set shape and the test set shape.
print("Train shape (after vectorization):", X_train_vectorized.shape)
print("Test shape (after vectorization):", X_test_vectorized.shape)
```

```
Train shape (after vectorization): (66, 7876)
Test shape (after vectorization): (17, 7876)
```

```
# Try a Bernoulli Naïve Bayes model. What is your accuracy on the test set?
from sklearn.naive_bayes import *
nb = MultinomialNB()
nb.fit(X_train_vectorized, y_train)
nb_score = nb.score(X_test_vectorized, y_test)
print("Naive Bayes accuracy:", nb_score)
```

```
Naive Bayes accuracy: 0.5882352941176471
```

```
# Redo the vectorization with max_features option set to use only the 1000 most frequent w
# In addition to the words, add bigrams as a feature.
vectorizer_limited = TfidfVectorizer(max_features=1000, stop_words=stopwords, ngram_range=
X_train_vectorized_limited = vectorizer_limited.fit_transform(X_train)
X_test_vectorized_limited = vectorizer_limited.transform(X_test)

X_train_vectorized_limited = X_train_vectorized_limited.toarray()
X_test_vectorized_limited = X_test_vectorized_limited.toarray()
```

```
# Try Naive Bayes again on the new train/test vectors and compare your results.
nb_limited = MultinomialNB()
nb_limited.fit(X_train_vectorized_limited, y_train)
nb_limited_score = nb_limited.score(X_test_vectorized_limited, y_test)
print("Naive Bayes (limited) accuracy:", nb_limited_score)
print("Naive Bayes' accuracy with the limited vectorizer is", nb_limited_score / nb_score,
```

```
Naive Bayes (limited) accuracy: 0.5882352941176471
Naive Bayes' accuracy with the limited vectorizer is 1.0 times as accurate as Naive I
```

```
# Try logistic regression. Adjust at least one parameter in the LogisticRegression() model
# can improve results over having no parameters. What are your results?
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression(random_state=101)
logmodel.fit(X_train_vectorized_limited, y_train)
LR_score = logmodel.score(X_test_vectorized_limited, y_test)
print("Logistic Regression accuracy:", LR_score)
```

```
Logistic Regression accuracy: 0.5882352941176471
```

```
# Try a neural network.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X_train_vectorized_limited)
X_train_scaled = scaler.transform(X_train_vectorized_limited)
X_test_scaled = scaler.transform(X_test_vectorized_limited)

from sklearn.neural_network import MLPClassifier
regressor = MLPClassifier(hidden_layer_sizes=(10,6), max_iter=5000)
regressor.fit(X_train_scaled, y_train)
regressor_score = regressor.score(X_test_scaled, y_test)
print("Regressor accuracy:", regressor_score)
```

```
Regressor accuracy: 0.7647058823529411
```

```
# Try different topologies until you get good results.
regressor = MLPClassifier(hidden_layer_sizes=(20,10), max_iter=5000)
regressor.fit(X_train_scaled, y_train)
regressor_score = regressor.score(X_test_scaled, y_test)
print("Regressor accuracy:", regressor_score)
```

```
Regressor accuracy: 0.7058823529411765
```

```
regressor = MLPClassifier(hidden_layer_sizes=(5,2), max_iter=5000)
regressor.fit(X_train_scaled, y_train)
regressor_score = regressor.score(X_test_scaled, y_test)
print("Regressor accuracy:", regressor_score)
```

```
Regressor accuracy: 0.5882352941176471
```

```
# What is your final accuracy?
regressor = MLPClassifier(hidden_layer_sizes=(12,8), max_iter=5000, random_state=8)
regressor.fit(X_train_scaled, y_train)
regressor_score = regressor.score(X_test_scaled, y_test)
print("Regressor accuracy:", regressor_score)
```

```
Regressor accuracy: 0.8823529411764706
```